Auto-generated Date: May 15, 2024

Intel® Quartus® Prime Pro Edition User Guides - Combined

This auto-generated document contains the following user guides. To download individual standalone documents, click on the respective PDF/HTML links.

Quartus® Prime Pro Edition User Guide

Getting Started

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

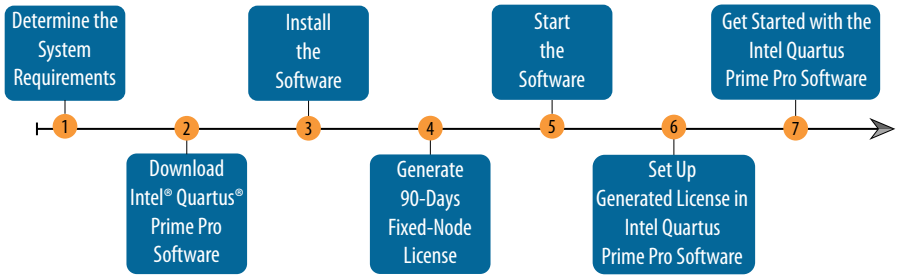
Contents

1. Introduction to Quartus® Prime Pro Edition

This user guide describes basic concepts, files, and design flow of the Quartus® Prime Pro Edition software, including initial design planning considerations, selecting a starting point to set up your Quartus Prime Pro Edition project and managing those projects and working with intellectual property (IP).

The Quartus Prime design suite is a comprehensive development platform to design with Intel FPGAs(1), from design entry and synthesis to optimization, verification, simulation, and binary generation. The Quartus Prime software supports fast design processing, straightforward device programming, and integration with other industry-standard EDA tools. The user interface makes it easy for you to focus on your design—not on the design tool. The modular compiler streamlines the FPGA development process and ensures the highest performance for the least effort.

Quartus Prime Pro Edition Software Quick Start Steps



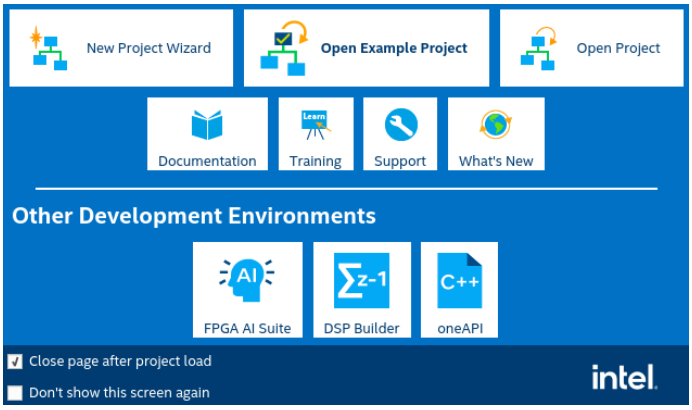
(1) A field-programmable gate array (FPGA) is a specialized integrated circuit that you can customize and reconfigure multiple times. To learn about and select a target Intel FPGA device family, refer to https://www.intel.com/content/www/us/en/products/details/fpga.html.

Use the following links to get started with the Quartus Prime Pro Edition software:

Table 1. Quartus Prime Pro Edition Software Quick Start Steps

Step		Useful Links
Determine the System Requirements	Verify hardware and software requirements and review Quartus Prime Pro Edition software release notes.	• Reviewing the Quartus Prime Installer Software Release Notes • Determining Hardware Requirements • Determining Software Requirements
Download the software	Download the latest Quartus Prime Pro Edition software.	• Quartus Prime Design Software Download Page • FPGA Software Download Center • Downloading Software Using the Quartus Prime Installer or Quartus Prime Installer (GUI mode) • Downloading Software Packages Manually
Install the software	Install the latest Quartus Prime Pro Edition software.	• Quartus Prime Installer (GUI mode) or Installing Intel® FPGA Software Through Quartus Prime Installer • Installing the Intel FPGA Software Manually • Setting Quartus Prime Environment Variables
Generate 90-days Fixed-node License	Generate 90- days free license in Intel FPGA Self-Service Licensing Center (SSLC).	• Intel FPGA Self-Service Licensing Center • Intel FPGA SSLC No-cost/Evaluation License Setup • Setting up Intel FPGA SSLC No-cost/ Evaluation License
Start the software	**On Windows:** Use one of the following options: • On the desktop, double-click the Intel FPGA software icon.  • Click **Start ➤ All Programs ➤ Intel FPGA *<Version>* Pro Edition ➤ Quartus (Quartus Prime Pro *<Version>*** • At the command prompt, type: `<installation-directory>\bin64\quartus` **On Linux:** • At the command prompt, type: `<installation-directory>/quartus/bin/quartus`	Starting the Quartus Prime Software
Set up the generated license in the Quartus Prime Pro Edition software	In the Quartus Prime Pro Edition software, click **Tools ➤ License Setup**, browse and select the license file, and click **OK**.	• Intel FPGA Self-Service Licensing Center (SSLC) • Setting up Intel FPGA SSLC No-cost/ Evaluation License • Summary of Intel FPGA Software Licenses Required • Licensing Intel FPGA Software Walkthrough

continued...

Step		Useful Links
Quartus Prime Software Home Page	Main page of the Quartus Prime software. 	
Plan FPGA Design for RTL Flow	Planning for RTL flow is an essential step for advanced FPGA design. Determining your design priorities early on helps you to choose the best device, tools, features, and methodologies for your design.	Plan FPGA Design for RTL Flow
Create your FPGA project	The Quartus Prime software makes it easy for you to quickly setup a new FPGA design project.	Creating a New FPGA Design Project on page 22

Quartus Prime Software Editions

The Quartus Prime Software is available in three editions based on your design requirements:

Table 2. Quartus Prime Software Editions

	Pro Edition	Standard Edition	Lite Edition
	Optimized to support the advanced features in Intel FPGAs and SoCs with the following device families: • Agilex™ 5 • Agilex 7 • Stratix® 10 • Arria® 10 • Cyclone® 10 GX	Includes extensive support for earlier device families in addition to the following device families: • Cyclone 10 LP • MAX® 10	An ideal entry point to Intel's high-volume device families and is available as a free download with no license file required.

Supported Features

The following is the Quartus Prime feature support matrix:

Figure 2. Quartus Prime Feature Support Matrix

Software Features	Intel Quartus® Prime Pro Edition	Intel Quartus® Prime Standard Edition
Intel Agilex® 5 Device Support	✔	
Intel Agilex® 7 Device Support	✔	
Intel Stratix® 10 Device Support	✔	
New Hybrid Placer & Global Router	✔	✔
New Timing Analyzer	✔	✔
New Physical Synthesis	✔	✔
Platform Designer (formerly Qsys)	✔	✔
Partial Reconfiguration	✔	
Block-Based (Hierarchical) Design Flows	✔	
Interface Planner (formerly BluePrint)	✔	
Incremental Fitter Optimization	✔	

The following features are only available in the Quartus Prime Pro Edition software:

Table 3. Supported Features of the Quartus Prime Pro Edition Software

Feature	Description
Hyper-Aware Design Flow	Use Hyper-Retiming to reach the highest performance in Agilex 5, Agilex 7, and Stratix 10 devices.
Advanced synthesis	Integrates new, stricter language parser supporting all major IEEE RTL languages, with enhanced algorithms, and parallel synthesis capabilities, and support for SystemVerilog 2009.
Hierarchical project structure	Preserves individual post-synthesis, post-placement, and post-place and route results for design instances. Optimizes without impacting other partition placement or routing.
Incremental Fitter Optimizations	Run and optimize Fitter stages incrementally. Each Fitter stage generates detailed reports.
Faster, more accurate I/O placement	Plan interface I/O in Interface Planner.
Platform Designer (Pro)	Builds on the system design and custom IP integration capabilities of Platform Designer (Standard). Platform Designer (Pro) introduces hierarchical isolation between system interconnect and IP components.
Block-Based Design Flows	Preserve and reuse design blocks at various stages of compilation.

Quartus Prime Pro Edition software does not support the following Quartus Prime Standard Edition features:

- I/O Timing Analysis
- NativeLink third party tool integration (other third-party tool integration available)
- Video and Image Processing Suite IP Cores
- Talkback features
- Various register merging and duplication settings
- Saving a node-level netlist as `.vqm` or RTL to schematic conversion

Supported Intel FPGA Developmental Tools

The Quartus Prime software suite supports the following Intel FPGA development tools:

Table 4. Intel FPGA Developmental Tools Supported by the Quartus Prime Software Suite

Tools	Description
Questa*-Intel FPGA Edition	Simulates FPGA designs using Intel-specific simulation libraries. It includes all features of Siemens EDA Questa* Core, including behavioral simulation, HDL test benches, and Tcl scripting.
Intel Advanced Link Analyzer	Analyzes jitter/noise and evaluates high-speed serial link performance. It is an ideal predesign tool supporting Intel FPGA IBIS-AMI standards and enhanced models to help you understand how Intel FPGA solutions can fit your system requirements.
Intel SoC FPGA Embedded Development Suite	A comprehensive tool suite for embedded software development on Intel SoC FPGAs.
Ashling* RiscFree* IDE for Intel FPGAs	Integrated development environment for creating embedded applications on the RISC-V-based Nios® V soft processors and the Arm*-based hard processor system.
Intel HLS Compiler	A high-level synthesis (HLS) tool that accepts untimed C++ code as an input and generates production-quality register transfer level (RTL) code optimized for Intel FPGAs. This tool accelerates verification time over RTL by raising the abstraction level for FPGA hardware design. Models developed in C++ have typically verified orders of magnitude faster than RTL.
DSP Builder for Intel FPGAs	Supports a model-based design flow from algorithms to hardware in a common environment.
Intel oneAPI Base Toolkit	Enables you to target FPGAs for heterogeneous acceleration and simulate entire system flows by abstracting some parts of the hardware.
Intel Simics® simulator for Intel FPGAs	A full-system simulator that supports defining, developing, and deploying virtual platforms.
FPGA AI Suite	Provides several components to help in enabling Artificial Intelligence (AI) and creating optimized Intel FPGA AI platforms efficiently.
Intel FPGA Power and Thermal Calculator	Estimates your design's power consumption and provides thermal design parameters for Intel FPGA devices, such as Agilex 5, Agilex 7, and Stratix 10.

1.1. Before You Begin

Before you get started with setting up your Quartus Prime Pro Edition project, review the following topics:

1.1.1. Prerequisite Knowledge and Training

Using the Quartus Prime software to create a basic FPGA design requires the following basic knowledge. There are several training modules available if you need help.

Prerequisite Knowledge

- Basic knowledge of digital logic design.
- Basic knowledge of how to describe a hardware design using VHDL, Verilog HDL, SystemVerilog, or EDA schematic tools.

Note: You can accelerate design creation and success by starting your design project from a pre-verified design example that targets an Intel FPGA development board, as Creating a New Project from a Design Example on page 24 describes.

Prerequisite Training

If you are new to FPGA or the Quartus Prime software, you can review the following training modules:

- Read Me First!
- How to Begin a Simple FPGA Design
- University Self-Guided Lab: Become an FPGA Designer in 4 Hours
- Beginner Workshop for Intel FPGAs
- University Self-Guided Lab: Introduction to FPGAs and the Quartus Prime Software
- Basics of Programmable Logic: History of Digital Logic Design
- Basics of Programmable Logic: FPGA Architecture
- The Quartus Prime Software: Foundation (Pro Edition) (Online Training)
- Instructor-Led Training: Using Quartus Prime Software
- Using the Quartus Prime Standard Edition Software: An Introduction
- Verilog HDL Basics
- Verilog HDL Advanced
- VHDL Basics
- SystemVerilog with the Quartus Prime Software
- Introduction to Tcl Scripting

Related Information

- Intel FPGA Software Installation and Licensing
- FPGAs for Dummies eBook

1.1.2. Navigate Content Through Tasks

Use the following navigation diagram to navigate this guide through user-tasks:

Table 6. Navigate Content Through Tasks

A	B	C	D
Plan FPGA Design for RTL Flow or Work With Intel FPGA IP Cores	Create a New Project or Migrate Your Existing Project to Quartus Prime Pro Edition Software	Manage Your Quartus Prime Pro Edition Projects	Review Next Steps

1.1.3. Acronyms

This document uses the following acronyms throughout:

Acronym	Meaning
ALR	Additional Logic Resources
DSE	Design Space Explorer
DSP	Digital Signal Processing
EDA	Electronic Design Automation
EDS	Embedded Design Suite
EPE	Early Power Estimator
FIFO	First In, First Out
FPGA	Field Programmable Gate Arrays
GUI	Graphical User Interface
HDL	Hardware Description Language
HTML	HyperText Markup Language
HPS	Hard Processor System
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
JTAG	Joint Test Action Group
LAB	Logic Array Block
LAI	Logic Analyzer Interface
MTBF	Mean Time Between Failures
PCB	Printed Circuit Board
PLD	Programmable Logic Devices
PLLs	Phase-Locked Loops
PTC	Power and Thermal Calculator
PVT	Process, Voltage, and Temperature
QSF	Quartus Settings File

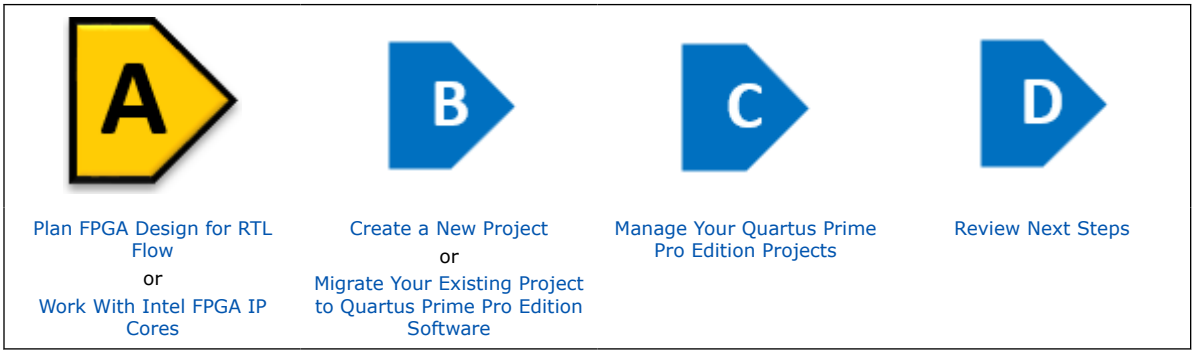
continued...

Acronym	Meaning
RAM	Random-Access Memory
RTL	Register-Transfer Level or Register-Transfer Logic
SDC	Synopsys* Design Constraints
Tcl	Tool Command Language
UART	Universal Asynchronous Receiver-Transmitter
VCS	Verilog Compiler and Simulator
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VPN	Virtual Private Network
VREF	Voltage Reference

2. Planning FPGA Design for RTL Flow

Navigating Content Through Tasks

Use the following navigation diagram to navigate this guide through user-tasks:



Planning for RTL flow is an essential step for advanced FPGA design. This chapter provides some useful tips and programming methods to consider in your planning process to help you detect and solve potential problems early in the design cycle. Determining your design priorities early on helps you to choose the best device, tools, features, and methodologies for your design.

Review the following topics to help you get started with the planning process:

2.1. Design Planning

Design planning is an essential step in advanced FPGA design. System architects must consider the target device characteristics in order to plan for interface I/O, integration of IP, on-chip debugging tools, and use of other EDA tools. Designers must consider device power consumption and programming methods when planning the layout. You can solve potential problems early in the design cycle by following the design planning considerations in this chapter.

By default, the Quartus Prime software optimizes designs for the best overall results. However, you can adjust settings to better optimize one aspect of your design, such as performance, routability, area, or power utilization. Consider your own design priorities and trade-offs when reviewing the techniques in this chapter. For example, certain device features, density, and performance requirements can increase system cost. Signal integrity and board issues can impact I/O pin locations. Power, timing performance, and area utilization all affect one another. Compilation time is affected when optimizing these priorities.

Determining your design priorities early on helps you to choose the best device, tools, features, and methodologies for your design.

Table 7. Checklist for Design Planning

Item	Considerations	Links
Create a design specification and test plan	• Create detailed design specifications that define the system. • Specify the I/O interfaces for the FPGA. • Identify the different clock domains. • Include a block diagram of basic design functions. • Create a test plan for verification and ease of manufacture. • Consider a common design directory structure or source control system to make design integration easy. • Consider whether you want to standardize on an interface protocol for each design block.	
Planning for Target Device or Board	• Refer to the Product Selector tool to compare the specifications and features of Intel FPGA devices and development kits. • Refer to the device family documentation for detailed device characteristics. View a summary of each device's resources by selecting a device in the Device dialog box (**Assignments ➤ Device**). • Consider whether the device family meets your design requirements for high-speed transceivers, global or regional clock networks, and the number of phase-locked loops (PLLs). • Consider the density requirements of your design. — Devices with more logic resources and higher I/O counts can implement larger and more complex designs, but at a higher cost. — Smaller devices use lower static power. — Select a device larger than what your design requires if you may want to add more logic later in the design cycle, or to reserve logic and memory for on-chip debugging. • Consider requirements for types of dedicated logic blocks, such as memory blocks of different sizes, or digital signal processing (DSP) blocks to implement certain arithmetic functions. • Alternatively, create a system that targets a specific development board to accelerate the process of appropriately configuring, connecting, and validating IP for the target board.	• Product Selector Guide Tool • Using the Board-Aware Flow
Planning for Device Migration	• Determine whether you want to migrate your design to another device density to allow flexibility when your design nears completion. • Target a small (less expensive) device and move to a larger device if necessary to meet your design requirements. • Develop a prototype of your design in a larger device to reduce optimization time and achieve timing closure more quickly, and then migrate to a smaller device after prototyping. *Note:* Selecting a migration device impacts pin placement because some pins may serve different functions in different device densities or package sizes.	
Planning for Intellectual Property (IP) Cores	Plan which I/O interfaces or blocks in the system you want to implement using IP cores. Integrate functions into your design using Intel FPGA IP cores, many of which are available for production use in the Quartus Prime software without additional license.	• Working With Intel FPGA IP Cores on page 50 • Intel FPGA IP Portfolio Web Page

continued...

Item	Considerations	Links
	For IP cores that require additional licenses for production, use the Intel FPGA IP Evaluation Mode, which allows you to program the FPGA to verify the IP in the hardware before you purchase the IP license.	
Planning for Standard Interfaces	• Use standard interfaces in system design to ensure compatibility between design blocks from different design teams or vendors. • Use the Quartus Prime Interface Planner to accurately plan constraints for design implementation, prototype interface implementations, and rapidly define a legal device floorplan. • Use the Quartus PrimePlatform Designer system integration tool to use standard interfaces and speed-up system-level integration.	Quartus Prime Pro Edition User Guide: Platform Designer
Planning for Device Power Consumption	Estimating power consumption early in the design cycle allows you to plan power budgets and avoid unexpected results when designing the PCB. • Use the Early Power Estimator (EPE) spreadsheet for older devices, such as the Arria 10 and Cyclone 10 families or to estimate power consumption before compiling or creating any source code. • Use the Quartus Prime Power Analyzer to ensure that your design satisfies thermal and power supply requirements. • Use the Intel FPGA Power and Thermal Calculator (PTC) to estimate power utilization for your design for Stratix 10, Agilex 7, and Agilex 5 device families. PTC does not support older devices.	• Quartus Prime Pro Edition User Guide: Power Analysis and Optimization • Intel FPGA Power and Thermal Calculator User Guide
Planning for I/O Interfaces	• Create a preliminary pin-out for an Intel FPGA with the Quartus Prime Pin Planner before you develop the source code, based on standard I/O interfaces (such as memory and bus interfaces) and any other I/O requirements for your system. • Configure how to connect the functions and cores to each other by specifying matching node names for selected ports. • Create other I/O-related assignments for these interfaces or other design I/O pins in the Pin Planner. • Compile your design to automatically run I/O Assignment Analysis in Fitter to validate I/O-related assignments that you created or modified throughout the design process.	• Quartus Prime Pro Edition User Guide: Design Optimization • I/O Planning Overview
Planning for Other EDA Tools	• Use supported standard third-party EDA synthesis tools to synthesize your Verilog HDL or VHDL design, and compile the resulting output netlist file in the Quartus Prime software. • Use the simulator version that your Quartus Prime software version supports for best results. You must also use the model libraries provided with your Quartus Prime software version. Libraries can change between versions, which might cause a mismatch with your simulation netlist.	• Quartus Prime Pro Edition User Guide: Third-party Synthesis • Quartus Prime Pro Edition User Guide: Third-party Synthesis

continued...

Item	Considerations	Links
Planning for On-Chip Debugging Tools	• Consider whether to include on-chip debugging tools early in the design process. Adding the debugging tools late in the design process can be more time-consuming and error-prone. • Consider the following debugging requirements when planning your design to support debugging tools: — **JTAG connections**—required to perform in-system debugging with JTAG tools. Plan your system and board with JTAG ports that are available for debugging. — **Additional logic resources (ALR)**—required to implement JTAG hub logic. If you set up the appropriate tool early in your design cycle, you can include these device resources in your early resource estimations to ensure that you do not overload the device with logic. — **Reserve device memory**—required if your tool uses device memory to capture data during system operation. To ensure that you have enough memory resources to take advantage of this debugging technique, consider reserving device memory to use during debugging. — **Reserve I/O pins**—required if you use the Logic Analyzer Interface (LAI), which requires I/O pins for debugging. If you reserve I/O pins for debugging, you do not have to later change your design or board. The LAI can multiplex signals with design I/O pins if required. Ensure that your board supports a debugging mode, in which debugging signals do not affect system operation. — **Instantiate an IP core in your HDL code**—required if your debugging tool uses an Intel FPGA IP core. — **Instantiate the Signal Tap Logic Analyzer IP core**—required if you want to manually connect the Signal Tap Logic Analyzer to nodes in your design and ensure that the tapped node names do not change during synthesis.	• Factors to Consider When Using Debugging Tools During Design Planning Stages • Quartus Prime Pro Edition User Guide: Debug Tools
Planning HDL Coding Styles	• Use synchronous design practices to consistently meet your design goals. In a synchronous design, a clock signal triggers all events. When you meet all register timing requirements, a synchronous design behaves in a predictable and reliable manner for all process, voltage, and temperature (PVT) conditions. You can easily target synchronous designs to different device families or speed grades. *Note:* Problems with asynchronous design techniques include reliance on propagation delays in a device, incomplete timing analysis, and possible glitches. • Use dedicated clock pins and clock routing for best results, and if you have PLLs in your target device, use the PLLs for clock inversion, multiplication, and division. • For clock multiplexing and gating, use the dedicated clock control block or PLL clock switchover feature instead of combinational logic, if these features are available in your device. If you must use internally generated clock signals, register the output of any combinational logic used as a clock signal to reduce glitches.	• Quartus Prime Pro Edition User Guide: Design Recommendations • Quartus Prime Pro Edition User Guide: Timing Analyzer

continued...

Item	Considerations	Links
	• Consider the architecture of the device you choose so that you can use specific features in your design. For example, the control signals should use the dedicated control signals in the device architecture. Sometimes, you might need to limit the number of different control signals used in your design to achieve the best results. • HDL coding styles can have a significant effect on the quality of results for programmable logic designs. Follow the coding guidelines for inferring Intel FPGA IP and targeting dedicated device hardware, such as memory and DSP blocks. • Ensure that your design accounts for synchronization between any asynchronous clock domains. Consider using a synchronizer chain of more than two registers for high-frequency clocks and frequently-toggling data signals to reduce the chance of a metastability failure. • Use the Quartus Prime software to analyze the average mean time between failures (MTBF) due to metastability when a design synchronizes asynchronous signals, and optimize your design to improve the metastability MTBF.	
Planning your Project Path Length	Design files with lengthy file paths might cause an internal error in the Windows* version of the Quartus Prime Pro Edition software. Windows has a 260-character maximum path length limitation on the combined length of a file name and its file path. To reduce the length of a file path to a design file, Intel strongly recommends creating, storing, or moving your Quartus Prime project to a shorter path. For example: `C:\quartus_pro\<version>\project` *Tip:* There are several third-party software freely available to help you fix the long path issue for Windows.	Maximum Path Length Limitation

Table 8. Factors to Consider When Using Debugging Tools During Design Planning Stages

Design Planning Factor	Signal Tap Logic Analyzer	System Console	In-System Memory Content Editor	Logic Analyzer Interface (LAI)	Signal Probe	In-System Sources and Probes	Virtual JTAG IP Core
JTAG connections	Yes	Yes	Yes	Yes	—	Yes	Yes
Additional logic resources	—	Yes	—	—	—	—	Yes
Reserve device memory	Yes	Yes	—	—	—	—	—
Reserve I/O pins	—	—	—	Yes	Yes	—	—
Instantiate IP core in your HDL code	—	—	—	—	—	Yes	Yes

2.2. Selecting the Design Methodology

When creating an FPGA design, you must consider various design methodologies the Quartus Prime software offers, such as the incremental block-based design, flat design, and partial reconfiguration design. You can use these design flows with or without EDA design entry and synthesis tools. Refer to the following topics for more information about each of these methodologies.

2.2.1. Flat Design Vs. Incremental Block-based Design

With the Quartus Prime Pro Edition software, you can either develop a flat design or a block-based design:

- **Flat Designs**: In a flat compilation flow, the design hierarchy is flattened without design partitions and the Quartus Prime software compiles the entire design in a "flat" netlist. Although the source code may be hierarchical, the compiler flattens and synthesizes all the design logic. Whenever you recompile the project, the compiler re-performs all available logic and placement optimizations on the entire design.

  The flat compilation flow does not require any planning for design partitions. However, because the Quartus Prime software recompiles the entire design whenever you change your design, flat design practices may require more overall compilation time for large designs. Additionally, you may find that the results for one part of the design change when you change a different part of your design.

  If you plan to develop a small design with no plans to reuse or preserve blocks, use a flat design. However, flat designs are generally more difficult to optimize and debug because you cannot always isolate the timing issue.

- **Block-based designs**: In block-based (hierarchical) flows, you can divide your design by creating design partitions. Block-based design flow is also known as modular or hierarchical design flow. You can designate a design block as a design partition to preserve or reuse the block. A design partition is a logical, named, hierarchical boundary assignment that you can apply to a design instance. Hierarchical flows allow you to isolate, optimize, and preserve compilation results for specific design blocks but require more design planning to ensure effective results.

  Using a hierarchical design methodology offers several advantages, such as:

  — Reuse design blocks with the same periphery configuration, share a synthesized design block with another designer, or replicate placed and routed IP in another project.

  — Design, implement, and verify core or periphery blocks once, and then reuse those blocks multiple times across different projects that use the same device.

  — Perform easier debugging and optimization of individual design blocks.

  — Assign the design hierarchy elements into logical partitions that are functionally independent.

  — Perform stand-alone block verification.

  — Use design blocks for reuse and preserve synthesis and timing results for blocks that are fully coded and meeting timing.

  — Preserve earlier results for a block you do not want to change when you change RTL code or compiler settings for another block in the design. The compiler produces different compilation results compared to previous settings and can cause timing violations in blocks that do not reflect the same corresponding code or setting changes. Block-based incremental compilation flow allows for preserving the block.

  — Partition a design, compile the design partitions separately, and reuse the results for unchanged partitions. You can preserve the performance of unchanged blocks and reduce the number of design iterations. The performance preservation of incremental block-based compilation allows you to focus timing closure on unpreserved partitions or on blocks that have difficulty meeting timing requirements

For more information about the block-based designing, refer to the *Quartus Prime Pro Edition User Guide: Block-Based Design*.

Related Information

Quartus Prime Pro Edition: Block-Based Design

2.2.2. Partial Reconfiguration Design

Partial reconfiguration (PR) allows you to reconfigure a portion of the FPGA dynamically while the remaining FPGA design continues to function. You can define multiple personas for a particular region in your design without impacting operation in areas outside this region. This methodology is effective in systems with various functions that time-share the same FPGA device resources. PR enables the implementation of more complex FPGA systems.

The Quartus Prime Pro Edition software supports the PR feature for the Agilex 5, Agilex 7, Stratix 10, Arria 10, and Cyclone 10 GX device families.

PR provides the following advantages over a flat design:

- Allows run-time design reconfiguration.
- Increases scalability of the design through time-multiplexing.
- Lowers cost and power consumption through efficient use of board space.
- Supports dynamic time-multiplexing functions in the design.
- Improves initial programming time through smaller bitstreams.
- Reduces system downtime through line upgrades.
- Enables easy system updates by allowing remote hardware change.
- Supports a simplified compilation flow for partial reconfiguration.

For more information about the PR design, refer to the *Quartus Prime Pro Edition User Guide: Partial Reconfiguration*.

Related Information

Quartus Prime Pro Edition: Partial Reconfiguration

2.3. Related Trainings

You can take up the following training to help you when planning FPGA design for RTL flow:

- Creating Reusable Design Blocks: Introduction to IP Reuse with the Quartus Prime Software
- Design Block Reuse in the Quartus Prime Pro Edition Software
- Incremental Block-Based Compilation in the Quartus Prime Pro Edition Software: Introduction
- Incremental Block-Based Compilation in the Quartus Prime Pro Edition Software: Design Partitioning
- Incremental Block-Based Compilation in the Quartus Prime Pro Edition Software: Timing Closure & Tips
- Partial Reconfiguration in Quartus Prime
- Partial Reconfiguration for Intel FPGA Devices: Introduction & Project Assignments

- Partial Reconfiguration for Intel FPGA Devices: Design Guidelines & Host Requirements
- Partial Reconfiguration for Intel FPGA Devices: PR Host IP & Implementations
- Partial Reconfiguration for Intel FPGA Devices: Output Files & Demonstration

3. Selecting a Starting Point for Your Quartus Prime Pro Edition Project

Navigating Content Through Tasks

Use the following navigation diagram to navigate this guide through user-tasks:

There are multiple ways you can set up your project depending on your design needs Either you can create a new project with project files and libraries or with a design example, or you can import an existing project into the Quartus Prime Pro Edition software.

Select one of the following methods to set up your Quartus Prime Pro Edition project:

3.1. Creating a New FPGA Design Project

The Quartus Prime software makes it easy for you to quickly setup a new FPGA design project.

Use the quick access icons on the home page to set up your FPGA design project:

- **New Project Wizard**: Use this option to create a project either by specifying project files, libraries, target device family, and EDA tool settings, or using an existing design example.
- **Open Example Project**: Use this option if you want to base your current project on a pre-installed and verified design example.
- **Open Project**: Use this option to browse and open your existing projects.

The wizard guides you through specifying various options for new project setup and includes access to helpful project templates and design examples that allow you to preconfigure project settings for specific applications, FPGA devices, and target boards.

3.1.1. Using the Board-Aware Flow

The Quartus Prime Pro Edition software allows you to create a system that targets a specific development board, rather than only targeting a specific FPGA device. When you target a specific development board, the Quartus Prime software is *aware* of the target board (board-aware) which accelerates the process of appropriately configuring, connecting, and validating IP for the target board.

What is the Quartus Prime Software Board-Aware Flow?

In the board-aware flow, you can optionally start your project from a pre-verified design example (rather than an empty project) and target a specific Intel FPGA development board. You can also create appropriate IP presets to target the specific board. The Quartus Prime Platform Designer system integration tool is also board-aware, allowing you to automatically set pin assignments and export appropriate system interfaces for the target board.

The board-aware flow simplifies the application of appropriate parameters and pin assignments for the instantiated IP in your project, thereby reducing the chance of configuration errors. You can also save and reuse your preferred and verified board and IP configurations for use in other projects that target the same IP or board.

The board-aware flow helps to ensure the proper hand-off, consistency, and reuse of configuration options across multiple projects, developers, and boards.

Note: To define new boards and IP preset files in Platform Designer, refer to *Intel Quartus Prime Pro Edition User Guide: Platform Designer* and *AN 988: Using the Board-Aware Flow in the Intel Quartus Prime Pro Edition Software*.

Related Information

- Creating a New Project from a Design Example on page 24
- Specifying a Target Board for the Project on page 32
- Applying Preset Parameters for Specific Applications on page 61
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
- AN 988: Using the Board-Aware Flow in the Intel Quartus Prime Pro Edition Software

3.1.1.1. Creating a New Project from a Design Example

The Quartus Prime software provides access to installed and online platform- and board-specific design examples that you can use as a starting point for your own design. You can accelerate your design progress by starting from a pre-validated design example that installs with the Quartus Prime software or is available online.

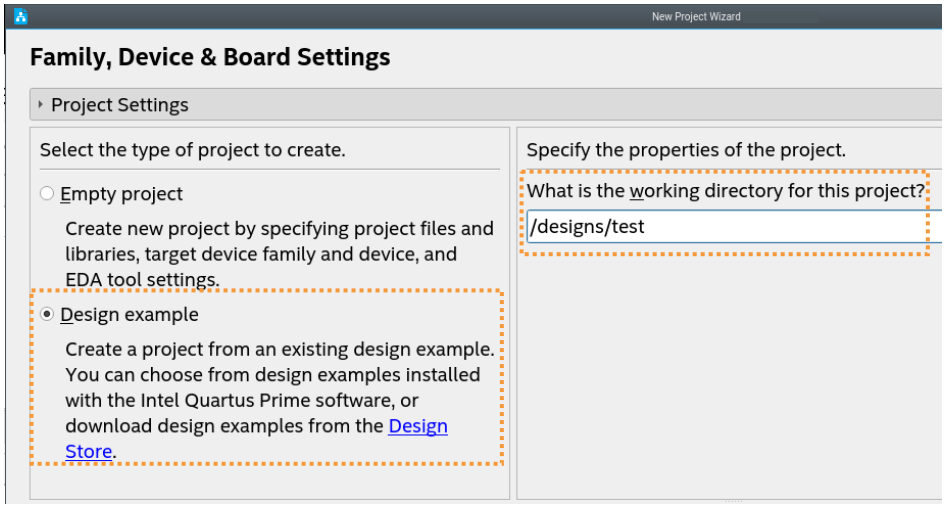
This technique can be especially helpful if you are new to FPGA design or EDA design tools. The design example can help you to quickly analyze a validated design on a board and appropriately configure it in various ways to match your users' needs. Alternatively, you can start with an **Empty Project** for which you specify all settings and design files.

- **Pre-installed design examples**—you can immediately access the design examples that install along with the Quartus Prime software installation at: `<quartus>\acds\quartus\common\board_designs`.
- **Online design examples**—you can access design examples hosted online, which includes designs from the Intel FPGA Design Store or directly from Quartus Prime software by clicking **Open Example Project** from the home page. For more information, refer to Design Example Discovery.
- **Downloaded design examples**—you can access your previously downloaded design examples, or any design example that you store in a local drive, under downloaded reference designs.

To create a new Quartus Prime project that is based on a design example, follow these steps:

1. the Quartus Prime software, click **File ➤ New Project Wizard**. Click **Next** to view the **Family, Device & Board Settings** wizard page.
2. Under the **Select the type of project to create**, select **Design Example** and click **Next**. The **Family, Device & Board Settings** page appears, allowing you to find and select the design example from which to base your project.

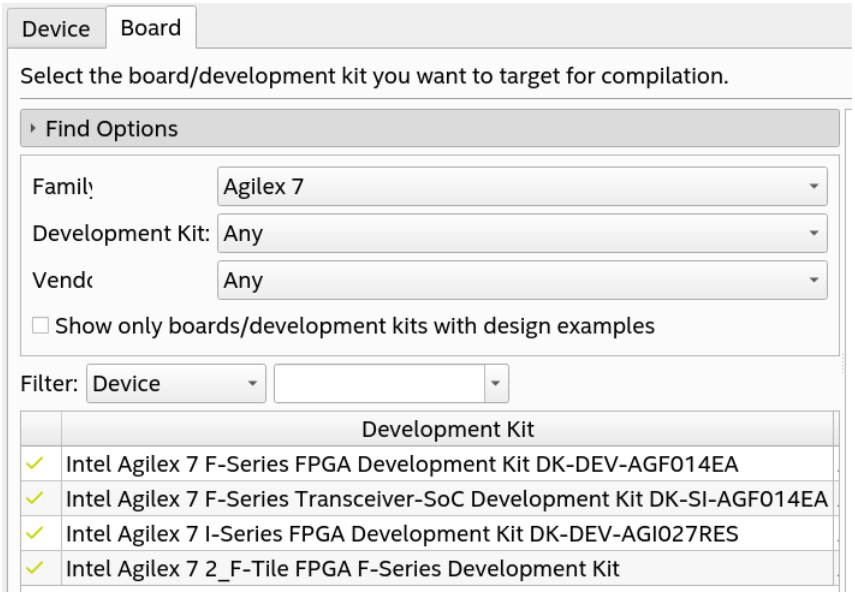
Figure 4. Family, Device & Board Settings Page of New Project Wizard



3. Under **What is the working directory for this project?**, specify the directory to store your project files and click **Next**.

4. Under **Find Options**, select the **Family**, **Development Kit**, and **Vendor** design example you want to use. Refer to Family, Device & Board Settings on page 25.

Figure 5. Board Tab in New Project Wizard



The search results display the design examples that meet your search criteria.

5. Select the design example that you want in the search results and click **Next**. If the design example is licensed by Intel FPGA, a **Software License Agreement** page appears that prompts you to accept the license agreement before you can proceed.
6. Click **Next** to proceed to the **Summary** page.
7. Click **Finish** to deploy the selected design example in the Quartus Prime software. When a design example downloads, the design's `.par` downloads to the download path that you define in **More Settings**, but the design itself extracts to the project working directory that you specify.

Also refer to Accessing Online Design Examples on page 27 and Accessing Downloaded Design Examples on page 31.

Related Information

Intel FPGA Design Examples

3.1.1.1.1. Family, Device & Board Settings

The following options are available in the **Family, Device & Board Settings** page of the **New Project Wizard**. Specify these options to locate and deploy a validated design example targeting a specific board as a starting point for your FPGA design project. Some options are only available from **File ➤ Open Example Project**

Table 9. Family, Device & Board Settings Page Options

Option	Description
Select the type of project to create	• **Empty**—create a new empty FPGA design project to which you add all design files, settings, and constraints. • **Design example**—create a new project from an existing design example. You can access installed or online available design examples.
What is the working directory for this project?	Specifies the directory where you want to extract and deploy the design example.
Find Options	Allows you to filter design example search results by one of the following facets: • **Load from** >**Pre-installed design examples**—specifies that search includes examples installed with the Quartus Prime software. • **Load from** > **User downloaded design examples**—search includes design examples that you download or your own design examples that you store in a local repository. • **Load from** > **Online design examples**—search includes design examples hosted online, including examples from the Intel FPGA Design Store. • **Family**—search only includes design examples for the device families that you specify. You can specify multiple values. • **Quartus Prime version**—search only includes design examples that support the Quartus Prime software version that you specify. You can specify multiple pipe separated values. • **Development kit**—search only includes design examples that support the Intel FPGA development kit that you specify. You can specify multiple pipe separated values.
More settings button	Opens the **Options** panel that allows you to configure the **Internet Connectivity** and **Design Examples** connection and download settings, as Design Examples Options on page 30 describes.
Reset button	Resets the **Find Options** to default settings.
Legend panel	Displays the meaning of design example status icons in the search results. Design example status is validated (checkmark icon), unvalidated (question mark icon), or unsupported for the current Intel Quartus Prime software version (x icon).
Filter text box	Specifies a text string to further filter design example search results according to any text string you specify.
Search results list	Displays the design examples, status, and location that match your search filters.
Details panel	Displays a detailed description and diagram of the selected design example.
Design Store button	Opens the Design Store website in your default web browser from which you can download available Intel FPGA validated design examples.

3.1.1.1.2. Accessing Pre-Installed Design Examples

The Quartus Prime software installation includes design examples for your immediate use.

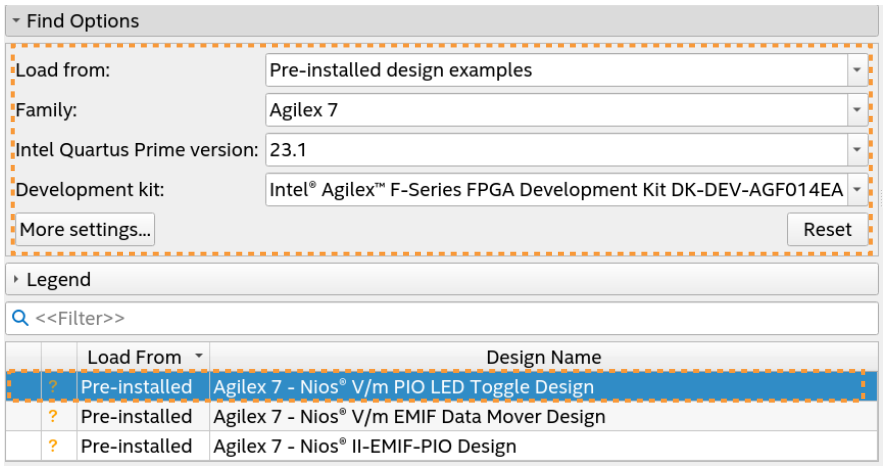
You can access the pre-installed design examples while using the Quartus Prime software using the following methods:

- Click **File ➤ New Project Wizard ➤ Family, Device & Board Settings** page.
- Click **Open Example Project** on the **Home** tab (**Help ➤ Home**).
- Click **File ➤ Open Example Project**.

To create a new project based on pre-installed design examples, follow these steps Quartus Prime Pro Edition software:

1. Click **File ➤ Open Example Project**. The **Design Example** page of the **New Project Wizard** opens.
2. For **What is the working directory for this project?**, specify the directory location to store your project files.

Figure 6. Find Options Locate Pre-Installed Design Example



3. Under **Find Options**, specify the following settings to filter the list of design examples for the target device and board. Also refer to Family, Device & Board Settings on page 25.
   a. In **Load from**, select the **Pre-Installed design examples**, repository.
   b. In **Family**, select your target FPGA device family.
   c. In **Intel Quartus Prime version**, select the software version.
   d. In **Development kit**, select the target kit or board.
4. Under **Design name**, select the design example to base your project on.
5. Click **Next**, and then click **Finish**. The design extracts to the working directory and opens in the Quartus Prime software.

Related Information

Design Example Discovery

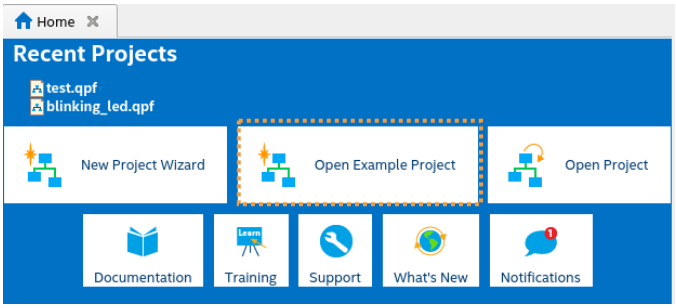
3.1.1.1.3. Accessing Online Design Examples

You can create a new project based on a design example that you access from an online repository. To use this method, you may need to specify a proxy server for access and the download path.

To create a new project in the Intel Quartus Prime software based on online design examples, follow these steps:

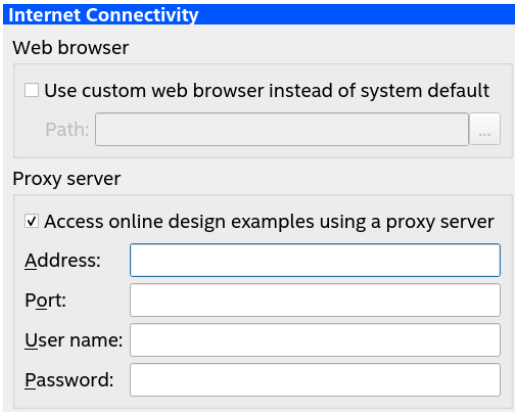
1. Click **File ➤ Open Example Project**. The **Design Example** page of the **New Project Wizard** opens.

Figure 7. Open Example Project Icon on Home Page



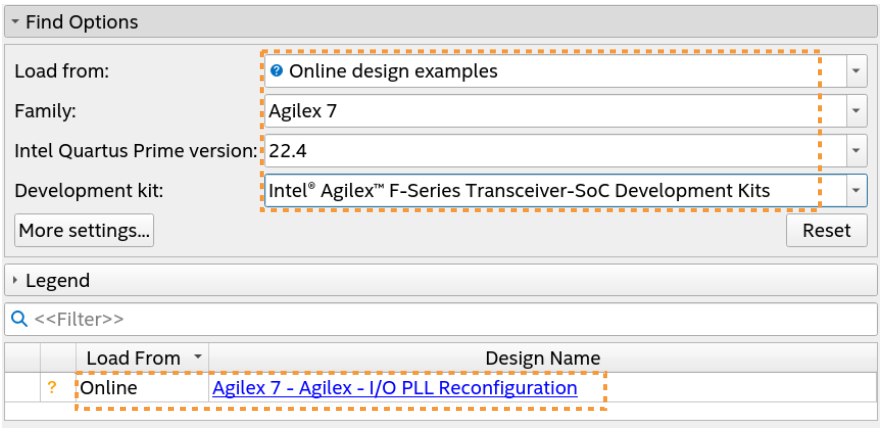
2. For **What is the working directory for this project?**, specify the directory location to store your project files.
3. Click the **More Settings** button. The **Options** dialog box opens with the **Internet Connectivity** tab open by default.

Figure 8. Intel Quartus Prime Software Internet Connectivity Settings



4. If your internet connection requires a proxy server (using VPN), turn on the **Access online design examples using a proxy server** option, and then specify your proxy **Address**, **Port**, **User name**, and **Password**. If your internet connection does not require a proxy server, skip this step.
5. On the **Design Example Search Locations** tab, specify the **Download path** for download of the design example `.par` file.
6. Click **OK**.
7. Under **Find Options**, specify the following settings. Also refer to Family, Device & Board Settings on page 25.
   a. In **Load from**, select **Downloaded design examples**.
   b. In **Family**, **Intel Quartus Prime version**, and **Development kit** fields select the values to match your target design and board.
8. In the design example list, select the design that you want to deploy.

Figure 9. Online Agilex 7 - I/O PLL Reconfiguration Design



9. Click **Next**, and then click **Finish**. The design extracts to the working directory and opens in the Quartus Prime software.

Internet Connectivity Options

You can specify the following internet connectivity options that determine how the Quartus Prime software connects to the internet for various functions, such as accessing Help and design examples, with either of the following:

- Click **Tools ➤ Options ➤ Internet Connectivity**

  Or

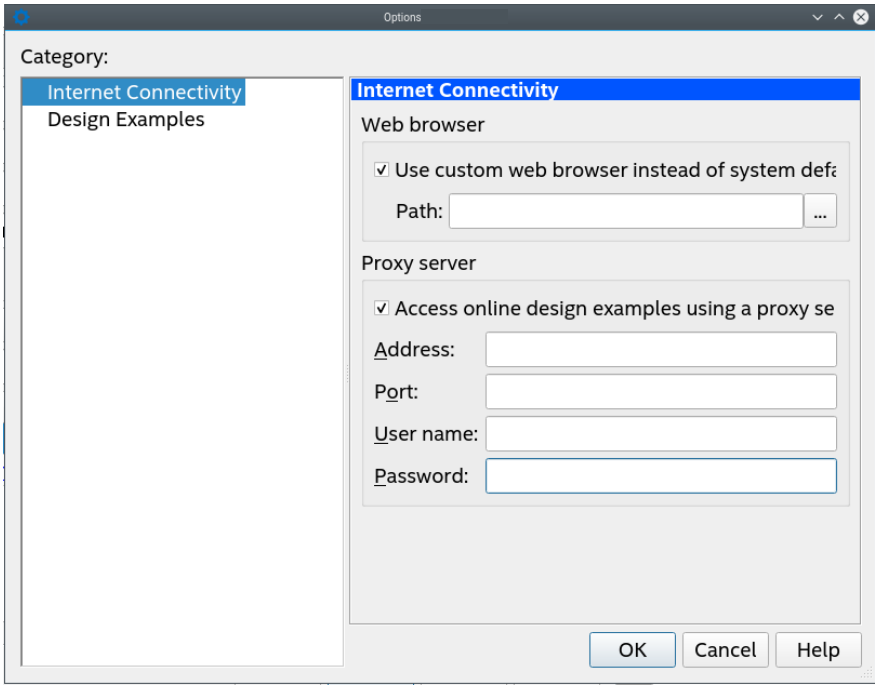
- Click **More Settings** on the **Design Example** page of the **New Project Wizard** (**File ➤ New Project Wizard**).

The following options are available on the **Internet Connectivity** options page.

Table 10. Internet Connectivity Options

Option	Description
Web browser	Specifies the web browser that deploys when the Quartus Prime software accesses the internet, including the Intel FPGA Design Store web page. Enable **Use custom web browser** to specify the path to your preferred supported web browser.
Proxy server	Specify options if connecting to the internet through a proxy server. To access online design examples specify the appropriate option: • **Access online design examples using a proxy server**—turn on this option if you are connected to the internet through a VPN. Turn off this option if you are not connected to the internet through a VPN (such as connection through a private network).

Figure 10. Internet Connectivity Page



Related Information

Design Example Options Page on page 30

Design Examples Options

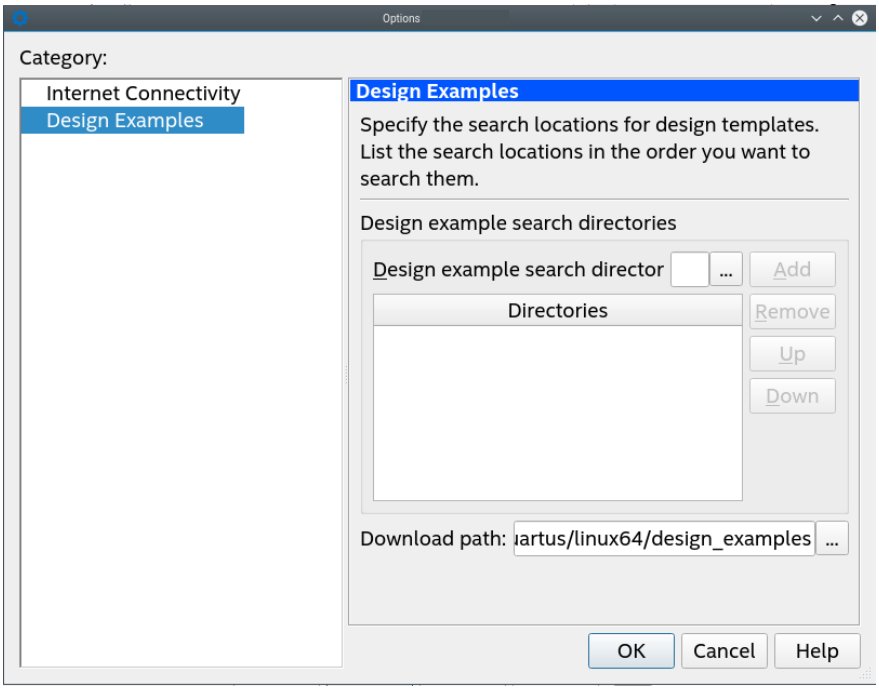
You can click the following to specify options that determine how the Quartus Prime software accesses available design examples.

- Click **File ➤ New Project Wizard** and then click the **More Settings** button on the **Design Example** page of the **New Project Wizard**.

Table 11. Design Examples Options

Option	Description
Design Example search directory	Specifies the local directories that the Quartus Prime software searches for design examples. This setting determines which directories you include in search when using the **New Project Wizard** to start a project from an existing design example. Click **Add**, **Remove**, **Up**, or **Down** to change the search order and contents in the **Directories** list.
Directories	Lists the various directories that you include in the design example search path for the **New Project Wizard**.
Download path	Specifies the path for download of online design examples.

Figure 11. Design Examples Page



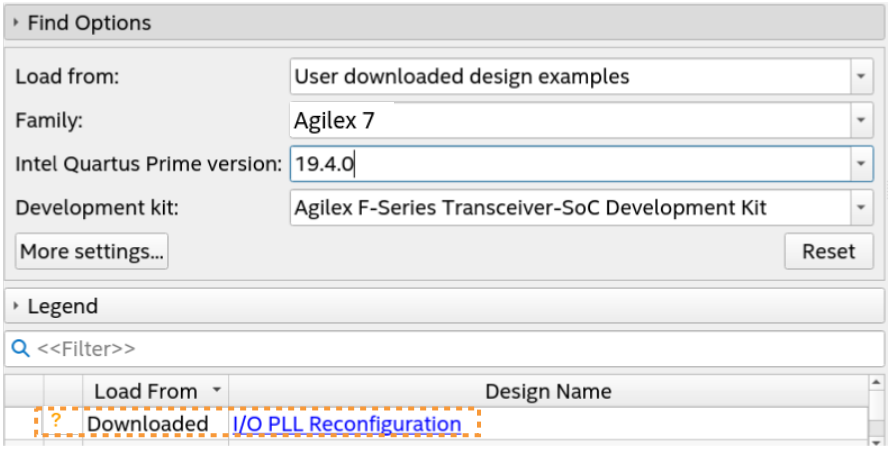
3.1.1.1.4. Accessing Downloaded Design Examples

You can create a new project from a design example that you have previously downloaded. Download a design example `.par` file from an online repository (such as the Intel FPGA Design Store) into your working directory. Designs that you create yourself and store in a local drive also appear as downloaded examples.

To create a new project based on downloaded design examples, follow these steps:

1. Download a design example, as Accessing Online Design Examples on page 27 describes.
2. Click the **Open Example Project** icon on the Quartus Prime Pro Edition **Home** page. The **Design Example** page of the **New Project Wizard** opens.
3. For **What is the working directory for this project?**, specify the directory location to store your project files.
4. Under **Find Options**, specify the following settings. Refer to Family, Device & Board Settings on page 25.
   a. In **Load from**, select **Downloaded design examples**.
   b. In **Family**, **Intel Quartus Prime version**, and **Development kit** fields select the values to match your target design and board.
5. In the design example list, select the design that you want to deploy.

Figure 12. Downloaded Agilex 7 I/O PLL Reconfiguration Design



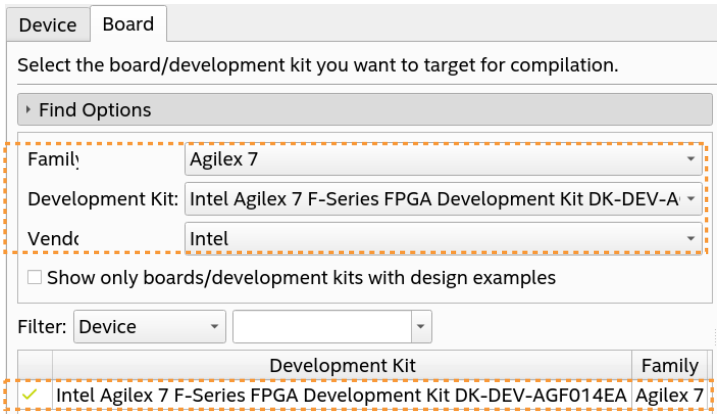
6. Click **Next**, and then click **Finish**. The design extracts to the working directory and opens in the Quartus Prime software.

3.1.1.2. Specifying a Target Board for the Project

You can specify the target for a new project in the New Project Wizard, or you can specify a target board for an existing project by clicking **Assignments ➤ Device**. To specify a target board for an existing project, follow these steps:

1. Click **Assignments ➤ Device**. The **Device** dialog box appears.
2. Click the **Board** tab. The **Board** tab allows you to target a specific FPGA device board, rather than just a specific FPGA device.

Figure 13. Board Tab Settings



3. In the **Family**, **Intel Quartus Prime version**, and **Development kit** fields, select the values to match your target design and board.
4. Click the desired board in the list. The board details appear in the right pane.
5. Click **OK**. Your project now targets the specified board and device.

3.2. Migrating Projects from Other Quartus Prime Editions to Quartus Prime Pro Edition

The Quartus Prime Pro Edition software supports migration of Quartus Prime Standard Edition, Quartus Prime Lite Edition, and Quartus II software projects.

Note: The migration steps for Quartus Prime Lite Edition, Quartus Prime Standard Edition, and the Quartus II software are identical. For brevity, this section refers to these design tools collectively as "other Quartus software products."

Migrating to Quartus Prime Pro Edition requires the following changes to other Quartus software product projects:

1. Upgrade project assignments and constraints with equivalent Quartus Prime Pro Edition assignments.
2. Upgrade all Intel FPGA IP core variations and Platform Designer systems in your project.
3. Upgrade design RTL to standards-compliant VHDL, Verilog HDL, or SystemVerilog.

This document describes each migration step in detail.

Related Information

Migrating to the Quartus Prime Pro Edition Software

3.2.1. Keeping Pro Edition Project Files Separate

The Quartus Prime Pro Edition software does not support project or constraint files from other Quartus software products. Do not place project files from other Quartus software products in the same directory as Quartus Prime Pro Edition project files. In general, use Quartus Prime Pro Edition project files and directories only for Quartus Prime Pro Edition projects, and use other Quartus software product files only with those software tools.

Quartus Prime Pro Edition projects do not support compilation in other Quartus software products, and vice versa. The Quartus Prime Pro Edition software generates an error if the Compiler detects other Quartus software product's features in project files.

Before migrating other Quartus software product projects, click **Project ➤ Archive Project** to save a copy of your original project before making modifications for migration.

3.2.2. Upgrading Project Assignments and Constraints

Quartus Prime Pro Edition software introduces changes to handling of project assignments and constraints that the Quartus Settings File (`.qsf`) stores. Upgrade other Quartus software product project assignments and constraints for migration to the Quartus Prime Pro Edition software. Upgrade other Quartus software product assignments with **Assignments ➤ Assignment Editor**, by editing the `.qsf` file directly, or by using a Tcl script.

Note: If you open a Quartus Prime Standard Edition software project in the Quartus Prime Pro Edition software, and that project contains unsupported junction temperature setting values, you can modify these settings by clicking **Assignments ➤ Settings ➤ Operating Settings and Conditions ➤ Temperature**. Click **OK**, and then click **Yes** when prompted to update the values.

The following sections detail the various types of project assignment upgrade that migration requires.

Related Information

- Modifying Entity Name Assignments on page 34
- Resolving Timing Constraint Entity Names on page 34
- Verifying Generated Node Name Assignments on page 35
- Replace Logic Lock (Standard) Regions on page 35
- Modifying Signal Tap Logic Analyzer Files on page 37
- Removing Unsupported Feature Assignments on page 38

3.2.2.1. Modifying Entity Name Assignments

Quartus Prime Pro Edition software supports assignments that include instance names *without* a corresponding entity name.

- "`a_entity:a|b_entity:b|c_entity:c`" (includes deprecated entity names)
- "`a|b|c`" (omits deprecated entity names)

While the current version of the Quartus Prime Pro Edition software still *accepts* entity names in the `.qsf`, the Compiler *ignores* the entity name. The Compiler generates a warning message upon detection of an entity names in the `.qsf`. Whenever possible, you should remove entity names from assignments, and discontinue reliance on entity-based assignments. Future versions of the Quartus Prime Pro Edition software may eliminate all support for entity-based assignments.

3.2.2.2. Resolving Timing Constraint Entity Names

The Quartus Prime Pro Edition Timing Analyzer honors entity names in Synopsys Design Constraints (`.sdc`) files.

Use `.sdc` files from other Quartus software products without modification. However, any scripts that include custom processing of names that the `.sdc` command returns, such as `get_registers` may require modification. Your scripts must reflect that returned strings do not include entity names.

The `.sdc` commands respect wildcard patterns containing entity names. Review the Timing Analyzer reports to verify application of all constraints. The following example illustrates differences between functioning and non-functioning `.sdc` scripts:

```
# Apply a constraint to all registers named "acc" in the entity "counter".
# This constraint functions in both SE and PE, because the SDC
# command always understands wildcard patterns with entity names in them
set_false_path –to [get_registers  “counter:*|*acc”]

# This does the same thing, but first it converts all register names to
# strings, which includes entity names by default in the SE
# but excludes them by default in the PE. The regexp will therefore
# fail in PE by default.
```

```
#
# This script would also fail in the SE, and earlier
# versions of Quartus II, if entity name display had been disabled
# in the QSF.
set all_reg_strs [query_collection –list –all [get_registers *]]
foreach keeper $all_reg_strs {
   if {[regexp {counter:*|:*acc} $keeper]} {
      set_false_path –to $keeper
   }
}
```

Removal of the entity name processing from `.sdc` files may not be possible due to complex processing involving node names. Use standard `.sdc` whenever possible to replace such processing. Alternatively, add the following code to the top and bottom of your script to temporarily re-enable entity name display in the `.sdc` file:

```
# This script requires that entity names be included
# due to custom name processing
set old_mode [set_project_mode -get_mode_value always_show_entity_name]
set_project_mode -always_show_entity_name on

<... the rest of your script goes here ...>

# Restore the project mode
set_project_mode -always_show_entity_name $old_mode
```

3.2.2.3. Verifying Generated Node Name Assignments

Quartus Prime synthesis generates and automatically names internal design nodes during processing. The Quartus Prime Pro Edition uses different conventions than other Quartus software products to generate node names during synthesis. When you synthesize your other Quartus software product project in Quartus Prime Pro Edition, the synthesis-generated node names may change. If any scripts or constraints depend on the synthesis-generated node names, update the scripts or constraints to match the Quartus Prime Pro Edition synthesis node names.

Avoid dependence on synthesis-generated names due to frequent changes in name generation. In addition, verify the names of duplicated registers and PLL clock outputs to ensure compatibility with any script or constraint.

3.2.2.4. Replace Logic Lock (Standard) Regions

Quartus Prime Pro Edition software introduces more simplified and flexible Logic Lock constraints, compared with previous Logic Lock regions. You must replace all Logic Lock (Standard) assignments with compatible Logic Lock assignments for migration.

To convert Logic Lock (Standard) regions to Logic Lock regions:

1. Edit the `.qsf` to delete or comment out all of the following Logic Lock assignments:

```
set_global_assignment -name LL_ENABLED*
set_global_assignment -name LL_AUTO_SIZE*
set_global_assignment -name LL_STATE FLOATING*
set_global_assignment -name LL_RESERVED*
set_global_assignment -name LL_CORE_ONLY*
set_global_assignment -name LL_SECURITY_ROUTING_INTERFACE*
set_global_assignment -name LL_IGNORE_IO_BANK_SECURITY_CONSTRAINT*
set_global_assignment -name LL_PR_REGION*
set_global_assignment -name LL_ROUTING_REGION_EXPANSION_SIZE*
set_global_assignment -name LL_WIDTH*
```

```
set_global_assignment -name LL_HEIGHT
set_global_assignment -name LL_ORIGIN
set_instance_assignment -name LL_MEMBER_OF
```

2. Edit the `.qsf` or click **Tools ➤ Chip Planner** to define new Logic Lock regions. Logic Lock constraint syntax is simplified, for example:

```
set_instance_assignment -name PLACE_REGION "1 1 20 20" -to fifo1
set_instance_assignment -name RESERVE_PLACE_REGION OFF -to fifo1
set_instance_assignment -name CORE_ONLY_PLACE_REGION OFF -to fifo1
```

Compilation fails if synthesis finds other Quartus software product's Logic Lock assignments in an Quartus Prime Pro Edition project. The following table compares other Quartus software product region constraint support with the Quartus Prime Pro Edition software.

Table 12. Region Constraints Per Edition

Constraint Type	Logic Lock (Standard) Region Support Other Quartus Software Products	Logic Lock Region Support Quartus Prime Pro Edition
Fixed rectangular, nonrectangular or non-contiguous regions	Full support.	Full support.
Chip Planner entry	Full support.	Full support.
Periphery element assignments	Supported in some instances.	Full support. Use “core-only” regions to exclude the periphery.
Nested (“hierarchical”) regions	Supported but separate hierarchy from the user instance tree.	Supported in same hierarchy as user instance tree.
Reserved regions	Limited support for nested or nonrectangular reserved regions. Reserved regions typically cannot cross I/O columns; use non-contiguous regions instead.	Full support for nested and nonrectangular regions. Reserved regions can cross I/O columns without affecting periphery logic if the regions are "core-only".
Routing regions	Limited support via “routing expansion.” No support with hierarchical regions.	Full support (including future support for hierarchical regions).
Floating or autosized regions	Full support.	No support.
Region names	Regions have names.	Regions are identified by the instance name of the constrained logic.
Multiple instances in the same region	Full support.	Support for non-reserved regions. Create one region per instance, and then specify the same definition for multiple instances to assign to the same area. Not supported for reserved regions.
Member exclusion	Full support.	No support for arbitrary logic. Use a core-only region to exclude periphery elements. Use non-rectangular regions to include more RAM or DSP columns as needed.

3.2.2.4.1. Logic Lock Region Assignment Examples

The following examples show the syntax of Logic Lock region assignments in the `.qsf` file. Optionally, you can enter these assignments in the Assignment Editor, the Logic Lock Regions Window, or the Chip Planner.

Example 1. Assign Rectangular Logic Lock Region

Assigns a rectangular Logic Lock region to a lower left corner location of (10,10), and an upper right corner of (20,20) inclusive.

```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
```

Example 2. Assign Non-Rectangular Logic Lock Region

Assigns instance with full hierarchical path "x|y|z" to non-rectangular L-shaped Logic Lock region. The software treats each set of four numbers as a new box.

```
set_instance_assignment –name PLACE_REGION –to x|y|z "X10 Y10 X20 Y50; X20 Y10
X50 Y20"
```

Example 3. Assign Subordinate Logic Lock Instances

By default, the Quartus Prime software constrains every child instance to the Logic Lock region of its parent. Any constraint to a child instance intersects with the constraint of its ancestors. For example, in the following example, all logic beneath "a|b|c|d" constrains to box (10,10), (15,15), and not (0,0), (15,15). This result occurs because the child constraint intersects with the parent constraint.

```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
set_instance_assignment –name PLACE_REGION –to a|b|c|d "X0 Y0 X15 Y15"
```

Example 4. Assign Multiple Logic Lock Instances

By default, a Logic Lock region constraint allows logic from other instances to share the same region. These assignments place instance c and instance g in the same location. This strategy is useful if instance c and instance g are heavily interacting.

```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
set_instance_assignment –name PLACE_REGION –to e|f|g "X10 Y10 X20 Y20"
```

Example 5. Assigned Reserved Logic Lock Regions

Optionally reserve an entire Logic Lock region for one instance and any of its subordinate instances.

```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
set_instance_assignment –name RESERVE_PLACE_REGION –to a|b|c ON

# The following assignment causes an error. The logic in e|f|g is not
# legally placeable anywhere:
# set_instance_assignment –name PLACE_REGION –to e|f|g "X10 Y10 X20 Y20"

# The following assignment does *not* cause an error, but is effectively
# constrained to the box (20,10), (30,20), since the (10,10),(20,20) box is
reserved
# for a|b|c
set_instance_assignment –name PLACE_REGION –to e|f|g "X10 Y10 X30 Y20"
```

3.2.2.5. Modifying Signal Tap Logic Analyzer Files

Quartus Prime Pro Edition introduces new methodology for entity names, settings, and assignments. These changes impact the processing of Signal Tap Logic Analyzer Files (.stp).

If you migrate a project that includes `.stp` files generated by other Quartus software products, you must make the following changes to migrate to the Quartus Prime Pro Edition:

1. Remove entity names from `.stp` files. The Signal Tap Logic Analyzer allows without error, but ignores, entity names in `.stp` files. Remove entity names from `.stp` files for migration to Quartus Prime Pro Edition:
   a. Click **View ➤ Node Finder** to locate and remove appropriate nodes. Use Node Finder options to filter on nodes.
   b. Click **Processing ➤ Start ➤ Start Analysis & Elaboration** to repopulate the database and add valid node names.
2. Remove post-fit nodes. Quartus Prime Pro Edition uses a different post-fit node naming scheme than other Quartus software products.
   a. Remove post-fit tap node names originating from other Quartus software products.
   b. Click **View ➤ Node Finder** to locate and remove post-fit nodes. Use Node Finder options to filter on nodes.
   c. Click **Processing ➤ Start Compilation** to repopulate the database and add valid post-fit nodes.
3. Run an initial compilation in Quartus Prime Pro Edition from the GUI. The Compiler automatically removes Signal Tap assignments originating other Quartus software products. Alternatively, from the command-line, run `quartus_stp` once on the project to remove outmoded assignments.

   Note: `quartus_stp` introduces no migration impact in the Quartus Prime Pro Edition. Your scripts require no changes to `quartus_stp` for migration.
4. Modify `.sdc` constraints for JTAG. Quartus Prime Pro Edition does not support embedded `.sdc` constraints for JTAG signals. Modify the timing template to suit the design's JTAG driver and board.

3.2.2.6. Removing References to .qip Files

In Quartus Prime Standard Edition projects, Platform Designer (Standard) generates `.qip` files. These files describe the parameterized IP cores to the Compiler, and appear as assignments in the project's `.qsf` file. However, in Quartus Prime Pro Edition projects, the parameterized IP core description occurs in `.ip` files. Moreover, references to `.qip` files in a project's `.qsf` file cause synthesis errors during compilation.

- When migrating a project to Quartus Prime Pro Edition, remove all references to `.qip` files from the `.qsf` file.

3.2.2.7. Removing Unsupported Feature Assignments

The Quartus Prime Pro Edition software does not support some feature assignments that other Quartus software products support. Remove the following unsupported feature assignments from other Quartus software product `.qsf` files for migration to the Quartus Prime Pro Edition software.

- Incremental Compilation (partitions)—The current version of the Quartus Prime Pro Edition software does not support Quartus Prime Standard Edition incremental compilation. Remove all incremental compilation feature assignments from other Quartus software product `.qsf` files before migration.
- Quartus Prime Standard Edition Physical synthesis assignments. Quartus Prime Pro Edition software does not support Quartus Prime Standard Edition Physical synthesis assignments. Remove any of the following assignments from the `.qsf` file or design RTL (instance assignments) before migration.

```
PHYSICAL_SYNTHESIS_COMBO_LOGIC_FOR_AREA
    PHYSICAL_SYNTHESIS_COMBO_LOGIC
    PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION
    PHYSICAL_SYNTHESIS_REGISTER_RETIMING
    PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING
    PHYSICAL_SYNTHESIS_MAP_LOGIC_TO_MEMORY_FOR_AREA
```

Note: If you open a Quartus Prime Standard Edition software project in the Quartus Prime Pro Edition software, and that project contains unsupported junction temperature setting values, you can modify these settings by clicking **Assignments ➤ Settings ➤ Operating Settings and Conditions ➤ Temperature**. Click **OK**, and then click **Yes** when prompted to update the values.

3.2.3. Upgrading IP Cores and Platform Designer Systems

Upgrade all IP cores and Platform Designer systems in your project for migration to the Quartus Prime Pro Edition software. The Quartus Prime Pro Edition software uses standards-compliant methodology for instantiation and generation of IP cores and Platform Designer systems. Most Intel FPGA IP cores and Platform Designer systems upgrade automatically in the **Upgrade IP Components** dialog box.

Other Quartus software products use a proprietary Verilog configuration scheme within the top level of IP cores and Platform Designer systems for synthesis files. The Quartus Prime Pro Edition does not support this scheme. To upgrade all IP cores and Platform Designer systems in your project, click **Project ➤ Upgrade IP Components**.[(2)]

Table 13. IP Core and Platform Designer System Differences

Other Quartus Software Products	Quartus Prime Pro Edition
IP and Platform Designer system generation use a proprietary Verilog HDL configuration scheme within the top level of IP cores and Platform Designer systems for synthesis files. This proprietary Verilog HDL configuration scheme prevents RTL entities from ambiguous instantiation errors during synthesis. However, these errors may manifest in simulation. Resolving this issue requires writing a Verilog HDL configuration to disambiguate the instantiation, delete the duplicate entity from the project, or rename one of the conflicting entities. Quartus Prime Pro Edition IP strategy resolves these issues.	IP and Platform Designer system generation does not use proprietary Verilog HDL configurations. The compilation library scheme changes in the following ways: • Compiles all variants of an IP core into the same compilation library across the entire project. Quartus Prime Pro Edition identically names IP cores with identical functionality and parameterization to avoid ambiguous entity instantiation errors. For example, the files for every Arria 10 PCI Express* IP core variant compile into the `altera_pcie_a10_hip_151` compilation library. • Simulation and synthesis file sets for IP cores and systems instantiate entities in the same manner. • The generated RTL directory structure now matches the compilation library structure.

(2) For brevity, this section refers to Quartus Prime Standard Edition, Intel Quartus Prime Lite Edition, and the Quartus II software collectively as "other Quartus software products."

Note: For complete information on upgrading IP cores, refer to *Managing Quartus Prime Projects*.

Related Information

- Working With Intel FPGA IP Cores on page 50
- Managing Quartus Prime Projects on page 85

3.2.4. Upgrading Non-Compliant Design RTL

The Quartus Prime Pro Edition software introduces a new synthesis engine (`quartus_syn` executable).

The `quartus_syn` synthesis enforces stricter industry-standard HDL structures and supports the following enhancements in this release:

- Support for modules with SystemVerilog Interfaces
- Improved support for VHDL2008
- New RAM inference engine infers RAMs from GENERATE statements or array of integers
- Stricter syntax/semantics check for improved compatibility with other EDA tools

Account for these synthesis differences in existing RTL code by ensuring that your design uses standards-compliant VHDL, Verilog HDL, or SystemVerilog. The Compiler generates errors when processing non-compliant RTL. Use the guidelines in this section to modify existing RTL for compatibility with the Quartus Prime Pro Edition synthesis.

Related Information

- Verifying Verilog Compilation Unit on page 40
- Updating Entity Auto-Discovery on page 41
- Ensuring Distinct VHDL Namespace for Each Library on page 42
- Removing Unsupported Parameter Passing on page 42
- Removing Unsized Constant from WYSIWYG Instantiation on page 42
- Removing Non-Standard Pragmas on page 43
- Declaring Objects Before Initial Values on page 43
- Confining SystemVerilog Features to SystemVerilog Files on page 43
- Avoiding Assignment Mixing in Always Blocks on page 44
- Avoiding Unconnected, Non-Existent Ports on page 44
- Avoiding Invalid Parameter Ranges on page 44
- Updating Verilog HDL and VHDL Type Mapping on page 45

3.2.4.1. Verifying Verilog Compilation Unit

Quartus Prime Pro Edition synthesis uses a different method to define the compilation unit. The Verilog LRM defines the concept of compilation unit as “a collection of one or more Verilog source files compiled together” forming the compilation-unit scope. Items visible only in the compilation-unit scope include macros, global declarations,

and default net types. The contents of included files become part of the compilation unit of the parent file. Modules, primitives, programs, interfaces, and packages are visible in all compilation units. Ensure that your RTL accommodates these changes.

Table 14. Verilog Compilation Unit Differences

Other Quartus Software Products	Quartus Prime Pro Edition
Synthesis in other Quartus software products follows the Multi-file compilation unit (MFCU) method to select compilation unit files. In MFCU, all files compile in the same compilation unit. Global definitions and directives are visible in all files. However, the default net type is reset at the start of each file.	Quartus Prime Pro Edition synthesis follows the Single-file compilation unit (SFCU) method to select compilation unit files. In SFCU, each file is a compilation unit, file order is irrelevant, and the macro is only defined until the end of the file.

Note: You can optionally change the MFCU mode using the following assignment:
`set_global_assignment -name VERILOG_CU_MODE MFCU`

3.2.4.1.1. Verilog HDL Configuration Instantiation

Quartus Prime Pro Edition synthesis requires instantiation of the Verilog HDL configuration, and not the module. In other Quartus software products, synthesis automatically finds any Verilog HDL configuration relating to a module that you instantiate. The Verilog HDL configuration then instantiates the design.

If your top-level entity is a Verilog HDL configuration, set the Verilog HDL configuration, rather than the module, as the top-level entity.

Table 15. Verilog HDL Configuration Instantiation

Other Quartus Software Products	Quartus Prime Pro Edition
From the Example RTL, synthesis automatically finds the `mid_config` Verilog HDL configuration relating to the instantiated module.	From the Example RTL, synthesis does not find the `mid_config` Verilog HDL configuration. You must instantiate the Verilog HDL configuration directly.
Example RTL: `config mid_config;` `design good_lib.mid;` `instance mid.sub_inst use good_lib.sub;` `endconfig` `module test (input a1, output b);` `mid_config mid_inst ( .a1(a1), .b(b));` **`// in other Quartus products preceding line would have been:`** **`//mid mid_inst ( .a1(a1), .b(b));`** `endmodule` `module mid (input a1, output b);` `sub sub_inst (.a1(a1), .b(b));` `endmodule`	

3.2.4.2. Updating Entity Auto-Discovery

All editions of the Quartus Prime and Quartus II software search your project directory for undefined entities. For example, if you instantiate entity "sub" in your design without specifying "sub" as a design file in the Quartus Settings File (`.qsf`), synthesis searches for `sub.v`, `sub.vhd`, and so on. However, Quartus Prime Pro Edition performs auto-discovery at a different stage in the flow. Ensure that your RTL code accommodates these auto-discovery changes.

Table 16. Entity Auto-Discovery Differences

Other Quartus Software Products	Quartus Prime Pro Edition
Always automatically searches your project directory and search path for undefined entities.	Always automatically searches your project directory and search path for undefined entities. Quartus Prime Pro Edition synthesis performs auto-discovery earlier in the flow than other Quartus software products. This results in discovery of more syntax errors. Optionally disable auto-discovery with the following `.qsf` assignment: `set_global_assignment -name AUTO_DISCOVER_AND_SORT OFF`

3.2.4.3. Ensuring Distinct VHDL Namespace for Each Library

Quartus Prime Pro Edition synthesis requires that VHDL namespaces are distinct for each library. The stricter library binding requirement complies with VHDL language specifications and results in deterministic behavior. This benefits team-based projects by avoiding unintentional name collisions. Confirm that your RTL respects this change.

Table 17. VHDL Namespace Differences

Other Quartus Software Products	Quartus Prime Pro Edition
For the Example RTL, the analyzer searches all libraries in an unspecified order until the analyzer finds package `utilities_pack` and uses items from that package. If another library, for example `projectLib` also contains `utilities_pack`, the analyzer may use this library instead of `myLib.utilites_pack` if found before the analyzer searches `myLib`.	For the Example RTL, the analyzer uses the specific `utilities_pack` in `myLib`. If `utilities_pack` does not exist in library `myLib`, the analyzer generates an error.
Example RTL: `library myLib; use myLib.utilities_pack.all;`	

3.2.4.4. Removing Unsupported Parameter Passing

Quartus Prime Pro Edition synthesis does not support parameter passing using `set_parameter` in the `.qsf`. Synthesis in other Quartus software products supports passing parameters with this method. Except for the top-level of the design where permitted, ensure that your RTL does not depend on this type of parameter passing.

Table 18. SystemVerilog Feature Differences

Other Quartus Software Products	Quartus Prime Pro Edition
From the Example RTL, synthesis overwrites the value of parameter SIZE in the instance of `my_ram` instantiated from entity `mid-level`.	From the Example RTL, synthesis generates a syntax error for detection of parameter passing assignments in the `.qsf`. Specify parameters in the RTL. The following example shows the supported top-level parameter passing format. This example applies only to the top-level and sets a value of 4 to parameter `N`: `set_parameter -name N 4`
Example RTL: `set_parameter –entity mid_level –to my_ram –name SIZE 16`	

3.2.4.5. Removing Unsized Constant from WYSIWYG Instantiation

Quartus Prime Pro Edition synthesis does not allow use of an unsized constant for WYSIWYG instantiation. Synthesis in other Quartus software products allows use of SystemVerilog (`.sv`) unsized constants when instantiating a WYSIWYG in a `.v` file.

Quartus Prime Pro Edition synthesis allows use of unsized constants in `.sv` files for uses other than WYSIWYG instantiation. Ensure that your RTL code does not use unsized constants for WYSIWYG instantiation. For example, specify a sized literal, such as 2'b11, rather than '1.

3.2.4.6. Removing Non-Standard Pragmas

Quartus Prime Pro Edition synthesis does not support the `vhdl(verilog)_input_version` pragma or the `library` pragma. Synthesis in other Quartus software products supports these pragmas. Remove any use of the pragmas from RTL for Quartus Prime Pro Edition migration. Use the following guidelines to implement the pragma functionality in Quartus Prime Pro Edition:

- `vhdl(verilog)_input_version` Pragma—allows change to the input version in the middle of an input file. For example, to change VHDL 1993 to VHDL 2008. For Quartus Prime Pro Edition migration, specify the input version for each file in the `.qsf`.
- `library` Pragma—allows changes to the VHDL library into which files compile. For Quartus Prime Pro Edition migration, specify the compilation library in the `.qsf`.

3.2.4.7. Declaring Objects Before Initial Values

Quartus Prime Pro Edition synthesis requires declaration of objects before initial value. Ensure that your RTL declares objects before initial value. Other Quartus software products allow declaration of initial value prior to declaration of the object.

Table 19. Object Declaration Differences

Other Quartus Software Products	Quartus Prime Pro Edition
From the Example RTL, synthesis initializes the output `p_prog_io1` with the value of `p_progio1_reg`, even though the register declaration occurs in Line 2.	From the Example RTL, synthesis generates a syntax error when you specify initial values before declaring the register.
Example RTL: `1 output p_prog_io1 = p_prog_io1_reg;` `2 reg p_prog_io1_reg;`	

3.2.4.8. Confining SystemVerilog Features to SystemVerilog Files

Quartus Prime Pro Edition synthesis does not allow SystemVerilog features in Verilog HDL files. Other Quartus software products allow use of a subset of SystemVerilog (`.sv`) features in Verilog HDL (`.v`) design files. To avoid syntax errors in Quartus Prime Pro Edition, allow only SystemVerilog features in Verilog HDL files.

To use SystemVerilog features in your existing Verilog HDL files, rename your Verilog HDL (`.v`) files as SystemVerilog (`.sv`) files. Alternatively, you can set the file type in the .qsf, as shown in the following example:

```
set_global_assignment -name SYSTEMVERILOG_FILE <file>.v
```

Table 20. SystemVerilog Feature Differences

Other Quartus Software Products	Quartus Prime Pro Edition
From the Example RTL, synthesis interprets `$clog2` in a `.v` file, even though the Verilog LRM does not define the `$clog2` feature. Other Quartus software products allow other SystemVerilog features in `.v` files.	From the Example RTL, synthesis generates a syntax error for detection of any non-Verilog HDL construct in `.v` files. Quartus Prime Pro Edition synthesis honors SystemVerilog features only in `.sv` files.
Example RTL: `localparam num_mem_locations = 1050;` `wire mem_addr [$clog2(num_mem_locations)-1 : 0];`	

3.2.4.9. Avoiding Assignment Mixing in Always Blocks

Quartus Prime Pro Edition synthesis does not allow mixed use of blocking and non-blocking assignments within ALWAYS blocks. Other Quartus software products allow mixed use of blocking and non-blocking assignments within ALWAYS blocks. To avoid syntax errors, ensure that ALWAYS block assignments are of the same type for Quartus Prime Pro Edition migration.

Table 21. ALWAYS Block Assignment Differences

Other Quartus Software Products	Quartus Prime Pro Edition
Synthesis honors the mixed blocking and non-blocking assignments, although the Verilog Language Specification no longer supports this construct.	Synthesis generates a syntax error for detection of mixed blocking and non-blocking assignments within an ALWAYS block.

3.2.4.10. Avoiding Unconnected, Non-Existent Ports

Quartus Prime Pro Edition synthesis requires that a port exists in the module prior to instantiation and naming. Other Quartus software products allow you to instantiate and name an unconnected port that does not exist in the module. Modify your RTL to match this requirement.

To avoid syntax errors, remove all unconnected and non-existent ports for Quartus Prime Pro Edition migration.

Table 22. Unconnected, Non-Existent Port Differences

Other Quartus Software Products	Quartus Prime Pro Edition
Synthesis allows you to instantiate and name unconnected or non-existent ports that do not exist on the module.	Synthesis generates a syntax error for detection of mixed blocking and non-blocking assignments within an ALWAYS block.

3.2.4.11. Avoiding Invalid Parameter Ranges

Quartus Prime Pro Edition synthesis generates an error for detection of constant numeric (integer or floating point) parameter values that exceed the language specification. Other Quartus software products allow constant numeric (integer or floating point) values for parameters that exceed the language specifications. To avoid syntax errors, ensure that constant numeric (integer or floating point) values for parameters conform to the language specifications.

3.2.4.12. Updating Verilog HDL and VHDL Type Mapping

Quartus Prime Pro Edition synthesis requires that you use `0` for "`false`" and `1` for "`true`" in Verilog HDL files (`.v`). Other Quartus software products map "true" and "false" strings in Verilog HDL to TRUE and FALSE Boolean values in VHDL. Quartus Prime Pro Edition synthesis generates an error for detection of non-Verilog HDL constructs in `.v` files. To avoid syntax errors, ensure that your RTL accommodates these standards.

3.2.4.13. Converting Symbolic BDF Files to Acceptable File Formats

Starting from the Quartus Prime Pro Edition software version 23.3, the compiler cannot synthesize schematic Block Design File (`.bdf`). You must convert it to an acceptable format, such as Verilog HDL or VHDL using the Intel Quartus Prime Standard Edition command `quartus_map` as shown in the following:

- To convert your `.bdf` file to Verilog Design File (`.v`):

```
quartus_map <project_name> --convert_bdf_to_verilog=<bdf_file_name>
```

- To convert your `.bdf` file to VHDL Design File (`.vhd`):

```
quartus_map <project_name> --convert_bdf_to_vhdl=<bdf_file_name>
```

3.3. Migrating Your AMD* Vivado* Project to Quartus Prime Pro Edition

Designing for Intel FPGA devices is similar in concept and practice to designing for AMD* Xilinx* FPGA devices. In most cases, you can import your RTL into the Quartus Prime Pro Edition software and compile your design to the target device.

Refer to the AN 307: Intel FPGA Design Flow for AMD* Xilinx* Users, which covers the following information:

- A comparison of the current AMD* Xilinx* and Intel FPGA technologies, features, and devices available for different process technologies.
- A comparison between the design flows in the AMD* Vivado* software and Quartus Prime Pro Edition software.
- Guidelines to convert AMD* Vivado* designs to the Quartus Prime Pro Edition software, including AMD* Xilinx* IP catalog modules and instantiated primitives.
- Guidelines to translate device and design constraints.

3.4. Migrating Projects Across Operating Systems

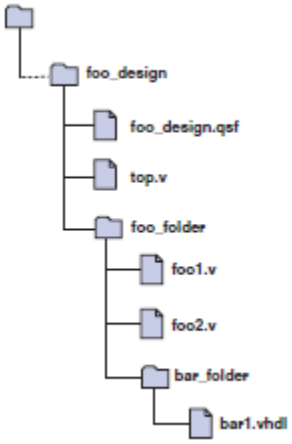
Consider the following cross-platform issues when moving your project from one operating system to another (for example, from Windows* to Linux*).

3.4.1. Migrating Design Files and Libraries

Consider file naming differences when migrating projects across operating systems.

- Use appropriate case for your platform in file path references.
- Use a character set common to both platforms.
- Do not change the forward-slash (`/`) and back-slash (`\`) path separators in the `.qsf`. The Quartus Prime software automatically changes all back-slash (`\`) path separators to forward-slashes (`/ `)in the `.qsf`.
- Observe the target platform's file name length limit.
- Use underscore instead of spaces in file and directory names.
- Change library absolute path references to relative paths in the `.qsf`.
- Ensure that any external project library exists in the new platform's file system.
- Specify file and directory paths as relative to the project directory. For example, for a project titled `foo_design`, specify the source files as: `top.v`, `foo_folder /foo1.v`, `foo_folder /foo2.v`, and `foo_folder/bar_folder/bar1.vhdl`.
- Ensure that all the subdirectories are in the same hierarchical structure and relative path as in the original platform.

Figure 14. All Inclusive Project Directory Structure

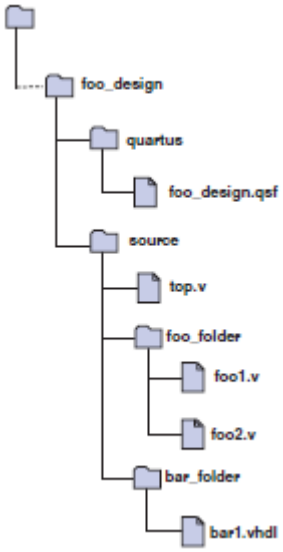


3.4.1.1. Use Relative Paths

Express file paths using relative path notation (`../`).

For example, in the directory structure shown you can specify `top.v` as `../source/top.v` and `foo1.v` as `../source/foo_folder/foo1.v`.

Figure 15. Quartus Prime Project Directory Separate from Design Files



3.4.2. Design Library Migration Guidelines

The following guidelines apply to library migration across computing platforms:

1. The project directory takes precedence over the project libraries.
2. For Linux, the Quartus Prime software creates the file in the `altera.quartus` directory under the <*home*> directory.
3. All library files are relative to the libraries. For example, if you specify the `user_lib1` directory as a project library and you want to add the `/user_lib1/foo1.v` file to the library, you can specify the `foo1.v` file in the `.qsf` as `foo1.v`. The Quartus Prime software includes files in specified libraries.
4. If the directory is outside of the project directory, an absolute path is created by default. Change the absolute path to a relative path before migration.
5. When copying projects that include libraries, you must either copy your project library files along with the project directory or ensure that your project library files exist in the target platform.
   - On Windows*, the Quartus Prime software searches for the `quartus2.ini` file in the following directories and order:
   - `USERPROFILE`, for example, `C:\Documents and Settings\<user name>`
   - Directory specified by the TMP environmental variable
   - Directory specified by the TEMP environmental variable
   - Root directory, for example, `C:\`

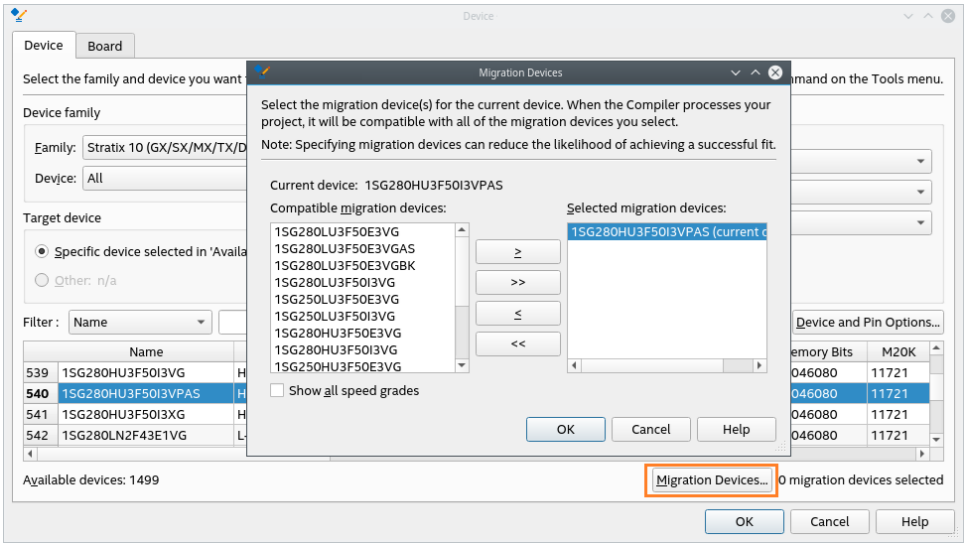
3.5. Migrating Project From One Device to Another

The Quartus Prime Pro Edition software supports migrating project from one device to another by providing a list of compatible migration devices available for the device your design targets.

To migrate your device, launch **Migration Devices** using one of the following options in the Quartus Prime Pro Edition software GUI:

- Right-click on your device in the **Project Navigator** and select **Device ➤ Migration Devices**.
- **Assignments ➤ Device ➤ Migration Devices**

Figure 16. Device Migration



In the **Migration Devices** dialog box, click **>**, **>>**, **<**, and **<<** to move migration devices between the **Compatible migration devices** list and the **Selected migration devices** list.

A device name in the **Selected migration devices** list with the text `(current device)` indicates that the device is currently specified in the **Available devices** list in the **Device** dialog box.

A device name in the **Compatible migration devices** list with the text `(not installed)` indicates that the device is supported in the Quartus Prime Pro Edition software, but support for the device is not installed in your copy of the software. To move this device to the **Selected migration devices** list, you must first install support for the device by running a custom installation procedure of the Quartus Prime software. For more information, refer to the Downloading Device Support in the *Intel FPGA Software Installation and Licensing* or contact Intel Support.

If you want the Quartus Prime software to display all compatible migration devices in the **Compatible migration devices** list regardless of a migration device's speed grade, turn on the **Show All Speed Grades** checkbox. If you want the Quartus Prime software to display in the **Compatible migration devices** list only the compatible migration devices that have the same speed grade as the target device, turn off **Show all speed grades**.

Related Information

- Migrating to the Quartus Prime Pro Edition Software
- AN 822: Intel FPGA Configuration Device Migration Guideline

3.6. Related Trainings

You can take up the following training to help you select your starting point for your project:

- Getting Started with the Quartus Prime New Project Wizard
- Creating a New Project with Quartus Prime Pro Edition Software
- Migrating to the Quartus Prime Pro Edition Software
- Migrating an Quartus Prime Project to a Different Intel FPGA Device
- Quartus Prime Software Pin Migration
- Preserve Compilation Results for migration to newer Quartus Prime Software releases

4. Working With Intel FPGA IP Cores

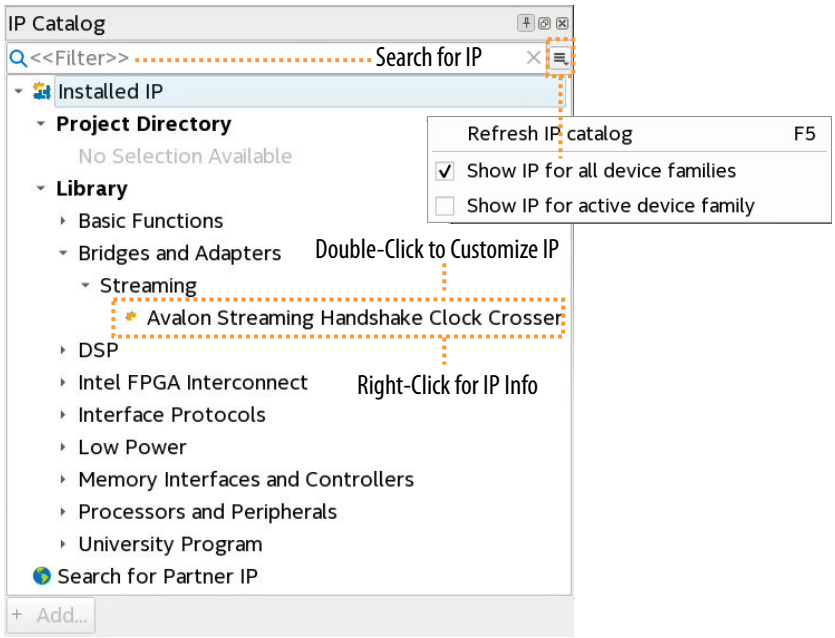
Intel and strategic IP partners offer a broad portfolio of configurable IP cores optimized for Intel FPGA devices.

The Quartus Prime software installation includes the Intel FPGA IP library. Integrate optimized and verified Intel FPGA IP cores into your design to shorten design cycles and maximize performance. The Quartus Prime software also supports integration of IP cores from other sources. Use the IP Catalog (**Tools ➤ IP Catalog**) to efficiently parameterize and generate synthesis and simulation files for your custom IP variation. The Intel FPGA IP library includes the following types of IP cores:

Basic functions	Interface protocols
Bridges and adapters	Low power functions
DSP functions	Memory interfaces and controllers
Intel FPGA interconnect	Processors and peripherals

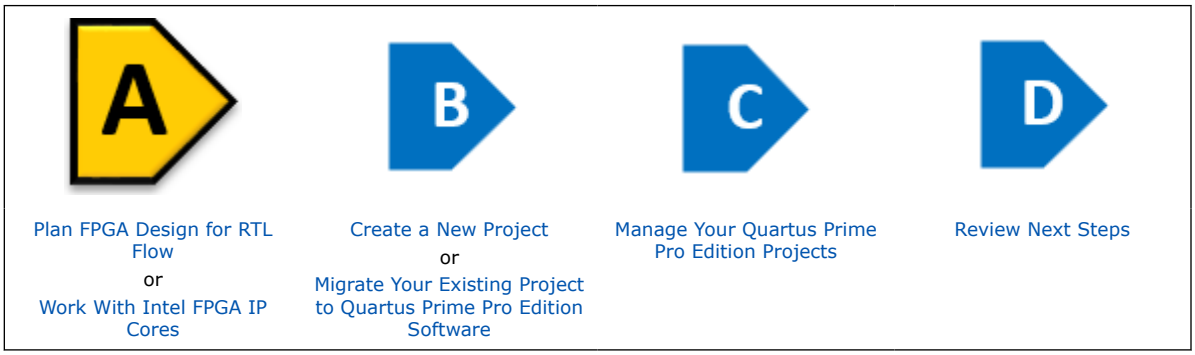
This document provides basic information about parameterizing, generating, upgrading, and simulating stand-alone IP cores in the Quartus Prime software.

Figure 17. Intel FPGA IP Catalog



Navigating Content Through Tasks

Use the following navigation diagram to navigate this guide through user-tasks:



4.1. IP Catalog and Parameter Editor

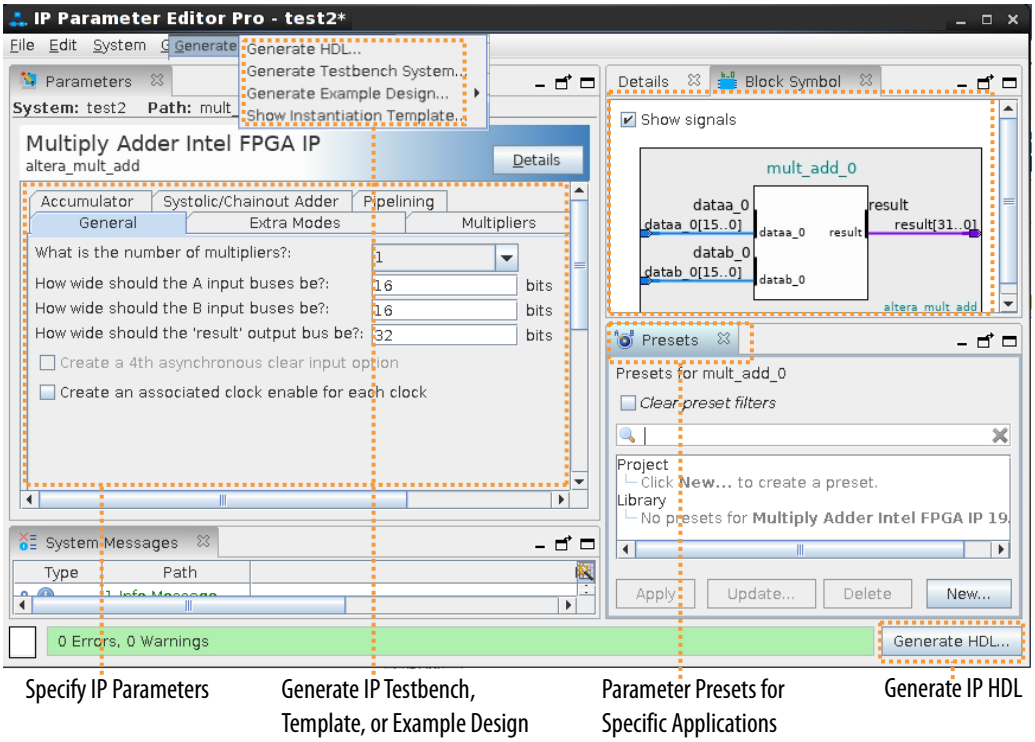
The IP Catalog displays the IP cores available for your project, including Intel FPGA IP and other IP that you add to the IP Catalog search path. Use the following features of the IP Catalog to locate and customize an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**. If you have no project open, select the **Device Family** in IP Catalog.
- Type in the Search field to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, to open the IP core's installation folder, and for links to IP documentation.
- Click **Search for Partner IP** to access partner IP information on the web.

The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Quartus Prime IP file (`.ip`) for an IP variation in Quartus Prime Pro Edition projects. This file represents the IP variation in the project, and stores parameterization information.(3)

(3) The parameter editor generates a top-level Quartus IP file (`.qip`) for an IP variation in Quartus Prime Standard Edition projects.

Figure 18. Example IP Parameter Editor



4.1.1. The Parameter Editor

The parameter editor helps you to configure IP core ports, parameters, and output file generation options. The basic parameter editor controls include the following:

- Use the **Presets** window to apply preset parameter values for specific applications (for select cores).
- Use the **Details** window to view port and parameter descriptions, and click links to documentation.
- Click **Generate ➤ Generate Testbench System** to generate a testbench system (for select cores).
- Click **Generate ➤ Generate Example Design** to generate an example design (for select cores).
- Click **Validate System Integrity** to validate a system's generic components against companion files. (Platform Designer systems only)
- Click **Sync All System Info** to validate a system's generic components against companion files. (Platform Designer systems only)

The IP Catalog is also available in Platform Designer (**View ➤ IP Catalog**). The Platform Designer IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IP that are not available in the Quartus Prime IP Catalog. Refer to *Creating a System with Platform Designer* or *Creating a System with Platform Designer* for information on use of IP in Platform Designer and Platform Designer, respectively.

Related Information

Creating a System with Platform Designer

4.2. Installing and Licensing Intel FPGA IP Cores

The Quartus Prime software installation includes the Intel FPGA IP library. This library provides many useful IP cores for your production use without the need for an additional license. Some Intel FPGA IP cores require purchase of a separate license for production use. The Intel FPGA IP Evaluation Mode allows you to evaluate these licensed Intel FPGA IP cores in simulation and hardware, before deciding to purchase a full production IP core license. You only need to purchase a full production license for licensed Intel IP cores after you complete hardware testing and are ready to use the IP in production.

The Quartus Prime software installs IP cores in the following locations by default:

Figure 19. IP Core Installation Path

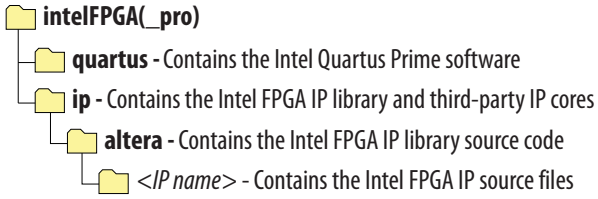


Table 23. IP Core Installation Locations

Location	Software	Platform
`<drive>:\intelFPGA_pro\quartus\ip\altera`	Quartus Prime Pro Edition	Windows
`<drive>:\intelFPGA\quartus\ip\altera`	Quartus Prime Standard Edition	Windows
`<home directory>:/intelFPGA_pro/quartus/ip/altera`	Quartus Prime Pro Edition	Linux*
`<home directory>:/intelFPGA/quartus/ip/altera`	Quartus Prime Standard Edition	Linux

Note: The Quartus Prime software does not support spaces in the installation path.

4.2.1. Intel FPGA IP Evaluation Mode

The free Intel FPGA IP Evaluation Mode allows you to evaluate licensed Intel FPGA IP cores in simulation and hardware before purchase. Intel FPGA IP Evaluation Mode supports the following evaluations without additional license:

- Simulate the behavior of a licensed Intel FPGA IP core in your system.
- Verify the functionality, size, and speed of the IP core quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device with your IP core and verify your design in hardware.

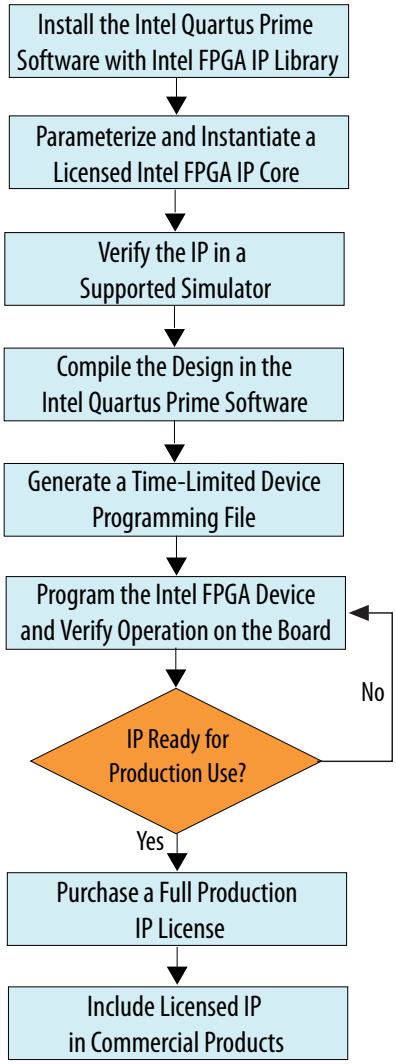
Intel FPGA IP Evaluation Mode supports the following operation modes:

- **Tethered**—Allows running the design containing the licensed Intel FPGA IP indefinitely with a connection between your board and the host computer. Tethered mode requires a serial joint test action group (JTAG) cable connected between the JTAG port on your board and the host computer, which is running the Quartus Prime Programmer for the duration of the hardware evaluation period. The Programmer only requires a minimum installation of the Quartus Prime software, and requires no Quartus Prime license. The host computer controls the evaluation time by sending a periodic signal to the device via the JTAG port. If all licensed IP cores in the design support tethered mode, the evaluation time runs until any IP core evaluation expires. If all of the IP cores support unlimited evaluation time, the device does not time-out.
- **Untethered**—Allows running the design containing the licensed IP for a limited time. The IP core reverts to untethered mode if the device disconnects from the host computer running the Quartus Prime software. The IP core also reverts to untethered mode if any other licensed IP core in the design does not support tethered mode.

When the evaluation time expires for any licensed Intel FPGA IP in the design, the design stops functioning. All IP cores that use the Intel FPGA IP Evaluation Mode time out simultaneously when any IP core in the design times out. When the evaluation time expires, you must reprogram the FPGA device before continuing hardware verification. To extend use of the IP core for production, purchase a full production license for the IP core.

You must purchase the license and generate a full production license key before you can generate an unrestricted device programming file. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (<*project name*>`_time_limited.sof`) that expires at the time limit.

Figure 20. Intel FPGA IP Evaluation Mode Flow



Note: Refer to each IP core's user guide for parameterization steps and implementation details.

Intel licenses IP cores on a per-seat, perpetual basis. The license fee includes first-year maintenance and support. You must renew the maintenance contract to receive updates, bug fixes, and technical support beyond the first year. You must purchase a full production license for Intel FPGA IP cores that require a production license, before generating programming files that you may use for an unlimited time. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (`<project name>_time_limited.sof`) that expires at the time limit. To obtain your production license keys, visit the Intel FPGA Self-Service Licensing Center.

The Intel FPGA Software License Agreements govern the installation and use of licensed IP cores, the Quartus Prime design software, and all unlicensed IP cores.

Related Information

- Intel FPGA Licensing Support Center
- Introduction to Intel FPGA Software Installation and Licensing

4.2.1.1. Intel FPGA IP Versioning

Intel FPGA IP versions match the Quartus Prime Design Suite software versions until v19.1. Starting in Quartus Prime Design Suite software version 19.2, Intel FPGA IP has a new versioning scheme.

The Intel FPGA IP version (X.Y.Z) number can change with each Quartus Prime software version. A change in:

- X indicates a major revision of the IP. If you update the Quartus Prime software, you must regenerate the IP.
- Y indicates the IP includes new features. Regenerate your IP to include these new features.
- Z indicates the IP includes minor changes. Regenerate your IP to include these changes.

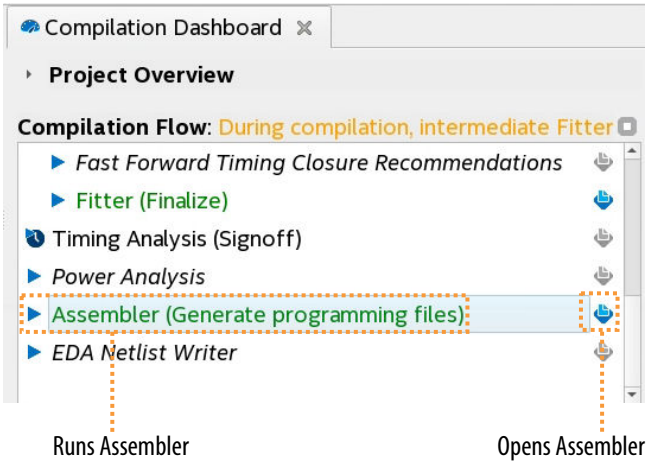
4.2.1.2. Checking the IP License Status

You can check the license status of all IP in an Quartus Prime project by viewing the Assembler report.

To generate and view the Assembler report in the GUI:

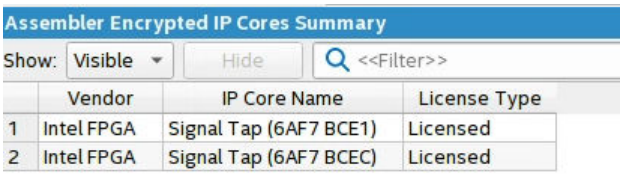
1. Click **Assembler** on the Compilation Dashboard.
2. When the Assembler (and any prerequisite stages of compilation) complete, click the **Report** icon for the Assembler in the Compilation Dashboard.

Figure 21. Assembler Report Icon in Compilation Dashboard



3. Click the **Encrypted IP Cores Summary** report.

Figure 22. Encrypted IP Cores Summary Report



To generate and view the Assembler report at the command line:

1. Type the following command:

```
quartus_asm <project name> -c  <project revision>
```

2. View the output report in `/output_files/<project_name>.asm.rpt`.

```
+-----------------------------------------------------------------------+
; Assembler Encrypted IP Cores Summary                                  ;
+--------+--------------------------------------------+--------------+
; Vendor ; IP Core Name                               ; License Type ;
+--------+--------------------------------------------+--------------+
; Intel  ; PCIe SRIOV with 4-PFs and 2K-VFs (6AF7 00FB) ; Unlicensed   ;
; Intel  ; Signal Tap (6AF7 BCE1)                     ; Licensed     ;
; Intel  ; Signal Tap (6AF7 BCEC)                     ; Licensed     ;
+--------+--------------------------------------------+--------------+
```

4.3. IP General Settings

The following settings control how the Quartus Prime software manages IP cores in a project:

Table 24. Location of IP Core General Settings in the Quartus Prime Software

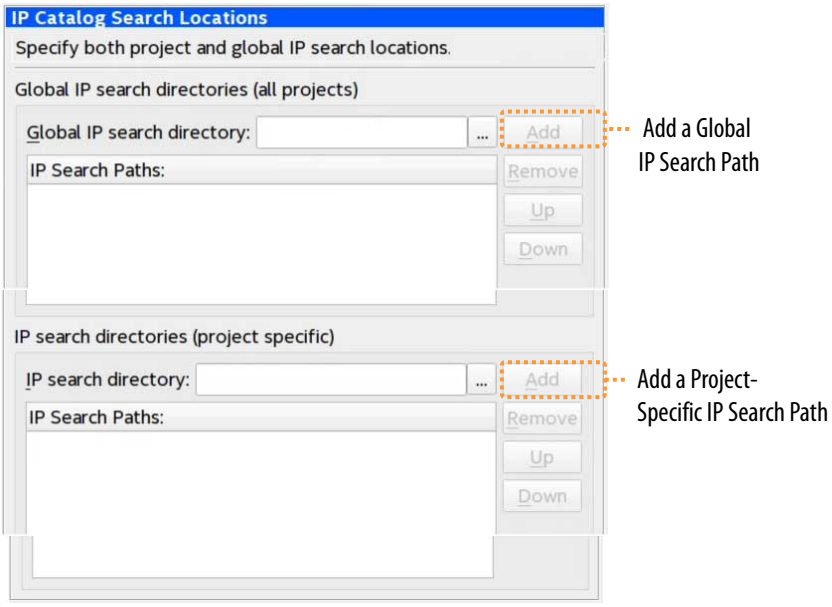
Setting	Description	Location
Maximum Platform Designer memory usage size	Increase if you experience slow processing for large systems, or for out of memory errors.	**Tools ➤ Options ➤ IP Settings** Or **Tasks pane ➤ Settings ➤ IP Settings**
IP generation HDL preference	The parameter editor generates the HDL you specify for IP variations.	
IP Regeneration Policy	Controls when synthesis files regenerate for each IP variation. Typically, you **Always regenerate synthesis files for IP cores** after making changes to an IP variation.	
Generate IP simulation model when generating IP	Enables automatic generation of simulation models every time you generate the IP.	
Use available processors for parallel generation of Quartus project IPs	Directs Platform Designer to generate IPs in parallel, using the number of processors that you specify in the **Compilation Process Settings** pane of the Quartus Prime project settings.	
Additional project and global IP search locations.	The Quartus Prime software searches for IP cores in the project directory, in the Quartus Prime installation directory, and in the IP search path.	**Tools ➤ Options ➤ IP Catalog Search Locations** Or **Tasks pane ➤ Settings ➤ IP Catalog Search Locations**

4.4. Adding IP to IP Catalog

The IP Catalog automatically displays Intel FPGA IP and other IP components that have a corresponding `_hw.tcl` or `.ipx` file located in the project directory, in the default Quartus Prime installation directory, or in the IP search path. You can optionally add your own custom or third-party IP component to IP Catalog by adding the component's `_hw.tcl` or `.ipx` file to the IP search path.

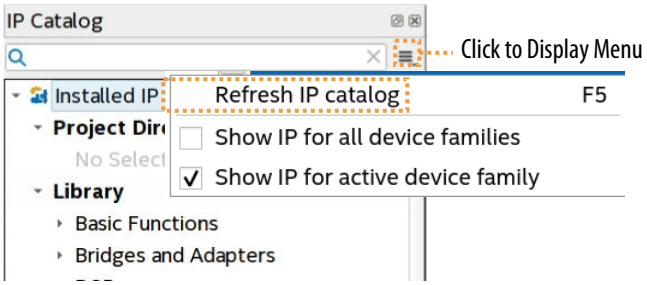
Follow these steps to add custom or third-party IP to the IP Catalog:

Figure 23. Specifying IP Search Locations



1. In the Quartus Prime software, click **Tools ➤ Options ➤ IP Search Path**) to open the **IP Search Path Options** dialog box.
2. Click **Add** or **Remove** to add/remove a location that contains IP.
3. To refresh the IP Catalog, click **Refresh IP Catalog** in the Quartus Prime Platform Designer, or click **File ➤ Refresh System** in Platform Designer.

Figure 24. Refreshing IP Catalog



4.5. Best Practices for Intel FPGA IP

Use the following best practices when working with Intel FPGA IP:

- Do not manually edit or write your own `.qsys`, `.ip`, or `.qip` file. Use the Quartus Prime software tools to create and edit these files.

  Note: When generating IP cores, do not generate files into a directory that has a space in the directory name or path. Spaces are not legal characters for IP core paths or names.
- When you generate an IP core using the IP Catalog, the Quartus Prime software generates a `.qsys` (for Platform Designer-generated IP cores) or a `.ip` file (for Quartus Prime Pro Edition) or a `.qip` file. The Quartus Prime Pro Edition software automatically adds the generated `.ip` to your project. In the Quartus Prime Standard Edition software, add the `.qip` to your project. Do not add the parameter editor generated file (`.v` or `.vhd`) to your design without the `.qsys` or `.qip` file. Otherwise, you cannot use the IP upgrade or IP parameter editor feature.
- Plan your directory structure ahead of time. Do not change the relative path between a `.qsys` file and it's generation output directory. If you must move the `.qsys` file, ensure that the generation output directory remains with the `.qsys` file.
- Do not add IP core files directly from the `/quartus/libraries/megafunctions` directory in your project. Otherwise, you must update the files for each subsequent software release. Instead, use the IP Catalog and then add the `.qip` to your project.
- Do not use IP files that the Quartus Prime software generates for RAM or FIFO blocks targeting older device families (even though the Quartus Prime software does not issue an error). The RAM blocks that Quartus Prime generates for older device families are not optimized for the latest device families.
- When generating a ROM function, save the resulting `.mif` or `.hex` file in the same folder as the corresponding IP core's `.qsys` or `.qip` file. For example, moving all of your project's `.mif` or `.hex` files to the same directory causes relative path problems after archiving the design.
- Always use the Quartus Prime `ip-setup-simulation` and `ip-make-simscript` utilities to generate simulation scripts for each IP core or Platform Designer system in your design. These utilities produce a single simulation script that does not require manual update for upgrades to Quartus Prime software or IP versions, as Simulating Intel FPGA IP Cores on page 75 describes.

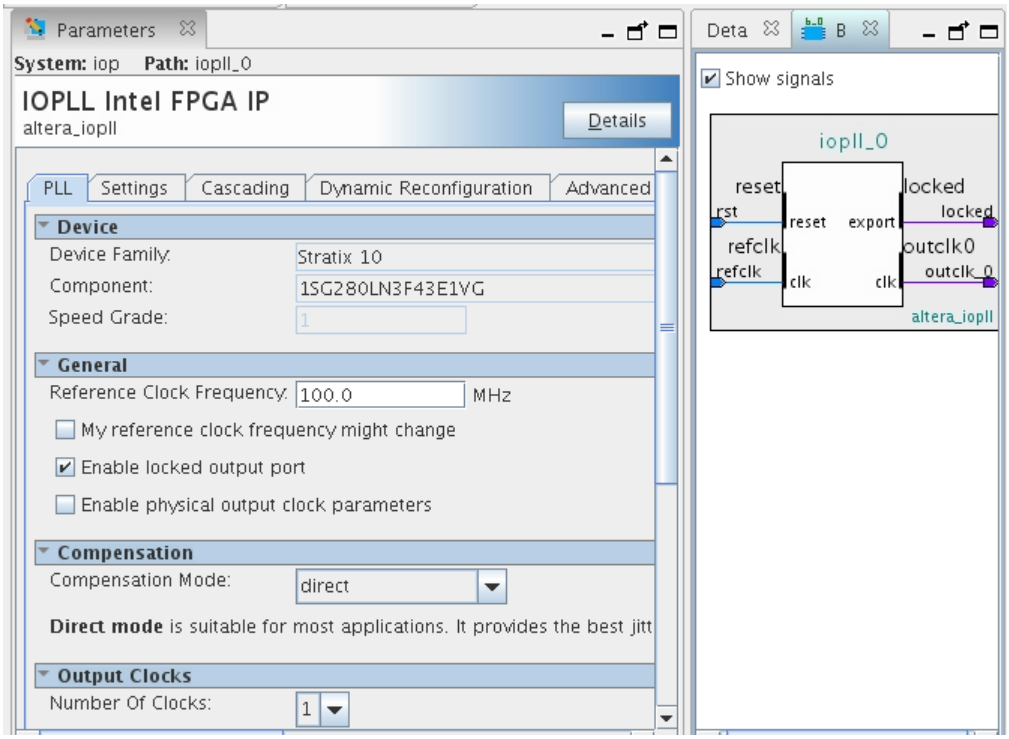
4.6. Specifying the IP Core Parameters and Options (Quartus Prime Pro Edition)

Quickly configure Intel FPGA IP cores in the Quartus Prime parameter editor. Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to define a custom variation of the IP core. The parameter editor generates the IP variation synthesis and optional simulation files, and adds the `.ip` file representing the variation to your project automatically.

Follow these steps to locate, instantiate, and customize an IP core in the parameter editor:

1. Create or open an Quartus Prime project (`.qpf`) to contain the instantiated IP variation.
2. In the IP Catalog (**Tools ➤ IP Catalog**), locate and double-click the name of the IP core to customize. To locate a specific component, type some or all of the component's name in the IP Catalog search box. The New IP Variation window appears.
3. Specify a top-level name for your custom IP variation. Do not include spaces in IP variation names or paths. The parameter editor saves the IP variation settings in a file named `<your_ip>.ip`. Click **OK**. The parameter editor appears.

Figure 25. IP Parameter Editor (Quartus Prime Pro Edition)



4. Set the parameter values in the parameter editor and view the block diagram for the component. The **Parameterization Messages** tab at the bottom displays any errors in IP parameters:
   - Optionally, select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for processing the IP core files in other EDA tools.

   Note: Refer to your IP core user guide for information about specific IP core parameters.
5. Click **Generate HDL**. The **Generation** dialog box appears.

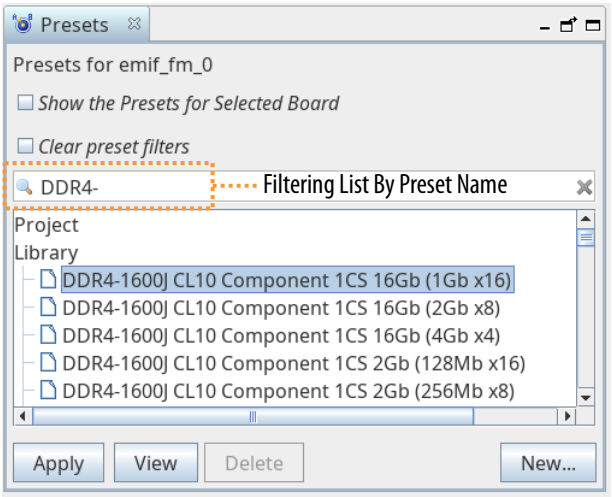
6. Specify output file generation options, and then click **Generate**. The synthesis and simulation files generate according to your specifications.
7. To generate a simulation testbench, click **Generate ➤ Generate Testbench System**. Specify testbench generation options, and then click **Generate**.
8. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate ➤ Show Instantiation Template**.
9. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project.
10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

    Note: Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script*.

4.6.1. Applying Preset Parameters for Specific Applications

The **Preset** tab displays the names of available preset settings for an IP component. A preset is a specific collection of parameter settings that are appropriate for a specific protocol, application, or board. Double-click the preset name (or click **Apply**) to instantly apply the parameter values defined in the preset to the current IP instance.

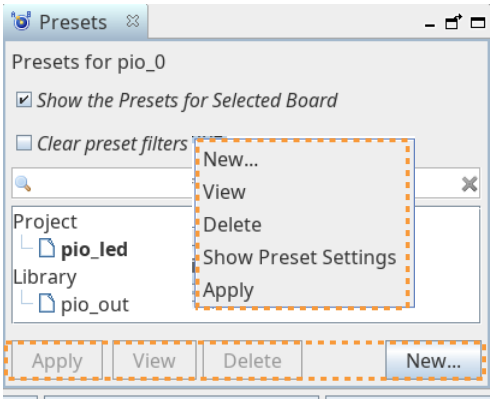
Figure 26. Selecting Preset Parameters



4.6.1.1. Viewing, Applying, and Deleting IP Presets

You can view the properties of a preset, apply a preset, or delete any existing preset in the **Presets** tab.

Figure 27. View, Apply, and Delete Presets in Presets Tab

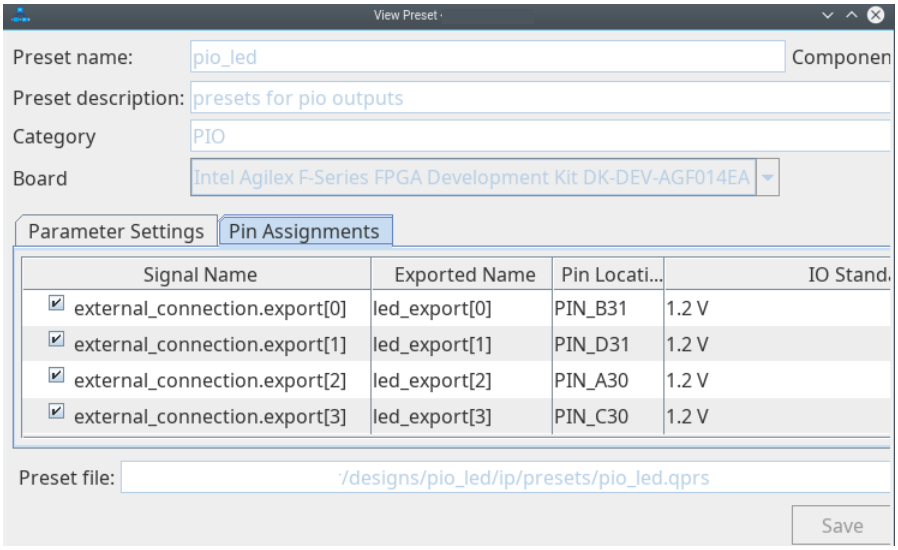


Note: Right-click a preset to access the same **View**, **Apply**, and **Delete** preset functions in the context menu.

Viewing Presets

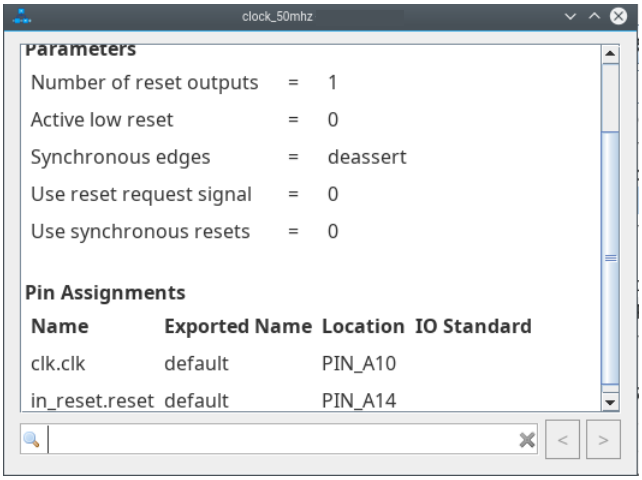
Click the **View** button to show the preset properties in the read-only **Update Preset** dialog box.

Figure 28. View Button Opens View Preset Dialog Box



Right-click a preset and click **Show Preset Settings** to view a searchable report of the preset settings.

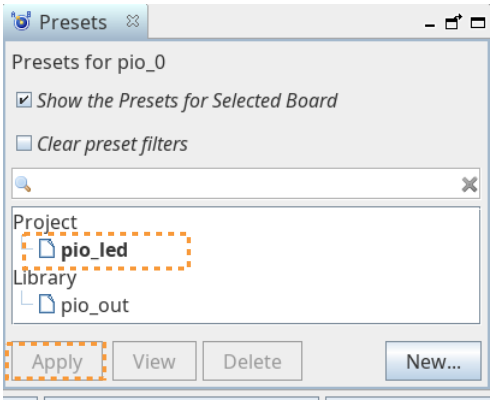
Figure 29. Show Preset Settings Searchable Report



Applying Presets to IP Instances

Click the **Apply** button (or double-click) to apply the IP preset to the currently selected IP. Applied presets appear in bold text.

Figure 30. Applied Presets Appear in Bold Text



Deleting Presets from the System

Click the **Delete** button to delete the current preset from the Platform Designer system.

4.6.2. Customizing IP Presets

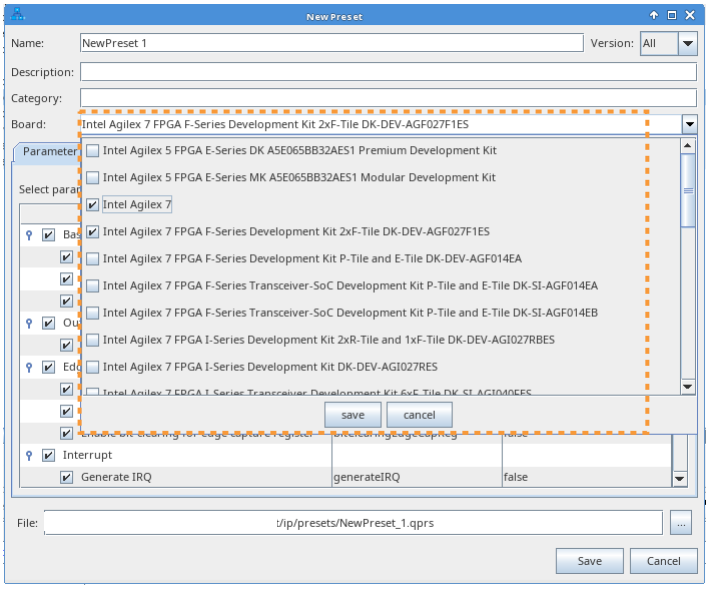
You can optionally define and save a custom set of parameter settings as an IP preset, and then apply the preset whenever you add an instance of the IP component to any system.

Follow these steps to save a custom IP preset:

1. In IP Catalog, double-click any component to launch the parameter editor.
2. To search for a specific preset to base initial settings, type a partial preset name in the search box.
3. In the **Presets** tab, click **New** to specify the **Preset name** and **Preset description**.
4. In the **Board** dropdown, specify the target board. The **Default** setting specifies the current board as the target board for this preset.

   Note: You can specify multiple boards for a preset, provided that the preset parameters and assignments are applicable to all boards in the preset.
5. Under **Select parameters to include in the preset**, enable or disable the parameters you want to include in the preset.
6. Specify the path for the **Preset file** that preserves the collection of parameter settings. The location of the new `.qprs` preset file is added to the IP search path automatically.

Figure 31. Create New Preset



7. Click **Save**.
8. To apply the preset to an IP component, click **Apply**. Preset parameter values that match the current parameter settings appear in bold.

4.6.2.1. Defining Preset Pin Assignments

You can define pin assignments that are included as part of an IP preset. When you apply the IP preset to an IP instance, the pin assignments export during the IP or system's HDL generation.

You define preset pin assignments in the **Pin Assignments** tab of the **New Preset** dialog box, or in a Pin Assignments File (`.tcl`) that you create.

4.6.2.1.1. Defining Preset Pin Assignments in Pin Assignments Tab

The **Pin Assignments** tab allows you to specify the **Exported Name** of the signals, to select the appropriate **Pin Location,** and to select the appropriate **IO Standard** for the target board.

By default, the **Exported Name** name takes the form of:

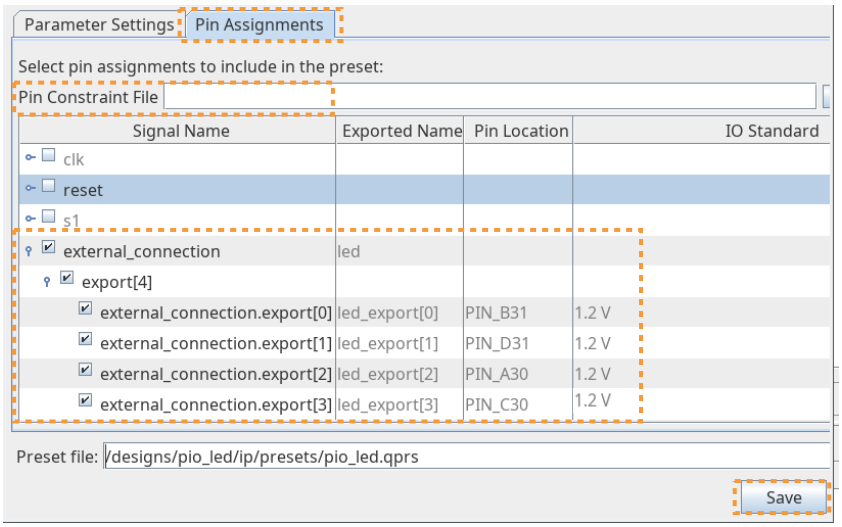
```
module_name + interface_name + pin_role
```

For example:

```
pio0_external_connection_export[0]
```

You can change the **Exported Name** by double-clicking on the **Exported Name** for the interface and typing a new name. All of the signals of the interface then update automatically to reflect the name you specify.

Figure 32. Pin Assignments Tab

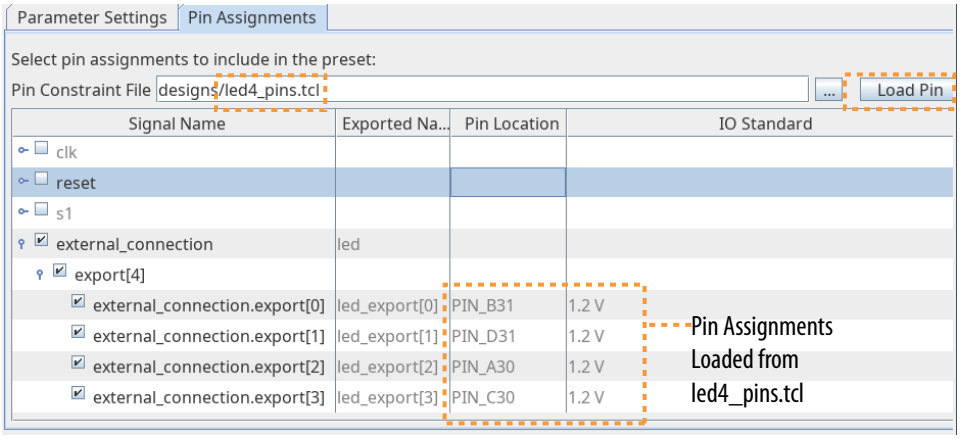


For example, typing `led` for the **external_connection** interface updates the signals of the interface to `led_export[n]`. The **external_connection** is the interface name, and **external_connection_export(0)** is the signal name.

4.6.2.1.2. Defining Preset Pin Assignments in a Pin File

Alternatively, you can specify the pin assignments in a Pin Constraints File (`.tcl`), which can be more efficient for projects with many ports. You specify this `.tcl` file as the **Pin Constraint File** on the **Pin Assignments** tab, and then click **Load Pin**. The **Pin Location** and **IO Standard** update per the loaded pin assignments.

Figure 33. Loading Pin Assignments from Tcl File



The following shows the contents of an example Pin Constraints File (`.tcl`):

```
set_instance_assignment -to "led_export[0]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[0]" "PIN_B31"
set_instance_assignment -to "led_export[1]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[1]" "PIN_D31"
set_instance_assignment -to "led_export[2]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[2]" "PIN_A30"
set_instance_assignment -to "led_export[3]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[3]" "PIN_C30"
```

4.7. IP Core Generation Output (Quartus Prime Pro Edition)

The Quartus Prime software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.

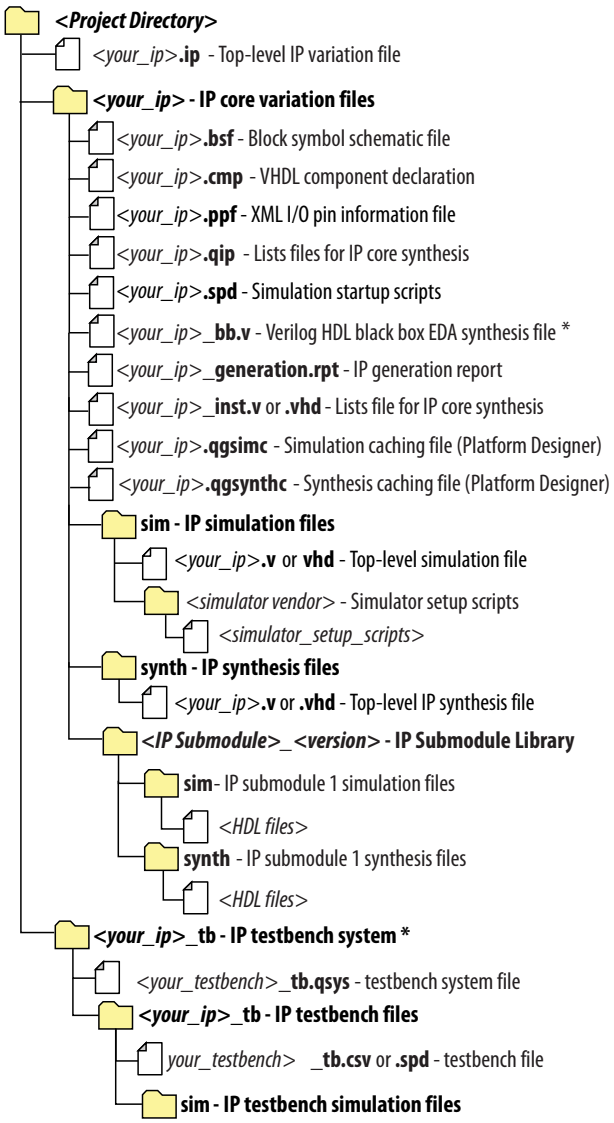
Table 25. Output Files of Intel FPGA IP Generation

File Name	Description
<*your_ip*>.ip	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a `.qsys` file.
<*your_ip*>.cmp	The VHDL Component Declaration (`.cmp`) file is a text file that contains local generic and port definitions that you use in VHDL design files.
<*your_ip*>_generation.rpt	IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.
<*your_ip*>.qgsimc (Platform Designer systems only)	Simulation caching file that compares the `.qsys` and `.ip` files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qgsynth (Platform Designer systems only)	Synthesis caching file that compares the `.qsys` and `.ip` files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.csv	Contains information about the upgrade status of the IP component.
<*your_ip*>.bsf	A symbol representation of the IP variation for use in Block Diagram Files (`.bdf`).

continued...

File Name	Description
<*your_ip*>.spd	Input file that `ip-make-simscript` requires to generate simulation scripts. The `.spd` file contains a list of files you generate for simulation, along with information about memories that you initialize.
<*your_ip*>.ppf	The Pin Planner File (`.ppf`) stores the port and node assignments for IP components you create for use with the Pin Planner.
<*your_ip*>_bb.v	Use the Verilog blackbox (`_bb.v`) file as an empty module declaration for use as a blackbox.
<*your_ip*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*your_ip*>.regmap	If the IP contains register information, the Quartus Prime software generates the `.regmap` file. The `.regmap` file describes the register map information of host and agent interfaces. This file complements the `.sopcinfo` file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*your_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Quartus Prime software stores the `.svd` files for agent interface visible to the System Console hosts in the `.sof` file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system agents, Platform Designer accesses the registers by name.
<*your_ip*>.v <*your_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a `msim_setup.tcl` script to set up and run a simulation with a supported Siemens EDA simulator, such as the QuestaSim simulator.
aldec/	Contains a Riviera-PRO* script `rivierapro_setup.tcl` to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script `vcs_setup.sh` to set up and run a VCS* simulation. Contains a shell script `vcsmx_setup.sh` and `synopsys_sim.setup` file to set up and run a VCS MX simulation.
/xcelium	Contains an Xcelium* Parallel simulator shell script `xcelium_setup.sh` and other setup files to set up and run a simulation.
/submodules	Contains HDL files for the IP core submodule.
<*IP submodule*>/	Platform Designer generates `/synth` and `/sim` sub-directories for each IP submodule directory that Platform Designer generates.

Figure 34. Individual IP Core Generation Output (Quartus Prime Pro Edition)



* If supported and enabled for your IP core variation.

4.8. Scripting IP Core Generation

Use the `qsys-script` and `qsys-generate` utilities to define and generate an IP core variation outside of the Quartus Prime GUI.

To parameterize and generate an IP core at command-line, follow these steps:

1. Run `qsys-script` to start a Tcl script that instantiates the IP and sets parameters:

```
qsys-script --script=<script_file>.tcl
```

2. Run `qsys-generate` to generate the IP core variation:

```
qsys-generate <IP variation file>.qsys
```

4.9. Modifying an IP Variation

After generating an IP core variation, use any of the following methods to modify the IP variation in the parameter editor.

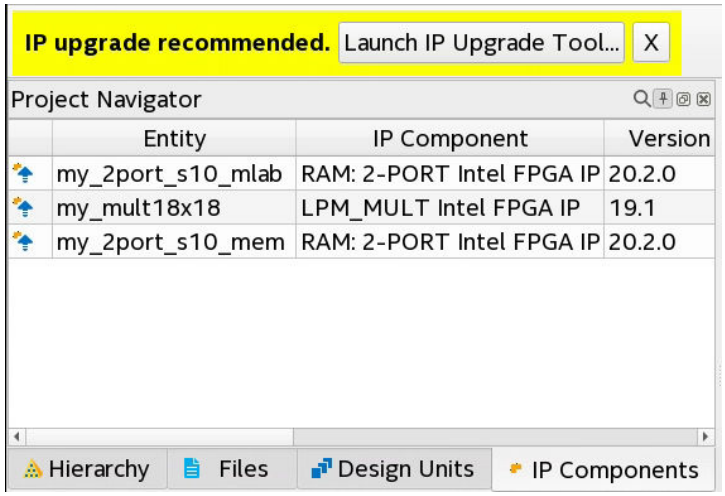
Table 26. Modifying an IP Variation

Menu Command	Action
File ➤ Open	Select the top-level HDL (`.v`, or `.vhd`) IP variation file to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes.
View ➤ Project Navigator ➤ IP Components	Double-click the IP variation to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes.
Project ➤ Upgrade IP Components	Select the IP variation and click **Upgrade in Editor** to launch the parameter editor and modify the IP variation. Regenerate the IP variation to implement your changes.

4.10. Upgrading IP Cores

Any Intel FPGA IP variations that you generate from a previous version or different edition of the Quartus Prime software, may require upgrade before compilation in the current software edition or version. The Project Navigator displays a banner indicating the IP upgrade status. Click **Launch IP Upgrade Tool** or **Project ➤ Upgrade IP Components** to upgrade outdated IP cores.

Figure 35. IP Upgrade Alert in Project Navigator



Icons in the **Upgrade IP Components** dialog box indicate when IP upgrade is required, optional, or unsupported for an IP variation in the project. Upgrade IP variations that require upgrade before compilation in the current version of the Quartus Prime software.

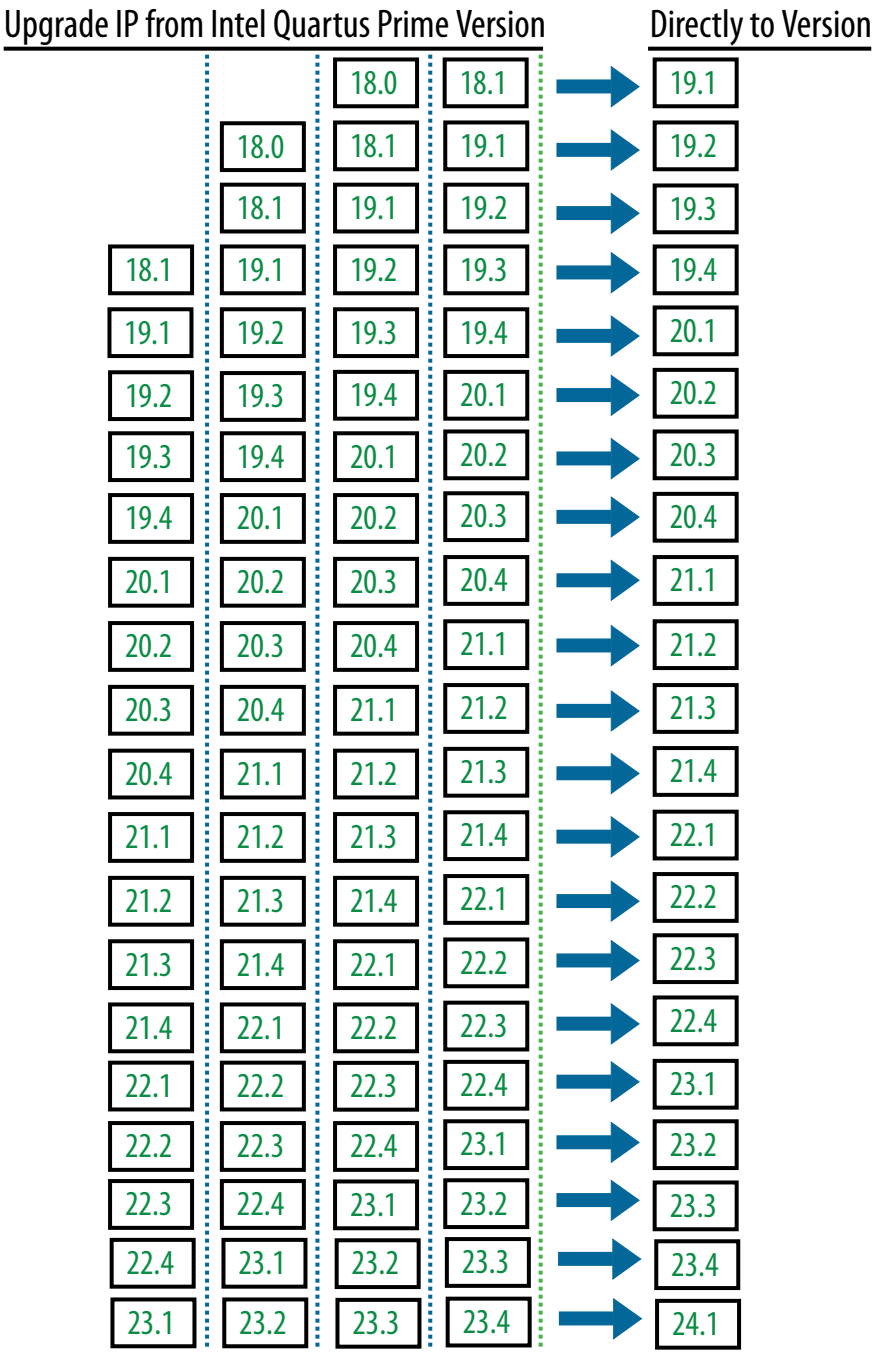
Note: Upgrading IP cores may append a unique identifier to the original IP core entity names, without similarly modifying the IP instance name. There is no requirement to update these entity references in any supporting Quartus Prime file, such as the Quartus Prime Settings File (`.qsf`), Synopsys* Design Constraints File (`.sdc`), or Signal Tap File (`.stp`), if these files contain instance names. The Quartus Prime software reads only the instance name and ignores the entity name in paths that specify both names. Use only instance names in assignments.

Table 27. IP Core Upgrade Status

IP Core Status	Description
IP Upgraded 	Indicates that your IP variation uses the latest version of the Intel FPGA IP core.
IP Component Outdated 	Indicates that your IP variation uses an outdated version of the IP core.
IP End of Life 	Indicates that Intel designates the IP core as end-of-life status. You may or may not be able to edit the IP core in the parameter editor. Support for this IP core discontinues in future releases of the Quartus Prime software.
IP Upgrade Mismatch Warning 	Provides warning of non-critical IP core differences in migrating IP to another device family.
IP has incompatible subcores 	Indicates that the current version of the Quartus Prime software does not support compilation of your IP variation, because the IP has incompatible subcores.
Compilation of IP Not Supported 	Indicates that the current version of the Quartus Prime software does not support compilation of your IP variation. This can occur if another edition of the Quartus Prime software, such as the Quartus Prime Standard Edition, generated this IP. Replace this IP component with a compatible component in the current edition.

Note: Beginning with the Quartus Prime Pro Edition software version 19.1, IP upgrade supports migration of IP released within one year of the Quartus Prime Pro Edition software version, as the following chart defines:

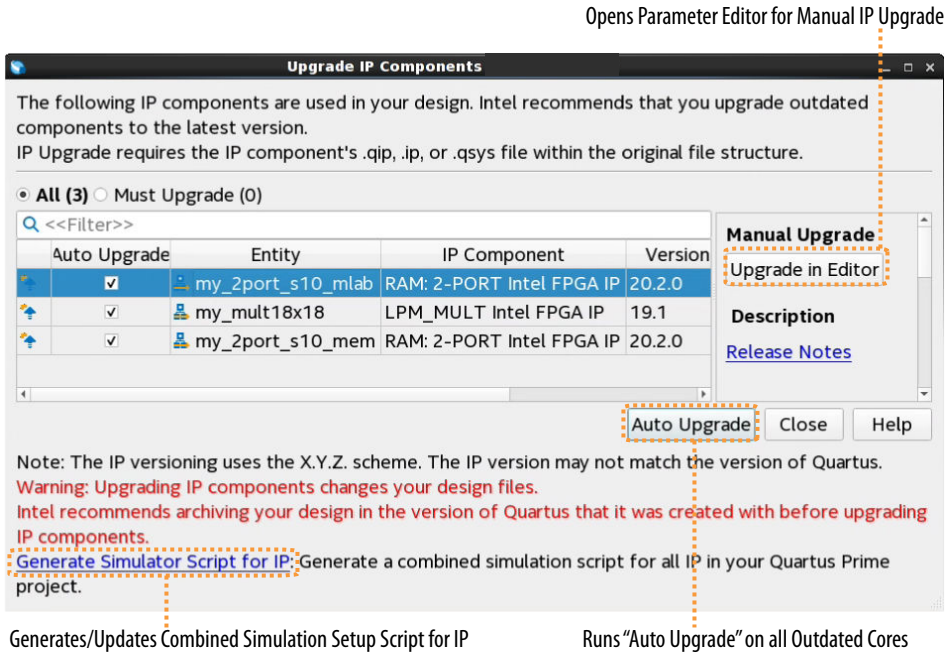
Figure 36. Quartus Prime Pro Edition IP Version Upgrade Paths



Follow these steps to upgrade IP cores:

1. In the latest version of the Quartus Prime software, open the Quartus Prime project containing an outdated IP core variation. The **Upgrade IP Components** dialog box automatically displays the status of IP cores in your project, along with instructions for upgrading each core. To access this dialog box manually, click **Project ➤ Upgrade IP Components**.
2. To upgrade one or more IP cores that support automatic upgrade, ensure that you turn on the **Auto Upgrade** option for the IP cores, and click **Auto Upgrade**. The **Status** and **Version** columns update when upgrade is complete. Example designs that any Intel FPGA IP core provides regenerate automatically whenever you upgrade an IP core.
3. To manually upgrade an individual IP core, select the IP core and click **Upgrade in Editor** (or simply double-click the IP core name). The parameter editor opens, allowing you to adjust parameters and regenerate the latest version of the IP core.

Figure 37. Upgrading IP Cores (Quartus Prime Pro Edition Example)



Note: Intel FPGA IP cores older than Quartus Prime software version 12.0 do not support upgrade. Intel verifies that the current version of the Quartus Prime software compiles the previous two versions of each IP core. The *Intel FPGA IP Core Release Notes* reports any verification exceptions for Intel FPGA IP cores. Intel does not verify compilation for IP cores older than the previous two releases.

Related Information

Intel FPGA IP Release Notes

4.10.1. Upgrading IP Cores at Command-Line

Optionally, upgrade an Intel FPGA IP core at the command-line, rather than using the GUI. IP cores that do not support automatic upgrade do not support command-line upgrade.

- To upgrade a single IP core at the command-line, type the following command:

```
quartus_sh –ip_upgrade –variation_files <my_ip>.<qsys,.v, .vhd> \
     <quartus_project>

Example:
quartus_sh -ip_upgrade -variation_files mega/pll25.qsys hps_testx
```

- To simultaneously upgrade multiple IP cores at the command-line, type the following command:

```
quartus_sh –ip_upgrade –variation_files “<my_ip1>.<qsys,.v, .vhd>> \
     ;  <my_ip_filepath/my_ip2>.<hdl>”  <quartus_project>

Example:
quartus_sh -ip_upgrade -variation_files "mega/pll_tx2.qsys;mega/
pll3.qsys" hps_testx
```

4.10.2. Migrating IP Cores to a Different Device

Migrate an Intel FPGA IP variation when you want to target a different (often newer) device. Most Intel FPGA IP cores support automatic migration. Some IP cores require manual IP regeneration for migration. A few IP cores do not support device migration, requiring you to replace them in the project. The **Upgrade IP Components** dialog box identifies the migration support level for each IP core in the design.

1. To display the IP cores that require migration, click **Project ➤ Upgrade IP Components**. The **Description** field provides migration instructions and version differences.
2. To migrate one or more IP cores that support automatic upgrade, ensure that the **Auto Upgrade** option is turned on for the IP cores, and click **Perform Automatic Upgrade**. The **Status** and **Version** columns update when upgrade is complete.
3. To migrate an IP core that does not support automatic upgrade, double-click the IP core name, and click **OK**. The parameter editor appears. If the parameter editor specifies a **Currently selected device family**, turn off **Match project/default**, and then select the new target device family.
4. Click **Generate HDL**, and confirm the **Synthesis** and **Simulation** file options. Verilog HDL is the default output file format. If you specify VHDL as the output format, select **VHDL** to retain the original output format.
5. Click **Finish** to complete migration of the IP core. Click **OK** if the software prompts you to overwrite IP core files. The **Device Family** column displays the new target device name when migration is complete.
6. To ensure correctness, review the latest parameters in the parameter editor or generated HDL.

Note: IP migration may change ports, parameters, or functionality of the IP variation. These changes may require you to modify your design or to re-parameterize your IP variant. During migration, the IP variation's HDL generates into a library that is different from the original output location of the IP core. Update any assignments that reference outdated locations. If a symbol in a supporting Block Design File schematic represents your upgraded IP core, replace the symbol with the newly generated `<my_ip>.bsf`. Migration of some IP cores requires installed support for the original and migration device families.

Related Information

Intel FPGA IP Release Notes

4.10.3. Troubleshooting IP or Platform Designer System Upgrade

The **Upgrade IP Components** dialog box reports the version and status of each Intel FPGA IP core and Platform Designer system following upgrade or migration.

If any upgrade or migration fails, the **Upgrade IP Components** dialog box provides information to help you resolve any errors.

Note: Do not use spaces in IP variation names or paths.

During automatic or manual upgrade, the Messages window dynamically displays upgrade information for each IP core or Platform Designer system. Use the following information to resolve upgrade errors:

Table 28. IP Upgrade Error Information

Upgrade IP Components Field	Description
Status	Displays the "Success" or "Failed" status of each upgrade or migration. Click the status of any upgrade that fails to open the **IP Upgrade Report**.
Version	Dynamically updates the version number when upgrade is successful. The text is red when the IP requires upgrade.
Device Family	Dynamically updates to the new device family when migration is successful. The text is red when the IP core requires upgrade.
Auto Upgrade	Runs automatic upgrade on all IP cores that support auto upgrade. Also, automatically generates a `<Project Directory>/ip_upgrade_port_diff_reports` report for IP cores or Platform Designer systems that fail upgrade. Review these reports to determine any port differences between the current and previous IP core version.

Use the following techniques to resolve errors if your IP core or Platform Designer system "Failed" to upgrade versions or migrate to another device. Review and implement the instructions in the **Description** field, including one or more of the following:

- If the current version of the software does not support the IP variant, right-click the component and click **Remove IP Component from Project**. Replace this IP core or Platform Designer system with the one supported in the current version of the software.
- If the current target device does not support the IP variant, select a supported device family for the project, or replace the IP variant with a suitable replacement that supports your target device.
- If an upgrade or migration fails, click **Failed** in the **Status** field to display and review details of the **IP Upgrade Report**. Click the **Release Notes** link for the latest known issues about the IP core. Use this information to determine the nature of the upgrade or migration failure and make corrections before upgrade.
- Run **Auto Upgrade** to automatically generate an **IP Ports Diff** report for each IP core or Platform Designer system that you upgrade. Review the reports to determine any port differences between the current and previous IP core version. Click **Upgrade in Editor** to make specific port changes and regenerate your IP core or Platform Designer system.
- If your IP core or Platform Designer system does not support **Auto Upgrade**, click **Upgrade in Editor** to resolve errors and regenerate the component in the parameter editor.

Figure 38. IP Port Differences Report

```
IP Ports Diff Report for pattern_generator_system_mm_bridge
Tue August 20 12:13:05 2020
Quartus Prime Version 20.3.0 Pro Edition


--------------------
; Table of Contents ;
--------------------
  1. IP Information
  2. IP Ports Diff Table
```

Report Summary

```
+-----------------------------------------------------------------------------------------------------------------------------------+
; IP Information                                                                                                                     ;
+---------------------+-------------------------------------------------------------------------------------------------------------+
; IP Variation Name    ; pattern_generator_system_mm_bridge                                                                          ;
; Prior Version        ; 19.3                                                                                                        ;
; New Version          ; 20.3                                                                                                        ;
; New File             ; /mydata/synth/pattern_generator_system_mm_bridge.v ;
; Has Port Differences ; Yes                                                                                                         ;
+---------------------+-------------------------------------------------------------------------------------------------------------+
```

IP Port Differences

```
+-----------------------------------------------------------------------------------------------------------------------------------+
; IP Ports Diff Table                                                                                                               ;
+-------------+-------------+-----------------+----------------+----------------------+--------------------+
; Change Type ; Port Name    ; Prior Port Range ; New Port Range ; Prior Port direction ; New Port direction ;
+-------------+-------------+-----------------+----------------+----------------------+--------------------+
; Modified    ; m0_address   ; [10:0]          ; [0:0]          ; output               ; output             ;
; Modified    ; m0_readdata  ; [31:0]          ; [0:0]          ; input                ; input              ;
; Modified    ; m0_writedata ; [31:0]          ; [0:0]          ; output               ; output             ;
; Modified    ; s0_address   ; [10:0]          ; [0:0]          ; input                ; input              ;
; Modified    ; s0_readdata  ; [31:0]          ; [0:0]          ; output               ; output             ;
; Modified    ; s0_writedata ; [31:0]          ; [0:0]          ; input                ; input              ;
+-------------+-------------+-----------------+----------------+----------------------+--------------------+
```

4.11. Simulating Intel FPGA IP Cores

The Quartus Prime software supports IP core RTL simulation in specific EDA simulators. IP generation optionally creates simulation files, including the functional simulation model, any testbench (or example design), and vendor-specific simulator

setup scripts for each IP core. You can use the functional simulation model and any testbench or example design for simulation. IP generation output may also include scripts to compile and run any testbench. The scripts list all models or libraries you require to simulate your IP core.

The Quartus Prime software provides integration with many simulators and supports multiple simulation flows, including your own scripted and custom simulation flows. Whichever flow you choose, IP core simulation involves the following steps:

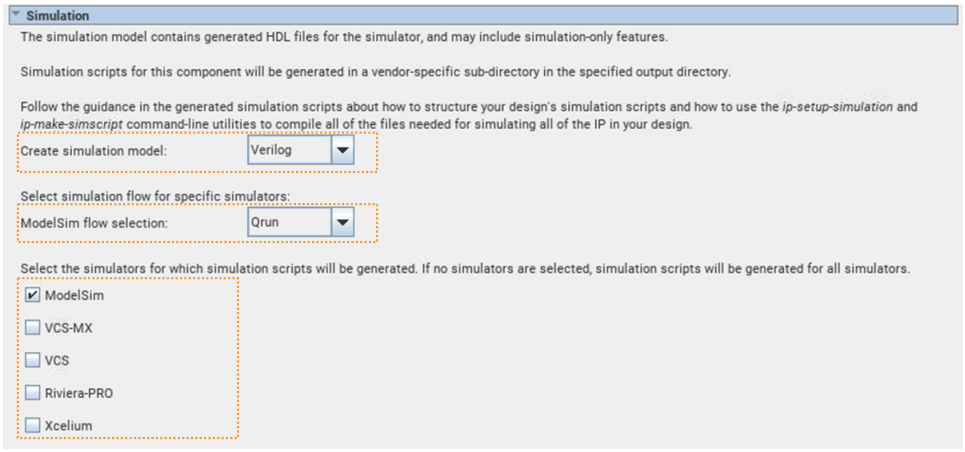
1. Generate IP HDL, testbench (or example design), and simulator setup script files.
2. Set up your simulator environment and any simulation scripts.
3. Compile simulation model libraries.
4. Run your simulator.

4.11.1. Generating IP Simulation Files

The Quartus Prime software optionally generates the functional simulation model, any testbench (or example design), and vendor-specific simulator setup scripts when you generate an IP core. To specify options for the generation of IP simulation files, follow these steps:

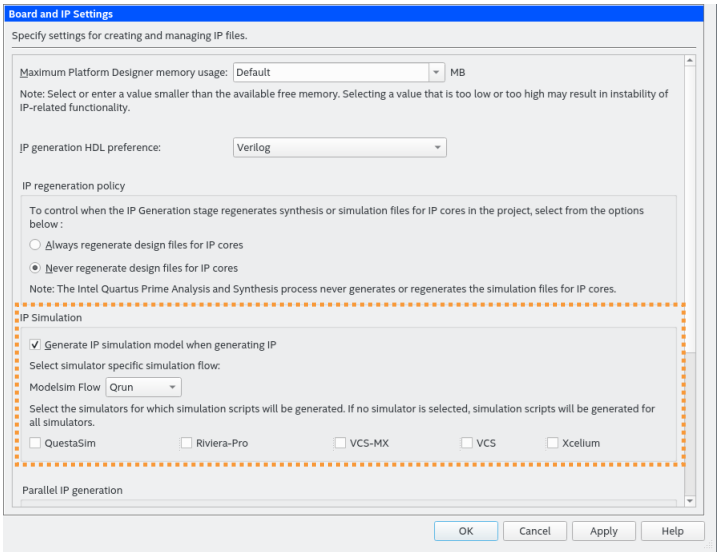
- To specify your supported simulator and options for design simulation file generation, click **Assignment ➤ Settings ➤ EDA Tool Settings ➤ Simulation**.

Figure 39. Simulation Options in Generation Dialog Box



- To specify your supported simulator and options for IP simulation file generation, click **Assignments ➤ Settings ➤ Board and IP Settings ➤ IP Simulation** and specify the following:
  - To enable automatic generation of simulation models for all IP in the project when you generate IP during compilation, turn on the **Generate IP simulation model when generating IP** option.
  - To specify one or more supported simulators for which to generate setup scripts, turn on one or more simulator option, or disable all simulator options to generate scripts for all simulators automatically.

Figure 40. Project-Wide IP Generation Settings



- To generate the simulation files, click **Processing ➤ Start Compilation** to compile the design. The simulation models and setup scripts for the Intel FPGA IP generate in the *<your_project>*/*<ip name>*/sim/*<vendor>* directory.

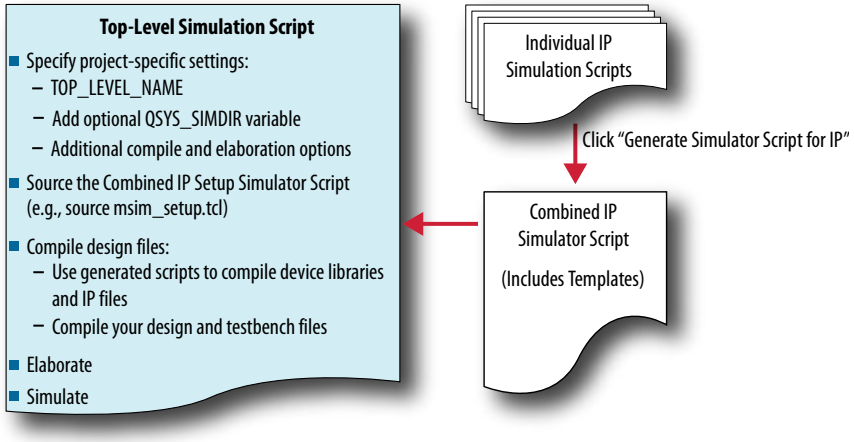
You can optionally override these project-level **IP Settings** when you generate HDL for individual IP cores with the IP Parameter Editor. Prior to generation, you can specify a supported simulator, or specify no simulator to generate the setup scripts for all simulators in the parameter editor.

4.11.2. Scripting IP Simulation

The Quartus Prime software supports the use of scripts to automate simulation processing in your preferred simulation environment. Use the scripting methodology that you prefer to control simulation.

Use a version-independent, top-level simulation script to control design, testbench, and IP core simulation. Because Quartus Prime-generated simulation file names may change after IP upgrade or regeneration, your top-level simulation script must "source" the generated setup scripts, rather than using the generated setup scripts directly. Follow these steps to generate or regenerate combined simulator setup scripts:

Figure 41. Incorporating Generated Simulator Setup Scripts into a Top-Level Simulation Script



1. Click **Tools ➤ Generate Simulator Script for IP** (or run the `ip-setup-simulation` utility) to generate or regenerate a combined simulator setup script for all IP for each simulator.
2. Use the templates in the generated script to source the combined script in your top-level simulation script. Each simulator's combined script file contains a rudimentary template that you adapt for integration of the setup script into a top-level simulation script.

   This technique eliminates manual update of simulation scripts if you modify or upgrade the IP variation.

4.11.2.1. Generating a Combined Simulator Setup Script

You can run the **Generate Simulator Setup Script for IP** command to generate a combined simulator setup script.

You can then source this combined script from a top-level simulation script. Click **Tools ➤ Generate Simulator Setup Script for IP** (or use of the `ip-setup-simulation` utility at the command-line) to generate or update the combined scripts, after any of the following occur:

- IP core initial generation or regeneration with new parameters
- Quartus Prime software version upgrade
- IP core version upgrade

Table 29. ip-setup-simulation Utility

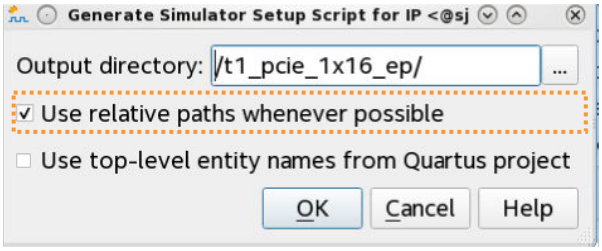
Utility	Syntax
`ip-setup-simulation` generates a combined, version-independent simulation script for all Intel FPGA IP cores in your project. The command also automates regeneration of the script after upgrading software or IP versions. Use the `compile-to-work` option to compile all simulation files into a single work library if your simulation environment requires. Use the `--use-relative-paths` option to use relative paths whenever possible.	`ip-setup-simulation` `--quartus-project=<my proj>` `--output-directory=<my_dir>` `--use-relative-paths` `--compile-to-work` `--use-relative-paths` and `--compile-to-work` are optional. For command-line help listing all options for these executables, type: `<utility name> --help`.

To generate a combined simulator setup script for all project IP cores for each simulator:[4]

1. Click **Tools ➤ Generate Simulator Setup Script for IP** (or run the `ip-setup-simulation` utility). Specify the **Output Directory** and library compilation options. Click **OK** to generate the file. By default, the files generate into the */<project directory>/<simulator>/* directory using relative paths.

   Note: For designs with F-tile IP, do not turn on the **Use top-level entity names from Quartus project** option.

Figure 42. Generate Simulator Setup Script for IP Dialog Box



2. To incorporate the generated simulator setup script into your top-level simulation script, refer to the template section in the generated simulator setup script as a guide to creating a top-level script:
   a. Copy the specified template sections from the simulator-specific generated scripts and paste them into a new top-level file.
   b. Remove the comments at the beginning of each line from the copied template sections.
   c. Specify the customizations you require to match your design simulation requirements, for example:
      - Specify the TOP_LEVEL_NAME variable to the design's simulation top-level file. The top-level entity of your simulation is often a testbench that instantiates your design. Then, your design instantiates IP cores or Platform Designer systems. Set the value of TOP_LEVEL_NAME to the top-level entity.
      - If necessary, set the QSYS_SIMDIR variable to point to the location of the generated IP simulation files.
      - Specify any other changes, such as using the `grep` command-line utility to search a transcript file for error signatures, or e-mail a report.
3. Re-run **Tools ➤ Generate Simulator Setup Script for IP** (or `ip-setup-simulation`) after regeneration of an IP variation.

Related Information

Quartus Prime Pro Edition User Guide: Third-party Simulation

[4] If your design contains one or more F-tile IPs, you must first perform **Start Analysis & Elaboration** and then **Support-Logic Generation** before performing these steps.

4.12. Generating Simulation Files for Platform Designer Systems and IP Variants

If your design contains Intel FPGA IP or a Platform Designer system, you must first generate files for RTL simulation of the IP or system with the Quartus Prime Platform Designer before running simulation.

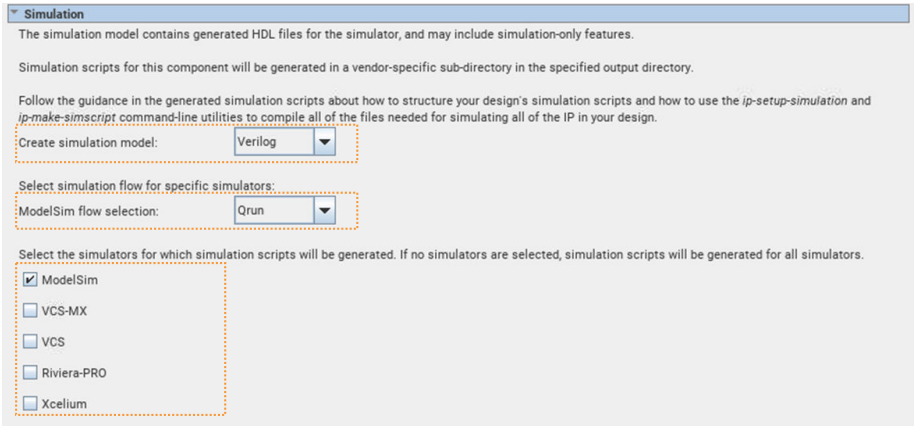
When you generate the system (or IP variant), Platform Designer optionally creates simulation files, including the functional simulation model, any testbench (or example design), and vendor-specific simulator setup scripts for each IP core.

You can use the functional simulation model and any testbench or example design for simulation of the IP or system. The IP generation output may also include scripts to compile and run any testbench. The scripts list all models or libraries you require to simulate your IP core.

To generate the simulation model and simulator setup scripts for your Platform Designer system or component, follow these steps:

1. Click **Tools ➤ Platform Designer**. Platform Designer and open or create a Platform Designer system or IP variant.
2. In Platform Designer, after specifying parameters, click **Generate ➤ Generate HDL**. The **Generation** dialog box appears.
3. Under **Simulation**, specify **Verilog** or **VHDL** for the **Create simulation model** option.

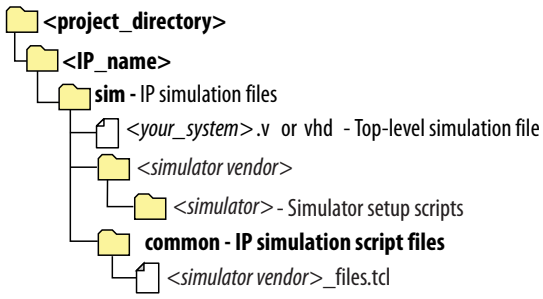
Figure 43. Simulation Options in Generation Dialog Box



4. If you want to specifically use ModelSim*, specify **Traditional** or **Qrun** for the ModelSim flow option. Otherwise, **Qrun** flow is the default selection.
5. Turn on or off the **ModelSim**, **VCS-MX**, **VCS**, **Riviera-Pro**, or **Xcelium** option to generate simulator setup scripts for the simulation tool. If you turn on no simulator options, the scripts generate for all simulators.
6. Click the **Generate** button. Platform Designer generates the simulation models and setup scripts for your system or IP component in the following directory:

```
<top-level system name>/<system name>/<sim>/<simulator>
```

Figure 44. Generated Simulation Files Location



By default, Platform Designer generates the simulation scripts for the currently loaded system and all subsystems. Alternatively, you can open a subsystem to generate a simulation script only for that subsystem.

You can use scripts to compile the required device libraries and system design files in the correct order and elaborate or load the top-level system for simulation.

Table 30. Simulation Script Variables

The simulation scripts provide variables that allow flexibility in your simulation environment.

Variable	Description
TOP_LEVEL_NAME	If the testbench Platform Designer system is not the top-level instance in your simulation environment because you instantiate the Platform Designer testbench within your own top-level simulation file, set the TOP_LEVEL_NAME variable to the top-level hierarchy name.
QSYS_SIMDIR	If the simulation files generated by Platform Designer are not in the simulation working directory, use the QSYS_SIMDIR variable to specify the directory location of the Platform Designer simulation files.
QUARTUS_INSTALL_DIR	Points to the Quartus installation directory that contains the device family library.

Example 6. Top-Level Simulation HDL File for a Testbench System

The example below shows the `pattern_generator_tb` generated for a Platform Designer system called `pattern_generator`. The `top.sv` file defines the top-level module that instantiates the `pattern_generator_tb` simulation model, as well as a custom SystemVerilog test program with BFM transactions, called `test_program`.

```
module top();
 pattern_generator_tb tb();
 test_program pgm();
endmodule
```

4.13. Synthesizing IP Cores in Other EDA Tools

Optionally, use another supported EDA tool to synthesize a design that includes Intel FPGA IP cores. When you generate the IP core synthesis files for use with third-party EDA synthesis tools, you can create an area and timing estimation netlist. To enable generation, turn on **Create timing and resource estimates for third-party EDA synthesis tools** when customizing your IP variation.

The area and timing estimation netlist describes the IP core connectivity and architecture, but does not include details about the true functionality. This information enables certain third-party synthesis tools to better report area and timing estimates. In addition, synthesis tools can use the timing information to achieve timing-driven optimizations and improve the quality of results.

The Quartus Prime software generates the <*variant name*>_syn.v netlist file in Verilog HDL format, regardless of the output file format you specify. If you use this netlist for synthesis, you must include the IP core wrapper file <*variant name*>.v or <*variant name*> .vhd in your Quartus Prime project.

4.14. Instantiating IP Cores in HDL

Instantiate an IP core directly in your HDL code by calling the IP core name and declaring the IP core's parameters. This approach is similar to instantiating any other module, component, or subdesign. When instantiating an IP core in VHDL, you must include the associated libraries.

4.14.1. Example Top-Level Verilog HDL Module

Verilog HDL ALTFP_MULT in Top-Level Module with One Input Connected to Multiplexer.

```
module MF_top (a, b, sel, datab, clock, result);
       input [31:0] a, b, datab;
       input clock, sel;
       output [31:0] result;
       wire [31:0] wire_dataa;

       assign wire_dataa = (sel)? a : b;
       altfp_mult inst1
(.dataa(wire_dataa), .datab(datab), .clock(clock), .result(result));

       defparam
               inst1.pipeline = 11,
               inst1.width_exp = 8,
               inst1.width_man = 23,
               inst1.exception_handling = "no";
endmodule
```

4.14.2. Example Top-Level VHDL Module

VHDL ALTFP_MULT in Top-Level Module with One Input Connected to Multiplexer.

```
library ieee;
use ieee.std_logic_1164.all;
library altera_mf;
use altera_mf.altera_mf_components.all;

entity MF_top is
        port (clock, sel  : in  std_logic;
              a, b, datab : in  std_logic_vector(31 downto 0);
              result      : out std_logic_vector(31 downto 0));
end entity;

architecture arch_MF_top of MF_top is
signal wire_dataa : std_logic_vector(31 downto 0);
begin

wire_dataa <= a when (sel = '1') else b;

inst1 : altfp_mult
```

```
        generic map    (
                  pipeline => 11,
                  width_exp => 8,
                  width_man => 23,
                  exception_handling => "no")
        port map (
                  dataa => wire_dataa,
                  datab => datab,
                  clock => clock,
                  result => result);
end arch_MF_top;
```

4.15. Support for the IEEE 1735 Encryption Standard

The Quartus Prime Pro Edition software supports the IEEE 1735 v1 encryption standard for IP core file decryption. You can encrypt the Verilog HDL or VHDL IP files with the `encrypt_1735` utility, or with a third-party encryption tool that supports the IEEE 1735 standard. You can then use the encrypted files in the Quartus Prime Pro Edition software and simulation tools that support the IEEE 1735 encryption standard.

The encryption key is the same for Verilog HDL and VHDL. You can pass parameters to the instantiation of an encrypted module using the same method as a non-encrypted module.

Type `encrypt_1735 --help` at the Quartus Prime command line to view syntax and all supported options for the `encrypt_1735` utility.

```
encrypt_1735 [-h | --help[=<option|topic>] | -v]
encrypt_1735 <other options>

Options:
--------
    -?
    -f <argument file>
    -h
    --256_bit[=<value>]
    --help[=<option|topic>]
    --language=<verilog | systemverilog| vhdl>
    --lower_priority
    --of=<some_file>
    --quartus
    --simulation[=<aldec | cadence | mentor | synopsys (comma delimited)>]
    --tcl_jou_file=<[tcl_jou_filename=]on|off>
    --tcl_log_file=<[tcl_log_filename=]on|off>
```

Adding the following Verilog or VHDL pragma to your RTL, along with the public key, enables the Quartus Prime software to use the key to decrypt IP core files.

Verilog/SystemVerilog Encryption Pragma (Third-Party Tools):

```
`pragma protect key_keyowner="Intel Corporation"
`pragma protect data_method="aes128-cbc"
`pragma protect key_method="rsa"
`pragma protect key_keyname="Intel-FPGA-Quartus-RSA-1"
`pragma protect key_public_key
<encrypted session key>

`pragma protect begin
`pragma protect end
```

VHDL Encryption Pragma (Third-Party Tools):

```
`protect key_keyowner = “Intel Corporation”
`protect data_method="aes128-cbc"
`protect key_method = “rsa”
`protect key_keyname = “Intel-FPGA-Quartus-RSA-1”
`protect key_block
<Encrypted session key>
```

Only file encryption with a third-party tool requires the public encryption key. File encryption with the Quartus Prime Pro Edition software does not require the public encryption key.

Use one of the following methods to obtain the public encryption key:

- If you are using the Quartus Prime Pro Edition software version 19.3 or later, the public encryption key is in *<install directory>*\quartus\common\misc\public_key.
- If you are using a version of the Quartus Prime Pro Edition software earlier than version 19.3, to obtain the encryption key, login or register for a My-Intel account, and then submit an Intel Premier Support case requesting the encryption key.
- If you are ineligible for Intel Premier Support, you can submit a question regarding the "IEEE 1735 Encryption Public Key" to the Intel Community Forum for assistance.

Note: The Quartus Prime Standard Edition software does not support IEEE 1735 encryption.

Related Information

- My-Intel.com
- Intel Community Forum

4.16. Related Trainings and Resources

You can take up the following training to help you understand how you can work with Intel FPGA IP cores:

- Creating Reusable Design Blocks: Introduction to IP Reuse with the Quartus Prime Software
- Creating Reusable Design Blocks: IP Integration with the Quartus Prime Software
- Creating Reusable Design Blocks: IP Design & Implementation with the Quartus Prime Software

For other IP-related trainings, review the Intel FPGA Training Catalog and YouTube*'s Intel FPGA channel.

Also, refer to the Embedded Peripherals IP User Guide for detailed description of the IP cores that the Quartus Prime design software provides.

5. Managing Quartus Prime Projects

The Quartus Prime software organizes and manages the elements of your design within a *project*. The project encapsulates information about your design files, hierarchy, libraries, constraints, and project settings. This chapter describes the basics of working with Quartus Prime software projects, including viewing project information, adding design files and constraints, and viewing and exporting the design compilation results.

After you create or open a project, the GUI displays integrated information and controls for the open project.

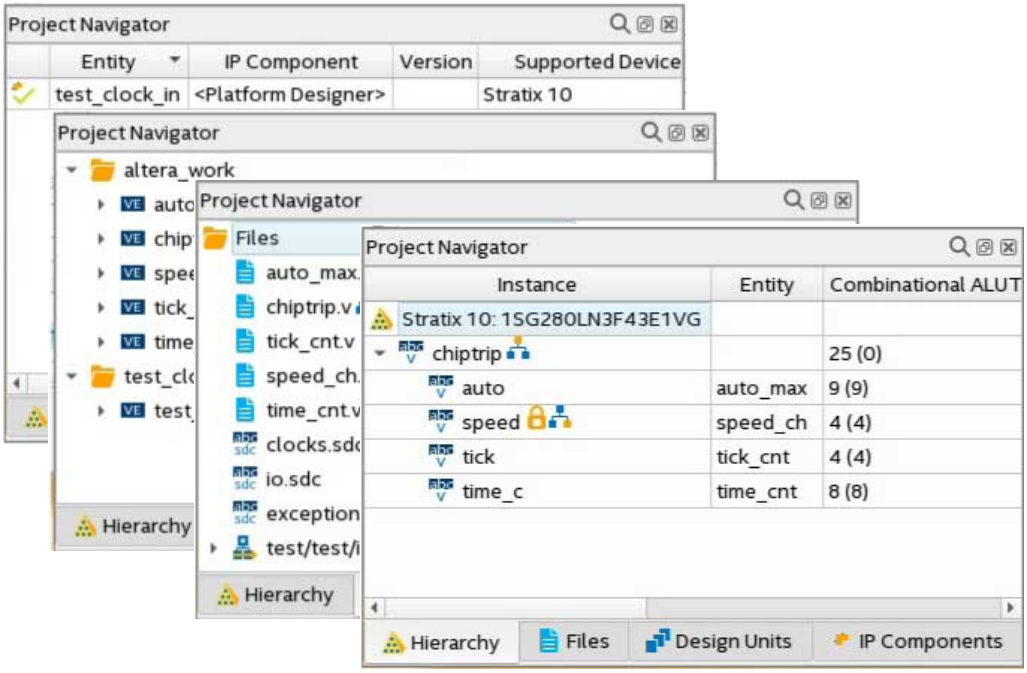
Navigating Content Through Tasks

Use the following navigation diagram to navigate this guide through user-tasks:

5.1. Viewing Basic Project Information

View basic information about your project in the Project Navigator, the **Tasks** pane, Compilation Dashboard, Report panel, and **Messages** window.

Figure 45. Project Navigator Hierarchy, Files, Design Units, and IP Components Tabs



The Project Navigator

The **Project Navigator** (**View ➤ Project Navigator**) displays the elements of your project, such as the design files, IP components, and your project hierarchy (after elaboration). Right-click in the **Project Navigator** to locate the elements of your project. Project information appears on the **Files**, **Hierarchy**, **Design Units**, and **IP Components** tabs.

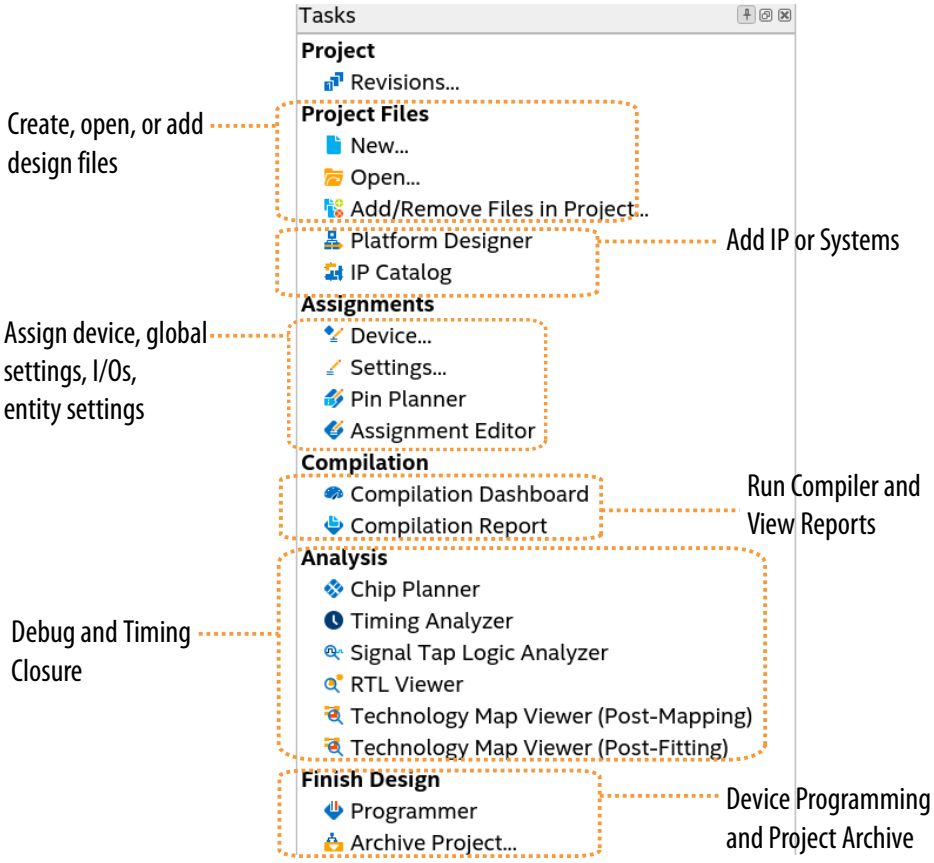
Table 31. Project Navigator Tabs

Project Navigator Tab	Description
Files	Lists all design files in the current project. Right-click design files in this tab to run these commands: • **Open** the file • **Remove** the file from project • View file **Properties**
Hierarchy	Provides a visual representation of the project hierarchy, specific resource usage information, and device and device family information. Right-click items in the hierarchy to **Locate**, **Set as Top-Level Entity**, or define Logic Lock regions or design partitions.
Design Units	Displays the design units in the project. Right-click a design unit to **Locate in Design File**.
IP Components	Displays the design files that make up the IP instantiated in the project, including Intel FPGA IP, Platform Designer components, and third-party IP. Click **Launch IP Upgrade Tool** from this tab to upgrade outdated IP components.

Project Tasks Pane

The **Tasks** pane (**View ➤ Tasks**) launches common project tasks, such as creating design files, adding IP, running compilation, and device programming.

Figure 46. Tasks Pane

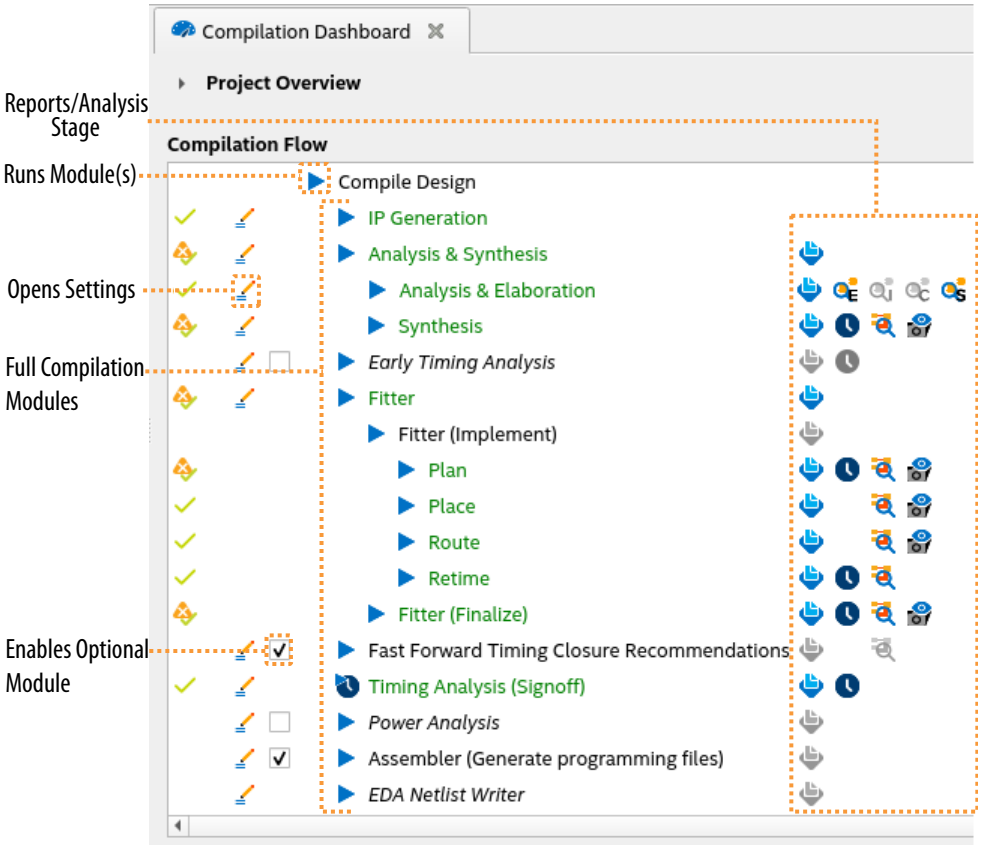


5.1.1. Using the Compilation Dashboard

The Compilation Dashboard provides immediate access to settings, controls, and reporting for each stage of the compilation flow.

The Compilation Dashboard appears by default when you open a project, or you can click **Compilation Dashboard** in the Tasks window to re-open it.

Figure 47. Compilation Dashboard



- Click the **Pencil** icon to edit settings for that stage of the compilation flow.
- Click any Compiler stage to run one or more Compiler stage.

  You can click a Compiler stage to resume an interrupted compilation flow provided no compilation settings have changed from the initial start of the compilation flow.
- Click the **Report**, **RTL Viewer**, **Technology Map Viewer**, **Timing Analyzer**, or **Snapshot Viewer** icons for analysis of stage results.

As the Compiler progresses through the flow, the dashboard updates the status of each module, and enables icons that you can click for reports and analysis. The dashboard is also updated if you run your compilation flow from a command line with the `quartus_sh --flow` command.

5.1.2. Exploring Quartus Prime Project Contents

The Quartus Prime software organizes your design work within a project. You can create and compare multiple revisions of your project, to experiment with settings that achieve your design goals. When you create a new project in the GUI, the Quartus Prime software automatically creates an Quartus Prime Project File (`.qpf`) for that project. The `.qpf` references the Quartus Prime Settings File (`.qsf`). The `.qsf` lists the project's design, constraint, and IP files, and stores project-wide and entity-

specific settings that you specify in the GUI. You do not need to edit the text-based `.qpf` or `.qsf` files directly. The Quartus Prime software creates and updates these files automatically as you make changes in the GUI.

Table 32. Quartus Prime Project Files

File Type	Contains	To Edit	Format
Project file	Project and revision name	**File ➤ New Project Wizard**	Quartus Prime Project File (`.qpf`)
Settings file	Lists design files, entity settings, target device, synthesis directives, placement constraints	**Assignments ➤ Settings**	Quartus Prime Settings File (`.qsf`)
Quartus database	Project compilation results	**Project ➤ Export Design**	Quartus Database File (`.qdb`)
Partition database	Partition compilation results	**Project ➤ Export Design Partition**	Partition Database File (`.qdb`)
Timing constraints	Clock properties, exceptions, setup/hold	**Tools ➤ Timing Analyzer**	Synopsys Design Constraints File (`.sdc`)
Logic design files	RTL and other design source files	**File ➤ New**	All supported HDL files
Programming files	Device programming image and information	**Tools ➤ Programmer**	SRAM Object File (`.sof`) Programmer Object File (`.pof`)
IP core files	IP core variation parameterization	**Tools ➤ IP Catalog**	Quartus Prime IP File (`.ip`)
Platform Designer system files	System definition	**Tools ➤ Platform Designer**	Platform Designer System File (`.qsys`)
EDA tool files	Scripts for third-party EDA tools	**Assignments ➤ Settings ➤ EDA Tool Settings**	Verilog Output File (`.vo`) VHDL Output File (`.vho`) Verilog Quartus Mapping File (`.vqm`)
Archive files	Complete project as single compressed file	**Project ➤ Archive Project**	Quartus Prime Archive File (`.qar`)

5.1.2.1. Project File Best Practices

The Quartus Prime software provides various options for specifying project settings and constraints. The following best practices help ensure automated management and portability of your project files.

- Avoid manually editing Quartus Prime data files, such as the Quartus Prime Project File (`.qpf`), Quartus Prime Settings File (`.qsf`), Quartus IP File (`.ip`), or Platform Designer System File (`.qsys`). Syntax errors in these files cause errors during compilation. For example, the software may ignore improperly formatted settings and assignments.
- Do not compile multiple projects into the same directory. Instead, use a separate directory for each project.
- By default, the Quartus Prime software saves all project output files, such as Text-Format Report Files (`.rpt`), in the project directory. If you want to change the location of output files, instead of manually moving project output files, click **Assignments ➤ Settings ➤ Compilation Process Settings**, and specify the **Save project output files in specified directory** option.

5.1.3. Viewing Design Hierarchy and Adding Missing Source Files

With the introduction of the fast hierarchy display feature, you can now view the design hierarchy quickly and add the missing source files without having to wait until the completion of the Analysis & Elaboration compilation process.

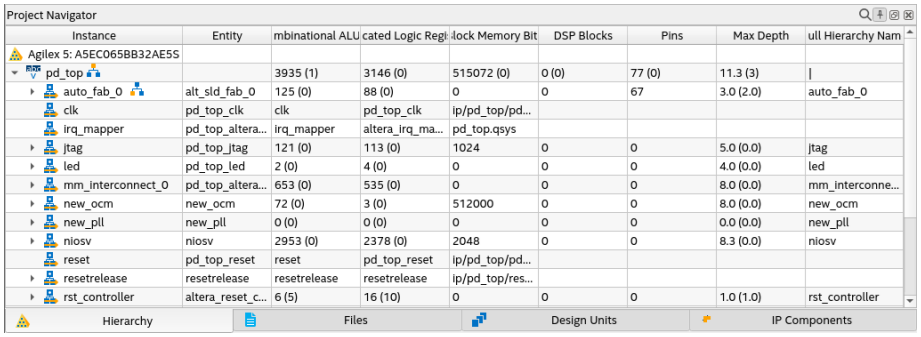
Perform these steps to view the design hierarchy and add the missing source files:

Prerequisite: After launching the Quartus Prime Pro Edition software GUI and creating your project, ensure you have added all RTL source files to the project and top-level entity name of the design, without which, you cannot proceed with Analysis & Elaboration.

1. Run either **Analysis & Synthesis** or **Analysis & Elaboration** on the compilation dashboard.
2. Navigate to the left-hand **Project Navigator ➤ Hierarchy** tab.

   The **Hierarchy** viewer allows you to quickly view your design hierarchy, and any errors or warnings before the design is fully elaborated. You can also cross-probe between the source files and information, warnings or error messages. All entities with missing source files are highlighted enabling you to make quick design fixes.

Figure 48. Fast Display of Design Hierarchy

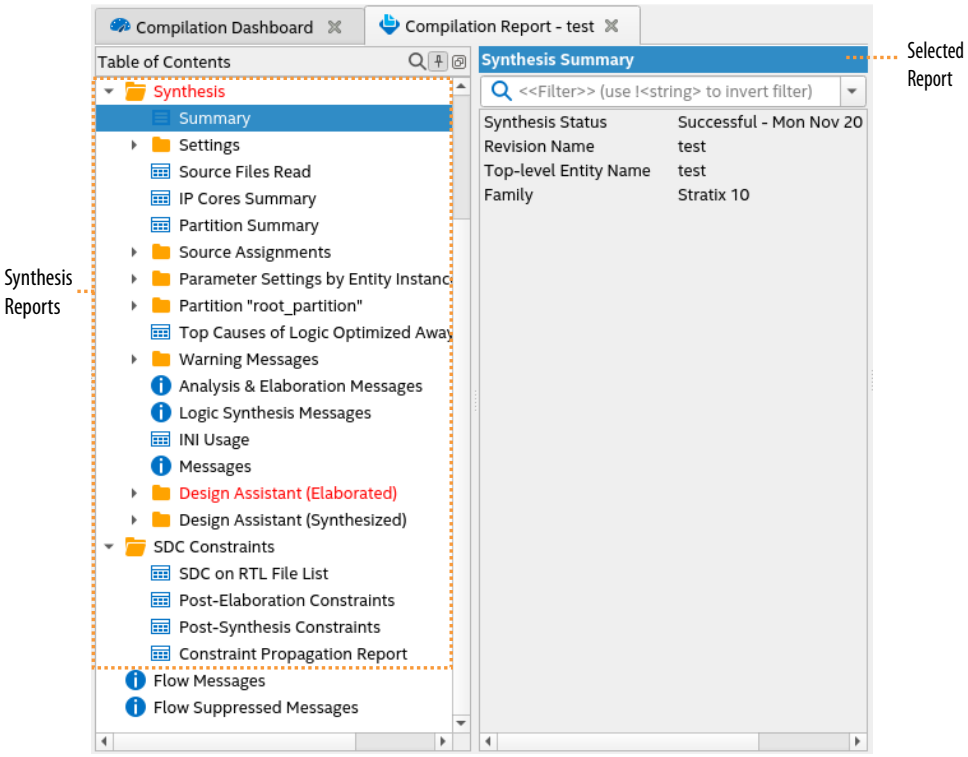


5.1.4. Viewing Project Reports

The Compilation Report panel updates dynamically to display detailed reports during project processing. To access Compilation Reports, click (**Processing ➤ Compilation Report**).

Review the detailed information in these the compilation reports to determine correct implementation. Right-click report data to locate and edit the source in project files.

Figure 49. Compilation Report

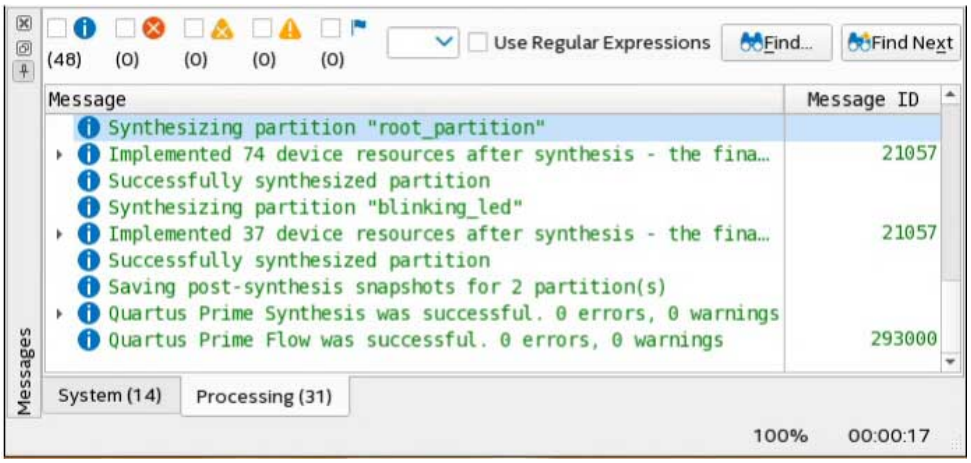


5.1.5. Viewing Project Messages

The Messages window (**View ➤ Messages**) displays information, warning, and error messages about Quartus Prime processes. Right-click messages to locate the source or get message help.

- **Processing** tab—displays messages from the most recent process
- **System** tab—displays messages unrelated to design processing
- **Find**—locates specific messages

Figure 50. Messages Window



5.1.5.1. Viewing Synthesis Warning Messages

Warning messages may contain hierarchies. In **Compilation Report ➤ Synthesis ➤ Messages** window, you can view hierarchical warning messages up to any level including the parent and child messages. For each message, you can view its source, file location, line number, and message ID by selecting appropriate column under **Message Column** (right-click on a message in the **Message** panel as shown in the following image and click **Message Column**).

Figure 51. Synthesis Warning Messages (Two levels)

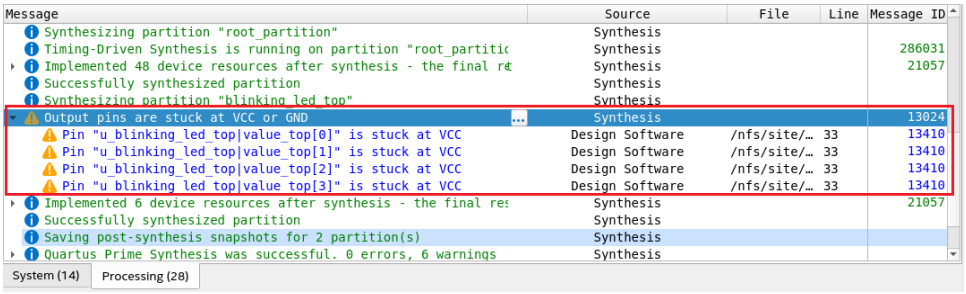
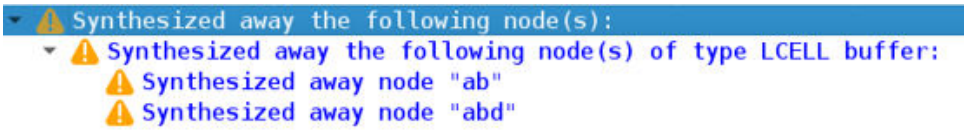


Figure 52. Example of Synthesis Warning Messages With Three Levels



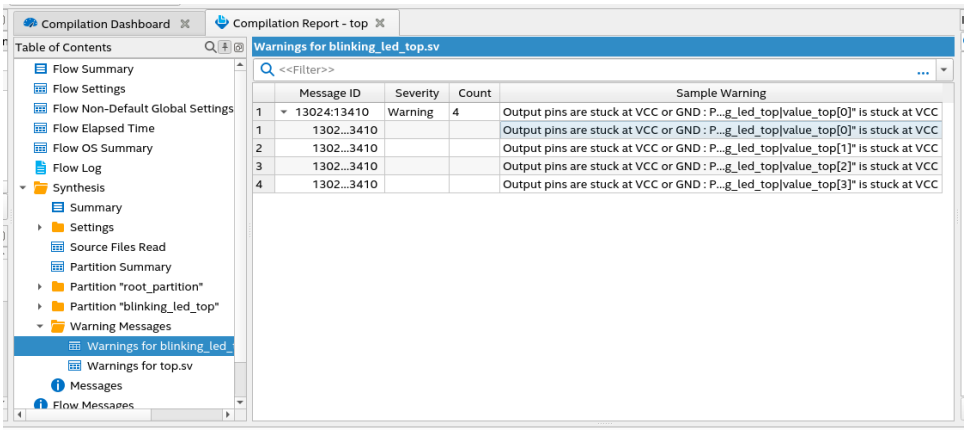
In **Compilation Report ➤ Synthesis ➤ Warning Messages**, you can view a comprehensive list of synthesis warning messages for each source file included in your design. You can view all child warning messages hidden within a parent warning message by expanding the collapsible rows. To view the location of each warning, perform these steps:

1. Right-click on the message.
2. Select the **Locate Node** option.
3. Select the desired tool to view the node.

Note: In the source file-specific warning messages window, messages are hierarchical in nature and display up to three levels. If the warning messages go deeper than three levels, use the **Message** (**View ➤ Messages**) window to view them.

In the source file-specific warning messages window, hierarchical messages are displayed with message IDs and sample warning messages that are a combination of the parent and child messages.

Figure 53. Synthesis Warning Messages for Each Source File



5.1.5.2. Suppressing Message Display

You can suppress display of unimportant messages from the Messages window, so that you can focus on the messages that are important to you. To suppress one or more messages from displaying in the Messages window, right-click the message, and then click any of the following commands:

- **Suppress Message**—suppresses all messages that match the exact text you specify.
- **Suppress Messages with Matching ID**—suppresses all messages that match the message ID number you specify, ignoring variables.
- **Suppress Messages with Matching Keyword**—suppresses all messages that match the keyword or hierarchy you specify.
- **Message Suppression Manager**—allows you to create and edit message suppression rules. You can define message suppression rules by message text, message ID number, or keyword.

Note:

- You cannot suppress error or Intel legal agreement messages.
- Suppressing a message also suppresses any submessages.
- A root message does not display if you suppress all of the root message's submessages.
- Message suppression is project revision-specific. Derivative project revisions inherit any suppression.
- You cannot edit messages or suppression rules during compilation.
- Messages are written to `stdout` when you use command-line executables.

Figure 54. Message Suppression Manager

Suppressing Messages by Design Entity

You can optionally suppress messages by design entity without modifying HDL. Entity-based message suppression can be helpful to eliminate insignificant warnings for specific IP components or design entities that may be obscuring other more important warnings.

To suppress messages by design entity, add the following line to the project `.qsf`, or to the `.qip` file for stand-alone IP components:

```
set_global_assignment -name MESSAGE_DISABLE -entity <name>
```

5.1.5.3. Promoting Critical Warnings to Errors

You can promote critical warnings to errors so that the compilation flow halts on receiving the critical warnings as it does with an errors. All critical warnings are supported.

You can only promote the message IDs on open projects.

1. In the **Message** dialog box, right-click on the critical warning you want to promote to an error.
2. Click **Message Promotion ➤ Promote Critical Message ID to Error**
   The software now treats the critical warning as an error.
3. To clear all promotions, click **Message Promotion ➤ Clear All Message Promotions**
4. Alternatively, manually promote or demote a critical warning in the `.qsf`. For example:

   ```
   set_global_assignment -name PROMOTE_WARNING_TO_ERROR 12677
   ```

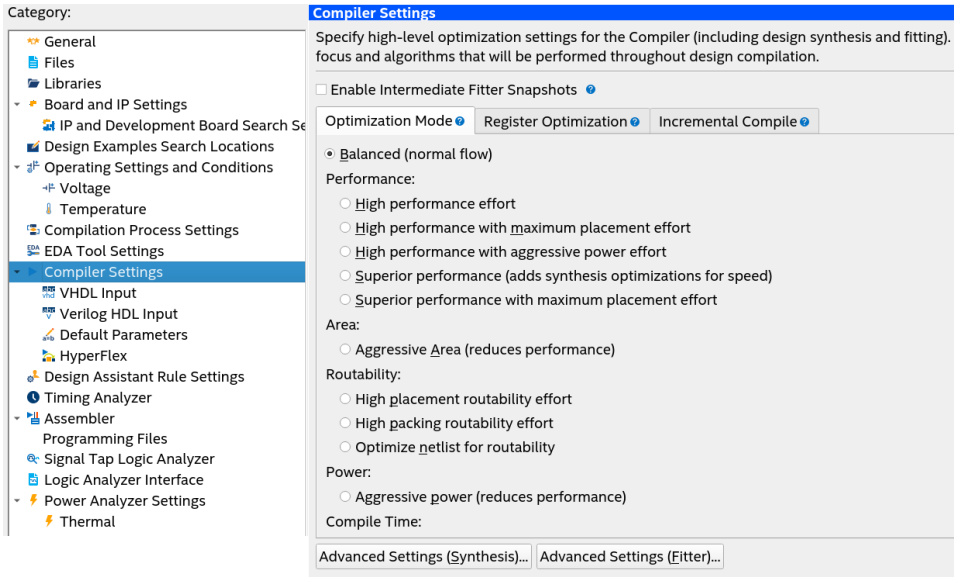
5.2. Managing Project Settings

The New Project Wizard guides you to make initial project settings when you setup a new project. You can modify these and other global project settings in the **Settings** and **Device** dialog boxes, respectively. The `.qsf` stores the settings for each project revision. The optimization of these project settings helps the Compiler to generate programming files that meet or exceed your specifications.

Global Project Settings

To access global project settings, click **Assignments ➤ Settings**, or click **Settings** on the **Tasks** pane.

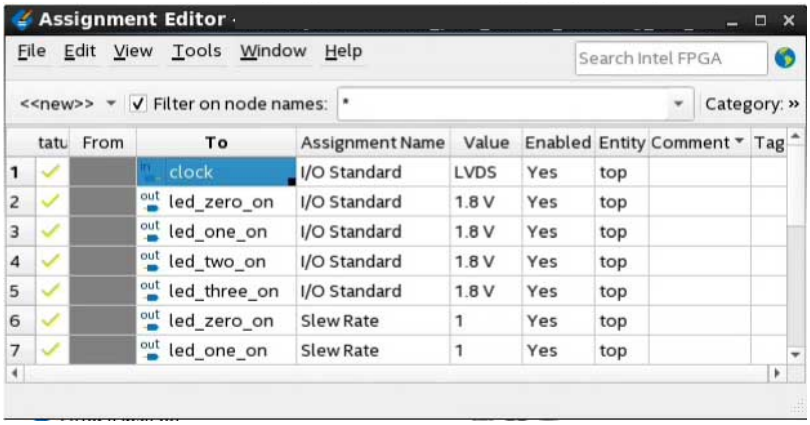
Figure 55. Settings Dialog Box for Global Project Settings



The **Settings** dialog box provides access to settings that control project design files, synthesis, Fitter, and timing constraints, operating conditions, EDA tool file generation, programming file generation, and other project-level settings.

Additionally, the Assignment Editor (**Assignments ➤ Assignment Editor**) provides a spreadsheet-like interface for specifying instance-specific settings and constraints.

Figure 56. Assignment Editor

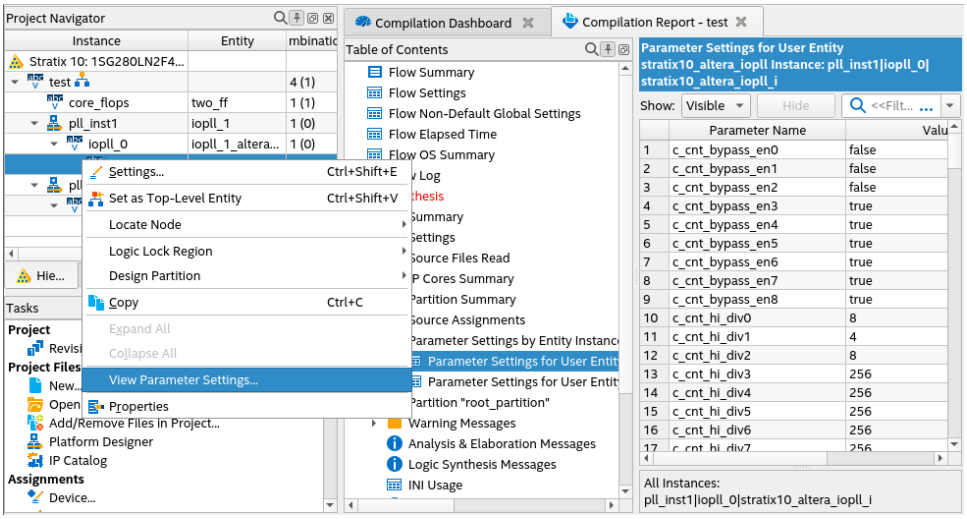


5.3. Viewing Parameter Settings From the Project Navigator

Starting from the Quartus Prime Pro Edition software version 23.3, you can view the parameter settings for a module directly from the **Project Navigator**.

To access the settings, locate the module, right-click and select **View Parameter Settings** in the context-sensitive menu. Compilation Report appears displaying the parameter settings for the entity, as shown in the following image:

Figure 57. Viewing Parameter Settings



5.4. Managing Logic Design Files

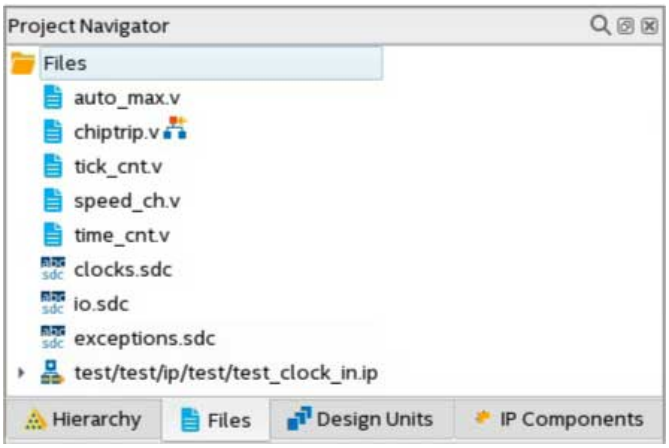
The Quartus Prime software helps you create and manage the logic design files in your project. Logic design files contain the logic that implements your design. When you add a logic design file to the project, the Compiler automatically includes that file in the next compilation. The Compiler synthesizes your logic design files to generate programming files for your target device.

The Quartus Prime software includes full-featured schematic and text editors, as well as HDL templates to accelerate your design work. The Quartus Prime software supports VHDL Design Files (`.vhd`), Verilog HDL Design Files (`.v`), and SystemVerilog (`.sv`). In addition, you can combine your logic design files with Intel and third-party IP core design files, including combining components into a Platform Designer system (`.qsys`).

Caution: Starting from the Quartus Prime Pro Edition software version 23.3, the compiler cannot synthesize schematic Block Design File (`.bdf`). For more information, refer to Converting Symbolic BDF Files to Acceptable File Formats on page 45.

The New Project Wizard prompts you to identify logic design files. Add or remove project files by clicking **Project > Add/Remove Files in Project.** View the project's logic design files in the Project Navigator.

Figure 58. Design and IP Files in Project Navigator



Right-click files in the Project Navigator to:

- **Open** and edit the file
- **Remove File from Project**
- **Set as Top-Level Entity** for the project revision
- **Create a Symbol File for Current File** for display in schematic editors
- Edit file **Properties**

5.4.1. Including Design Libraries

Include design files libraries in your project. Specify libraries for a single project, or for all Quartus Prime projects. The `.qsf` stores project library information.

The `quartus2.ini` file stores global library information.

1. Click **Assignment > Settings**.
2. Click **Libraries** and specify the **Project Library name** or **Global Library name**. Alternatively, you can specify project libraries with SEARCH_PATH in the `.qsf`, and global libraries in the `quartus2.ini` file.

Related Information

Design Library Migration Guidelines on page 47

5.4.2. Creating a Project Copy

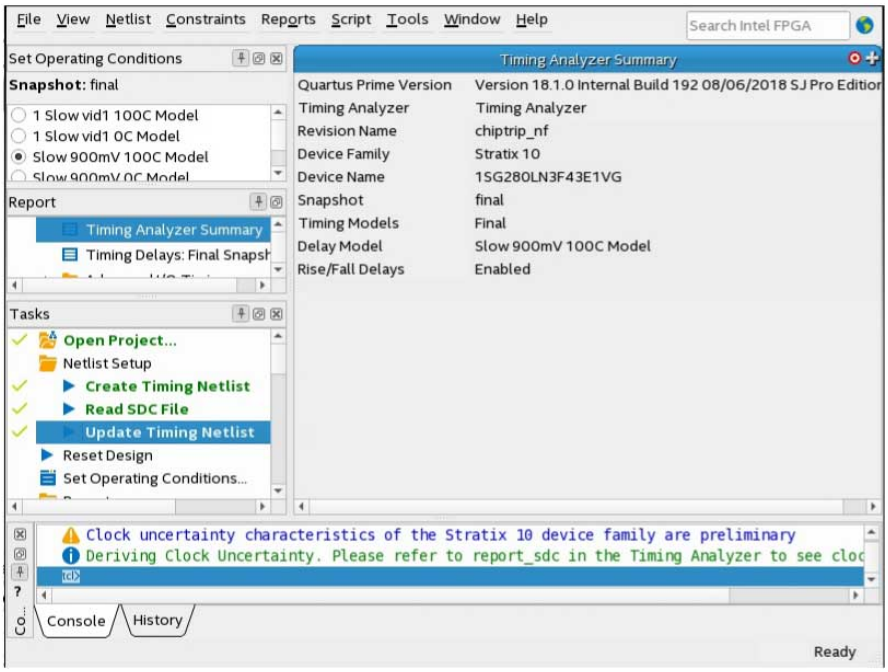
Click **Project > Copy Project** to create a separate copy of your project, rather than just a revision within the same project.

The project copy includes separate copies of all design files, any `.qsf` files, and project revisions. You can use this technique to optimize project copies for different applications that require design file differences. For example, you can optimize one project to interface with a 32-bit data bus, and optimize a project copy to interface with a 64-bit data bus.

5.5. Managing Timing Constraints

Apply appropriate timing constraints to correctly optimize fitting and analyze timing for your design. The Fitter optimizes the placement of logic in the device to meet your specified timing and routing constraints.

Figure 59. Timing Analyzer



Specify timing constraints in the Timing Analyzer (**Tools > Timing Analyzer**), or in an `.sdc` file. Specify constraints for clock characteristics, timing exceptions, and external signal setup and hold times before running analysis. The Timing Analyzer reports detailed information about the performance of your design compared with constraints in the Compilation Report panel.

Save the constraints you specify in the GUI in an industry-standard Synopsys Design Constraints File (`.sdc`). You can subsequently edit the text-based `.sdc` file directly. If you refer to multiple `.sdc` files in a parent `.sdc` file, the Timing Analyzer reads the `.sdc` files in the order you list.

5.6. Integrating Other EDA Tools

You can optionally integrate supported EDA synthesis, netlist partitioning, simulation, and signal integrity verification tools into the Quartus Prime design flow.

The Quartus Prime software supports input netlist files from supported EDA synthesis tools. The Compiler's EDA Netlist Writer module (`quartus_eda`) can automatically generate output files for processing in other EDA tools. The EDA Netlist Writer runs optionally as part of a full compilation, or you can run EDA Netlist Writer separately from the GUI or at the command line. The following functions are available to simplify EDA tool integration:

Table 33. EDA Tool Integration Functions

EDA Integration Task	EDA Integration Function
Specify settings for generation of output files for processing in other EDA tools.	Click **Assignments ➤ Settings ➤ EDA Tool Settings** to specify options for supported tools.
Generate output files for processing in other EDA tools.	Click **Processing ➤ Start ➤ Start EDA Netlist Writer** (or run `quartus_eda`) to generate files.
Compile RTL and gate-level simulation model libraries for your device, supported EDA simulators, and design language.	Click **Tools ➤ Launch Simulation Library Compiler** to compile simulation libraries easily.
Generate EDA tool-specific setup scripts to compile, elaborate, and simulate Intel FPGA IP models and simulation model library files.	Specify options for **Simulation** file output when generating Intel FPGA IP with IP parameter editor.
Generate files that allow supported EDA tools to perform netlist modifications, such as adding new modules, partitioning the netlist, and changing module connectivity.	Use the `quartus_eda –resynthesis` command to generate a Verilog Quartus Mapping File (`.vqm`) that contains a node-level (or atom) representation of the netlist in standard structural Verilog RTL.
Include files generated by other EDA design entry or synthesis tools in your project as synthesized design files.	Click **Project ➤ Add/Remove Files In Project** to add supported **Design File** files from other EDA tools.

5.7. Exporting Compilation Results

The Quartus Prime Compiler writes the results to a set of database files. You can run a command to export the compilation results database as a single Quartus Database File (`.qdb`).

After running design compilation, the exported `.qdb` file contains the data to reproduce similar compilation results in another project, or in a later software version. You can export your project's compilation results database for import to another project or migration to a later Quartus Prime software version.

You can export the `.qdb` for your entire project or for a design partition that you define in your project. When migrating the database for an entire project, you can export the compilation database in a *version-compatible* format to ensure

compatibility for import to a later software version. Although you cannot directly read the contents of the .qdb file after export, you can view attributes of the database file in the Quartus Database File Viewer.

Table 34. Exporting Compilation Results

To Export Compilation Results For	Method	Description
Complete Design	Click **Project ➤ Export Design**	Saves compilation results for the current project revision in a version-compatible Quartus database file (.qdb) that you can import to another project or migrate to a later version of the Quartus Prime software. You can export the results for the synthesized or final compilation snapshot. *Note:* Not supported for Agilex 7 devices.
Design Partition	Click **Project ➤ Export Design Partition**	Saves compilation results for a design partition as a Partition Database File (.qdb) that you can import to another project using the same version of the Quartus Prime software. You can export the results for the synthesized or final compilation snapshot.

Related Information

Creating Database-Only Archives on page 112

5.7.1. Exporting a Version-Compatible Compilation Database

You can export a project compilation database to a format that ensures version-compatibility with a later version of the Quartus Prime software. The Quartus Prime Pro Edition software version supports export of version-compatible databases for the following software versions and devices:

Table 35. Version-Compatible Compilation Database Support

The first table column indicates the first version to support version-compatible compilation database export for the specified devices.

Note:

- Database import supports two major versions back. For example, a database that you export from version 19.3, you can then import using version 19.3, 20.1, and 20.3. However, you cannot import version 19.3 to 21.1.
- You can export from any version that follows a supported version, if the version still supports the devices.

First Version with 'Export Design' Support	Stratix 10 and Devices	Arria 10 and Cyclone 10 GX Devices
18.0	No Support.	Supports all devices.
18.1	• 1SG250L • 1SG280H_S2 • 1SG280L • 1SG280L_S3 • 1SX250L • 1SX280L • 1SX280L_S3	Supports all devices.
19.1	• 1SM16BH • 1SM21BH • 1SM16CH	Supports all devices.

continued...

First Version with 'Export Design' Support	Stratix 10 and Devices	Arria 10 and Cyclone 10 GX Devices
	• 1SM21CH • 1SM21KH • 1SM16KH • 1SM21LH • 1SM16LH	
19.3	• 1SG10MH_U1 • 1SG10MH_U2 • 1ST250E • 1ST280E • 1SM16E • 1SM21E • 1ST165E • 1ST210E • 1SG166H • 1SG211H	Supports all devices.
20.1	• 1SD280P • 1ST040E • 1ST085E • 1ST110E	Supports all devices.
20.3	• 1SD21BP • 1SG040H • 1SX040H	Supports all devices.
20.4	• 1SN21BH • 1SN21CE	Supports all devices.

1. In the Quartus Prime software, open the project that you want to export.
2. Generate synthesis or final compilation results by running one of the following commands:
   - Click **Processing ➤ Start ➤ Start Analysis & Synthesis** to generate synthesized compilation results.
   - Click **Processing ➤ Start Compilation** to generate final compilation results.
3. Click **Project ➤ Export Design**. Select the **synthesized** or **final Snapshot**.

Figure 60. Export Design Dialog Box

4. Specify a name for the **Quartus Database File** to contain the exported results, and click **OK**.
5. To include the exported design's settings and constraint files, copy the `.qsf` and `.sdc` files to the import project directory.

5.7.2. Importing a Version-Compatible Compilation Database

Follow these steps to import a project compilation database into a newer version of the Quartus Prime software:

Note: Designs exported from the Quartus Prime Pro Edition software versions 23.2 or earlier cannot be imported into version 23.3 due to the new DNI database.

1. Export a version-compatible compilation database for a complete design, as Exporting a Version-Compatible Compilation Database on page 100 describes.
2. In a newer version of the Quartus Prime software, open the original project. Click **Yes** if prompted to open a project created with a different software version.
3. Click **Project ➤ Import Design** and specify the **Quartus Database File**. To remove previous results, turn on **Overwrite existing project's databases**

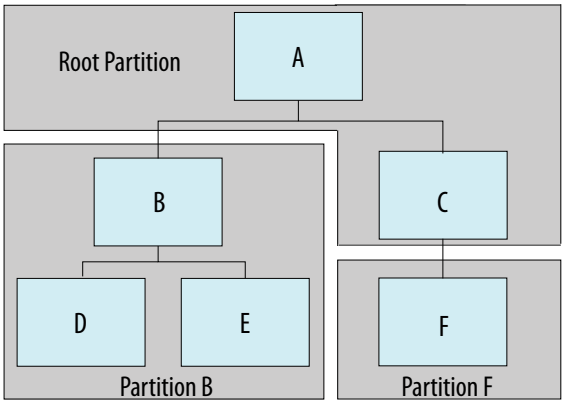
Figure 61. Import Design Dialog Box

4. Click **OK**. When you compile the imported design, run only Compiler stages that occur after the stage the `.qdb` preserves, rather than running a full compilation. For example, if you import a version-compatible database that contains the synthesized snapshot, start compilation with the Fitter (**Processing ➤ Start ➤ Start Fitter**). If you import a version-compatible database that contains the final snapshot, start compilation with Timing Analysis (Signoff) (**Processing ➤ Start ➤ Start Timing Analysis (Signoff)**).

5.7.3. Creating a Design Partition

A design partition is a logical, named, hierarchical boundary that you can assign to an instance in your design. Defining a design partition allows you to optimize and lock down the compilation results for individual blocks. You can then optionally export the compilation results of a design partition for reuse in another context, such as reuse in another project.

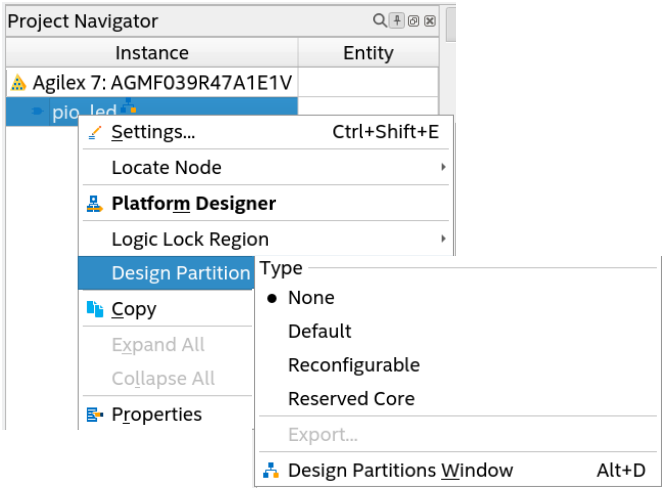
Figure 62. Design Partitions in Design Hierarchy



Follow these steps to create and modify design partitions:

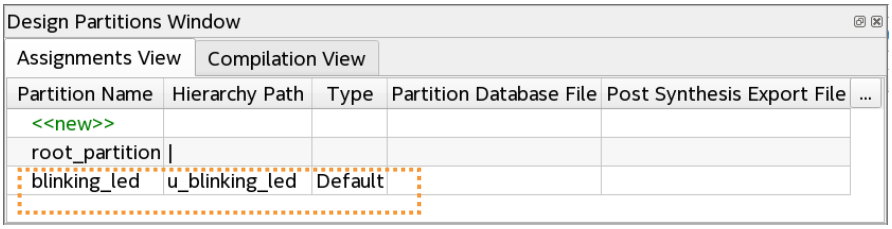
1. In the Quartus Prime software, open the project that you want to partition.
2. Generate synthesis or final compilation results by running one of the following commands:
   - Click **Processing ➤ Start ➤ Start Analysis & Synthesis** to generate synthesized compilation results.
   - Click **Processing ➤ Start Compilation** to generate final compilation results.
3. In the Project Navigator, right-click an instance in the **Hierarchy** tab, click **Design Partition ➤ Set as Design Partition**.

Figure 63. Creating a Design Partition from the Project Hierarchy



4. To view and edit all design partitions in the project, click **Assignments ➤ Design Partitions Window**.

Figure 64. Design Partitions Window



5. Specify the properties of the design partition in the Design Partitions Window. The following settings are available:

Table 36. Design Partition Settings

Option	Description
Partition Name	Specifies the partition name. Each partition name must be unique and consist of only alphanumeric characters. The Quartus Prime software automatically creates a top-level (|) "root_partition" for each project revision.
Hierarchy Path	Specifies the hierarchy path of the entity instance that you assign to the partition. You specify this value in the **Create New Partition** dialog box. The root partition hierarchy path is |.
Type	Double-click to specify one of the following partition types that control how the Compiler processes and implements the partition: • **Default**—Identifies a standard partition. The Compiler processes the partition using the associated design source files. • **Reconfigurable**—Identifies a reconfigurable partition in a partial reconfiguration flow. Specify the **Reconfigurable** type to preserve synthesis results, while allowing refit of the partition in the PR flow. • **Reserved Core**—Identifies a partition in a block-based design flow that is reserved for core development by a Consumer reusing the device periphery.
Empty	Specifies an empty partition that the Compiler skips. This setting is incompatible with the **Reserved Core** and **Partition Database File** settings for the same partition.
Partition Database File	Specifies a Partition Database File (.qdb) that the Compiler uses during compilation of the partition. You export the .qdb for the stage of compilation that you want to reuse (synthesized or final). Assign the .qdb to a partition to reuse those results in another context.
Entity Re-binding	• PR Flow—specifies the entity that replaces the default persona in each implementation revision. • Root Partition Reuse Flow —specifies the entity that replaces the reserved core logic in the consumer project.
Color	Specifies the color-coding of the partition in the Chip Planner and Design Partition Planner displays.
Post Synthesis Export File	Automatically exports post-synthesis compilation results for the partition to the specified .qdb file each time Analysis & Synthesis runs. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.
Post Final Export File	Automatically exports post-final compilation results for the partition to the specified .qdb file each time the final stage of the Fitter runs. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.

5.7.4. Exporting a Design Partition

The following steps describe export of design partitions that you create in your project.

Note: Design partitions exported from the Quartus Prime Pro Edition software versions 23.2 or earlier cannot be imported into version 23.3 or later due to the new DNI database.

When you compile a design containing design partitions, the Compiler can preserve a synthesis or final snapshot of results for each partition. You can export the synthesized or final compilation results for individual design partitions with the **Export Design Partition** dialog box.

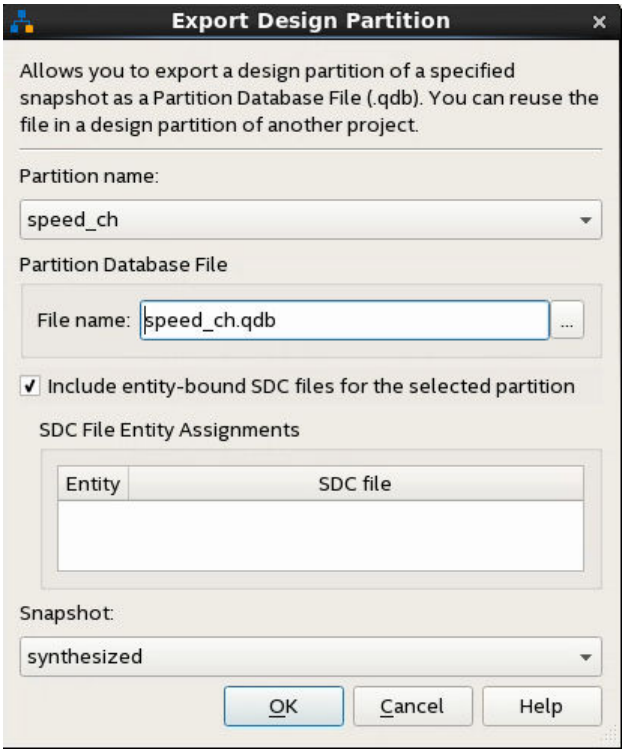
If the partition includes any entity-bound `.sdc` files, you can include those constraints in the `.qdb`. In addition, you can automate export of one or more partitions in the Design Partitions Window.

Manual Design Partition Export

Follow these steps to manually export a design partition with the **Export Design Partition** dialog box:

1. Open a project and create one or more design partitions. Creating a Design Partition on page 102 describes this process.
2. Run synthesis (**Processing ➤ Start ➤ Start Analysis & Synthesis**) or full compilation (**Processing ➤ Start Compilation**), depending on which compilation results that you want to export.
3. Click **Project ➤ Export Design Partition**, and specify one or more options in the **Export Design Partition** dialog box:

Figure 65. Export Design Partition Dialog Box



- Select the **Partition name** and the compilation **Snapshot** for export.
- To include any entity-bound `.sdc` files in the exported `.qdb`, turn on **Include entity-bound SDC files for the selected partition**.

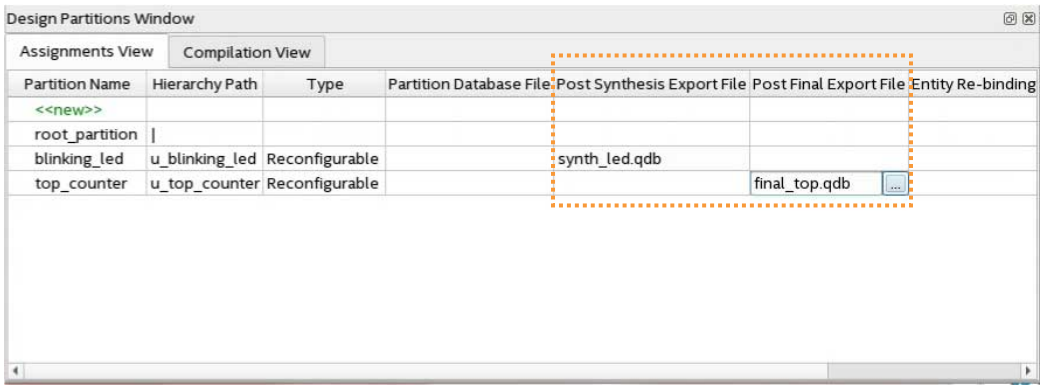
4. Click **OK**. The compilation results for the design partition exports to the file that you specify.

Automated Design Partition Export

Follow these steps to automatically export one or more design partitions following each compilation:

1. Open a project containing one or more design partitions. Creating a Design Partition on page 102 describes this process.
2. To open the Design Partitions Window, click **Assignments ➤ Design Partitions Window**.
3. To automatically export a partition with synthesis results after each time you run synthesis, specify the a `.qdb` export path and file name for the **Post Synthesis Export File** option for that partition. If you specify only a file name without a path, the file exports to the `output_files` directory after compilation.
4. To automatically export a partition with final snapshot results each time you run the Fitter, specify a `.qdb` file name for the **Post Final Export File** option for that partition. If you specify only a file name without a path, the file exports to the `output_files` directory after compilation.

Figure 66. Specifying Export File in Design Partitions Window



`.qsf` Equivalent Assignment:

```
set_instance_assignment -name EXPORT_PARTITION_SNAPSHOT_<FINAL|SYNTHESIZED> \
     <hpath> -to <file_name>.qdb
```

Related Information

- Intel Quartus Prime Pro Edition User Guide: Block Based Design
- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration

5.7.5. Reusing a Design Partition

You can reuse the compilation results of a design partition exported from another Quartus Prime project. Reuse of a design partition allows you to share a synthesized or final design block with another designer. Refer to *Intel Quartus Prime Pro Edition User Guide: Block-Based Design* for more information about reuse of design partitions.

To reuse an exported design partition in another project, you assign the exported partition `.qdb` to an appropriately configured design partition in the target project via the Design Partition Window:

1. Export a design partition with the appropriate snapshot, as Exporting a Design Partition on page 104 describes.
2. Open the target Quartus Prime project that you want to reuse the exported partition.
3. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
4. Click **Assignments ➤ Design Partitions Window**, and then create a design partition to contain the logic and compilation results of the exported `.qdb`.
5. Click the **Partition Database File** option for the new partition and select the exported `.qdb` file.

Figure 67. Partition Database File Setting in Design Partitions Window

6. Specify any other properties of the design partition in the Design Partitions Window. The Compiler uses the partition's assigned `.qdb` as the source.

5.7.6. Viewing Quartus Database File Information

Although you cannot directly read a `.qdb` file, you can view helpful attributes about the file to quickly identify its contents and suitability for use.

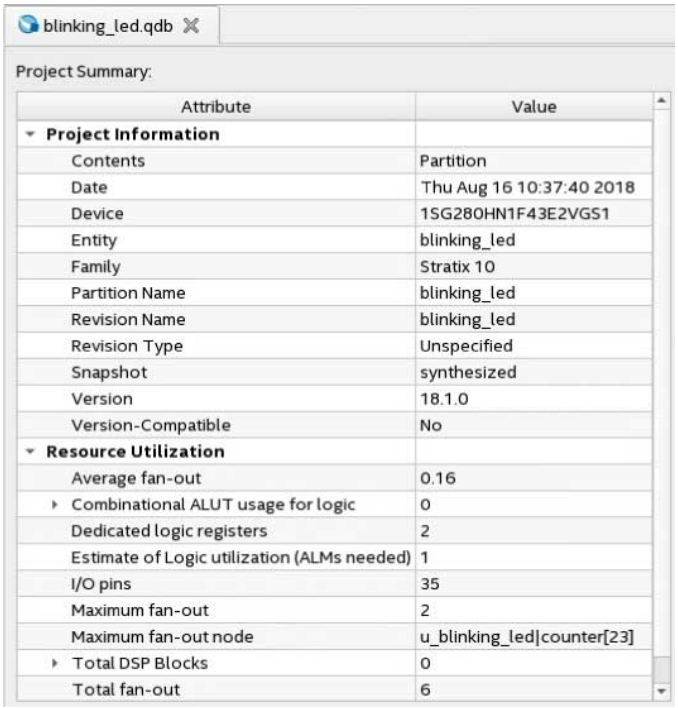
The Quartus Prime software automatically stores metadata about the project of origin when you export a Quartus Database File (`.qdb`). You can then use the Quartus Database File Viewer to display the attributes of any of these `.qdb` files.
Follow these steps to view the attributes of a `.qdb` file:

1. In the Quartus Prime software, click **File ➤ Open**, select **Design Files** for **Files of Type**, and select a `.qdb` file.
2. Click **Open**. The Quartus Database File Viewer displays project and resource utilization attributes of the `.qdb`.

   Alternatively, run the following command-line equivalent:

```
quartus_cdb --extract_metadata --file <archive_name.qdb> \
     --type quartus --dir <extraction_directory> \
     [--overwrite]
```

Figure 68. Quartus Database File Viewer



5.7.6.1. QDB File Attribute Types

The Quartus Database Viewer can display the following attributes of a `.qdb` file:

Table 37. QDB File Attributes

QDB Attribute Types	Attribute	Example
Project Information	Contents	`Partition`
	Date	`Thu Jan 23 10:56:23 2018`
	Device	`10AX016C3U19E2LG`
	Entity (if Partition)	`Counter`
	Family	`Arria 10`
	Partition Name	`root_partition`
	Revision Name	`Top`
	Revision Type	`PR_BASE`
	Snapshot	`synthesized`
	Version	`18.1.0 Pro Edition`
	Version-Compatible	`Yes`
Resource Utilization (exported for partition QDB only)	For synthesized snapshot partition lists data from the **Synthesis Resource Usage Summary** report.	`Average fan-out.16` `Dedicated logic registers:14` `Estimate of Logic utilization:1`

continued...

		I/O pins:35 Maximum fan-out:2 Maximum fan-out node:counter[23] Total DSP Blocks:0 Total fan-out:6 ...
	For the final snapshot partition, lists data from the **Fitter Partition Statistics** report.	Average fan-out:.16 Combinational ALUTs: 16 I/O Registers M20Ks ...

5.7.7. Clearing Compilation Results

You can clean the project database if you want to remove prior compilation results for all project revisions or for specific revisions. For example, you must clear previous compilation results before importing a version-compatible database to an existing project.

1. Click **Project > Clean Project**.
2. Select **All revisions** to clear the databases for all revisions of the current project, or specify a **Revision name** to clear only the revision's database you specify.
3. Click **OK**. A message indicates when the database is clean.

Figure 69. Clean Project Dialog Box Cleans the Project Database

5.8. Archiving Projects

You can optionally save the elements of a project in a single, compressed Quartus Prime Archive File (`.qar`) by clicking **Project > Archive Project**. The `.qar` preserves RTL design, project, and settings files required to restore the project.

Use this technique to share projects between designers, or to transfer your project to a new version of the Quartus Prime software, or to Intel support. Optionally add compilation results, Platform Designer system files, and third-party EDA tool files to the archive.

Related Information

Project Archive Commands on page 114

5.8.1. Manually Adding Files To Archives

Follow these steps to add files to a project archive manually:

Note: If preserving a custom component as part of an Quartus Prime Archive (`.qar`), you must first explicitly add the component `_hw.tcl` file to the project to ensure that the `.qar` includes the component. Click **Project ➤ Add/Remove Files in Project** to add files to your project.

1. Click **Project ➤ Archive Project** and specify the archive file name.
2. Click **Advanced**.
3. Select the **File set** for archive or select **Custom**. Turn on **File subsets** for the archive.
4. Click **Add** and select Platform Designer system or EDA tool files. Click **OK**.
5. Click **Archive**.

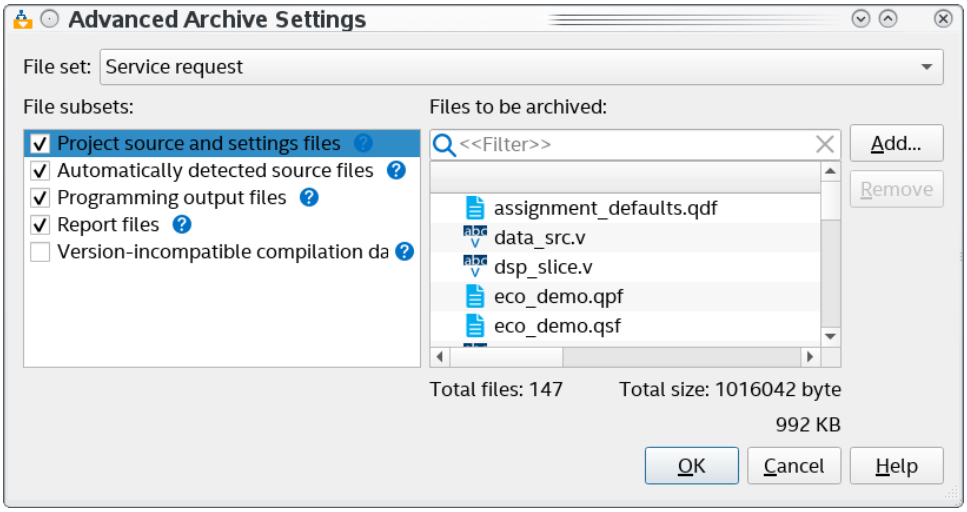
5.8.2. Archiving Projects for Service Requests

When archiving projects for a service request, include all needed file types for proper debugging by customer support.

To identify and include appropriate archive files for an Intel service request:

1. Click **Project > Archive Project** and specify the archive file name.
2. Click **Advanced**.
3. In **File set**, select **Service request** to include files for Intel Support.
   - Project source and setting files (`.v`, `.vhd`, `.vqm`, `.qsf`, `.sdc`, `.qip`, `.qpf`, `.cmp`)
   - Automatically detected source files (various)
   - Programming output files (`.jdi`, `.sof`, `.pof`)
   - Report files (`.rpt`, `.pin`, `.summary`, `.smsg`)
4. Click **OK**, and then click **Archive**.

Figure 70. Archiving Project for Service Request



5.8.3. Archiving Projects for External Revision Control

Your project may involve different team members with distributed responsibilities, such as sub-module design, device and system integration, simulation, and timing closure. In such cases, it may be useful to track and protect file revisions in an external revision control system.

While Quartus Prime project revisions preserve various project setting and constraint combinations, external revision control systems can also track and merge RTL source code, simulation testbenches, and build scripts. External revision control supports design file version experimentation through branching and merging different versions of source code from multiple designers. Refer to your external revision control documentation for setup information.

5.8.3.1. Project Files to Include In External Revision Control

When archiving Quartus Prime projects for external source control, The **Source control** setting in **Advanced Archive Settings** dialog box is preset to include all appropriate file types for source control automatically.

Figure 71. Advanced Archive Settings Dialog Box

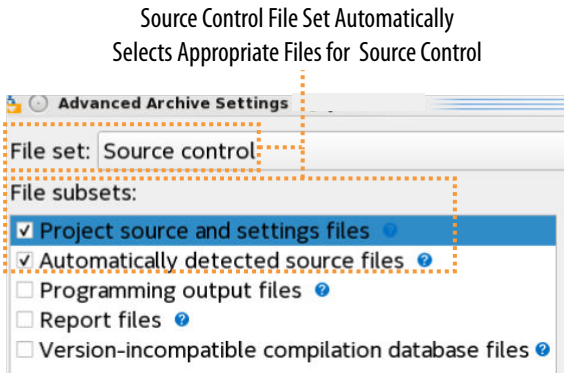


Table 38. Project Files to Include In External Revision Control

File Type	Description
Quartus Prime project setting and assignment files	• Quartus Prime Project Files (`.qpf`) • Quartus Prime Settings Files (`.qsf`) • Quartus Prime Pin Planner File (`.ppf`)
Timing constraint files	Synopsys Design Constraint Files (`.sdc`)
Design files	• Verilog HDL Design Files (`.v`) • SystemVerilog Design Files (`.sv`) • VHDL Design Files (`.vhd`) • Block Symbol Files (`.bsf`) • Verilog Quartus Mapping Files (`.vqm`) • Platform Designer System Files (`.qsys`) • State Machine Editor Files (`.smf`) • Tcl Script Design Files (`.tcl`)
System and IP files	• IP variation file (`.ip`) • Verilog IP design files (`.v`) • SystemVerilog IP design files (`.sv`) • VHDL IP design files (`.sv`) • VHDL Component Declaration Files (`.cmp`) • Quartus Prime IP file (`.qip`) • Quartus Prime Simulation IP File (`.sip`) • Platform Designer System Files (`.qsys`) • Platform Designer connection and parameterization files (`.sopcinfo`) • IP upgrade status files (`.csv`) • IP synthesis parameters files (`.qgsynthc`) • IP simulation parameters files (`.qgsimc`) • Platform Designer system exported as (`.tcl`).
EDA tool integration files	• Verilog HDL Output Files (`.vo`) • VHDL Output Files (`.vho`) • VHDL simulation model files (`.vhd`) • Verilog HDL simulation model files (`.v`) • Simulation library files (`cds.lib`, `hdl.var`) • Simulation setup scripts (`_setup.sh`, `.tcl`, `.spd`, `.txt`)

5.8.4. Creating Database-Only Archives

If your project contains sensitive RTL that you do not want to share with Intel support, you can create a *database-only* archive. A database-only archive contains only the minimum files required to run timing analysis, fitter, assembler, and GUI design-inspection tools like Chip Planner.

A database-only archive includes only the project Quartus databases and any additional files required for compilation. It does not include RTL files.

You can review the complete list of files included in the archive in a report generated when you create a database-only archive.

Security Note:

A database-only archive does not guarantee protection for sharing your design without sharing your RTL. The RTL Netlist Viewer, Technology Map Viewer, and other views, along with the EDA Netlist Writer, are still available for projects exported using this feature.

With some effort, the original content can be reverse engineered.

To disable these features, encrypt your design. For details, see Support for the IEEE 1735 Encryption Standard on page 83.

Before creating a database-only archive, your project must have completed one of the following compilation stages:

- Synthesis

  Archives that are created after running Synthesis can be used to run the fitter and then complete timing analysis, run the assembler, and use GUI-based design-inspection tools.

- Finalized

  Archives that are created after completing a full compilation flow for your project can be used to complete timing analysis, run the assembler, and use GUI design-inspection tools.

To create a database-only archive, run the following command:

```
quartus_sh --archive_database -project <project_file> [-use_final_db]
```

Specify the `-use_final_db` option to create a database-only archive based on the finalized snapshot of your project. Otherwise, a database-only archive based on the synthesized snapshot is created.

The command generates two files: a `.qar` file that contains the database-only archive, and a `.contents.txt` file that lists files that are included in the `.qar` file.

You can get command syntax details by running the following command:

```
quartus_sh --help=archive_database
```

Related Information

Support for the IEEE 1735 Encryption Standard on page 83

5.9. Command-Line Interface

You can optionally use command-line executables or scripts to run project commands, rather than using the GUI. This technique can be helpful if you have many settings and wish to track them in a single file or spreadsheet for iterative comparison. The `.qsf` supports only a limited subset of Tcl commands. Therefore, pass settings and constraints using a Tcl script:

1. Create a text file with the extension `.tcl` that contains your assignments in Tcl format.
2. Source the Tcl script file by adding the following line to the .qsf: `set_global_assignment -name SOURCE_TCL_SCR IPT_FILE <file name>`.

5.9.1. Project Revision Commands

create_revision Command

`create_revision` defines the properties of a new project revision.

```
create_revision <name> -based_on <revision_name> -set_current -new_rev_type \
     <rev_type> -root_partition_qdb_file <root qdb>
```

Table 39. create_revision Command Options

Option	Description
`based_on` (optional)	Specifies the revision name on which the new revision bases its settings.
`set_current` (optional)	Sets the new revision as the current revision.
`-new_rev_type`	Specifies a `base` or `impl` (implementation) type for a new revision.
`root_partition_qdb_file`	Specifies the name of a static region .qdb if already known when creating a revision.

get_project_revisions Command

`get_project_revisions` returns a list of all revisions in the project.

```
get_project_revisions <project_name>
```

delete_revision Command

`delete_revision` deletes the revision you specify from your project.

```
delete_revision <revision name>
```

set_current_revision Command

`set_current_revision` sets the revision you specify as the current revision.

```
set_current_revision -force <revision name>
```

Related Information

- Optimizing Settings with Project Revisions on page 121
- Optimize Settings with Design Space Explorer II
- Design Space Explorer II Tool
- Using Design Space Explorer II (DSE II) Video

5.9.2. Project Archive Commands

project_archive Command

`project_archive` archives your project into a single, compressed `.qar` file.

```
project_archive <name>.qar
```

Table 40. project_archive Command Options

Options	Description
`-all_revisions`	Includes all revisions of the current project in the archive.
`-common_directory /<name>`	Preserves original project directory structure in specified subdirectory.
`-include_libraries`	Includes libraries in archive.
`-include_outputs`	Includes output files in archive.
`-use_file_set <file_set>`	Includes specified fileset in archive.
`-version_compatible_database`	Includes version-compatible database files in archive.

restore_archive Command

Restores an archived project to a destination directory with optional overwriting of current contents.

```
project_restore <name>.qar -destination <directory name> -overwrite
```

Related Information

Archiving Projects on page 109

5.9.3. Project Database Commands

export_database Command

`export_design` exports the specified project database to the `.qdb` file you specify.

These commands require the `quartus_cdb` executable.

```
quartus_cdb <revision name> --export_design --file <file name>.qdb \
     --snapshot <synthesized/final>
```

import_database Command

`import_design` imports the specified project database to the `.qdb` file you specify.

```
quartus_cdb <revision name> --import_design --file <file name>.qdb
```

export_block Command

`export_block` exports the specified partition database to the `.qdb` file you specify.

```
quartus_cdb -r  <project name> -c <revision name> --export_block \
     <partition name> --snapshot <name> --file <file name>.qdb
```

5.9.3.1. quartus_cdb Executables to Manage Version-Compatible Databases

The command-line arguments to the `quartus_cdb` executable in the Quartus Prime Pro software are `export_design` and `import_design`. The exported version-compatible design files are archived in a file (with a `.qdb` extension). This differs from the Quartus Prime Standard Edition software, which writes all files to a directory.

In the Quartus Prime Standard Edition software, the flow exports both post-map and post-fit databases. In the Quartus Prime Pro Edition software, the export command requires the `snapshot` argument to indicate the target snapshot to export. If the specified snapshot has not been compiled, the flow exits with an error. In ACDS 16.0, export is limited to "synthesized" and "final" snapshots.

```
quartus_cdb <project_name> [-c <revision_name>] --export_design
--snapshot <snapshot_name> --file <filename>.qdb
```

The import command takes the exported `*.qdb` file and the project to which you want to import the design.

```
quartus_cdb <project_name> [-c <revision_name>] --import_design
--file <archive>.qdb [--overwrite] [--timing_analysis_mode]
```

The `--timing_analysis_mode` option is only available for Arria 10 designs. The option disables legality checks for certain configuration rules that may have changed from prior versions of the Quartus Prime software. Use this option only if you cannot successfully import your design without it. After you have imported a design in timing analysis mode, you cannot use it to generate programming files.

5.10. Related Trainings

You can take up the following training to help you understand how to manage your project:

- Instructor-Led Training: Using Quartus Prime Software

A. Next Steps After Getting Started

Navigating Content Through Tasks

Use the following table to navigate this guide through user-tasks:

A	B	C	D
Plan FPGA Design for RTL Flow or Work With Intel FPGA IP Cores	Create a New Project or Migrate Your Existing Project to Quartus Prime Pro Edition Software	Manage Your Quartus Prime Pro Edition Projects	Review Next Steps

Once you complete installing and licensing the required Intel FPGA development software and setting up your Quartus Prime Pro Edition project, refer to the following topics for additional resources and training to aid your design journey:

A.1. Additional Resources

Resource	Description
Intel Community for FPGA Intellectual Property	Allows you to post queries and get responses to Intel FPGA IP-related issues.
Intel Community for Quartus Prime Software	Allows you to post queries and get responses to Quartus Prime software-related issues.
Intel Community for Intel FPGA Software Installation and Licensing	Allows you to post queries and get responses to Intel FPGA software installation and licensing issues.
Intel FPGA Knowledge Base	Provides links to applicable articles that span a variety of Quartus Prime software-related issues.
Intel FPGA Self-Service Licensing Center	Provides support for licensing Intel FPGA software.
Quartus Prime Pro Edition User Guide: Design Recommendations	Describes best practices for designing FPGAs with the Quartus Prime Pro Edition software.
Quartus Prime Pro Edition User Guide: Design Compilation	Describes how to set up, run, and optimize for all stages of the Quartus Prime Pro Edition software compiler. The compiler synthesizes, places, and routes your design before generating a device programming file.
Quartus Prime Pro Edition User Guide: Scripting	Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

continued...

Resource	Description
Scripting with Quartus Prime Software	Demonstrates how to use the command-line executables for the Quartus Prime design flow.
Quartus Prime Pro Edition User Guide: Third-party Simulation	Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Mentor Graphics*, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
Questa*-Intel FPGA Edition Quick-Start: Quartus Prime Pro Edition User Guide	Demonstrates how to simulate a Quartus Prime Pro Edition design in the Questa*-Intel FPGA Edition simulator.
How to Setup RTL Simulations in Quartus Prime, Platform Designer, and Third-Party Simulators	Describes how to setup an RTL-based simulation using the IP setup simulation utility.
Quartus Prime Pro Edition Software User Guides Collection	Each user guide in the collection covers a specific topic and is designed to help you easily and efficiently find the information you need to see your design through to completion.

A.2. Training

You can take up the following training to aid your FPGA design journey:

- University Self-Guided Lab: Become an FPGA Designer in 4 Hours
- Introduction to Tcl
- Quartus Prime Software Tcl Scripting
- Command Line Scripting Capabilities in the Quartus Prime Pro Edition Software
- Quartus Prime Software Tcl Scripting
- Intel FPGA Power and Thermal Calculator for Intel FPGA Devices
- DSP Builder Advanced Blockset: Getting Started
- Using FPGAs with the Intel oneAPI Toolkits
- Instructor-Led Training: Intel SoC FPGA Basics
- University Self-Guided Lab: Introduction to Simulation and Debug of FPGAs

For more Intel FPGA trainings, refer to the Intel FPGA Technical Training Catalog and Intel FPGA channel on YouTube*.

B. Using the Design Space Explorer II

The Design Space Explorer II tool (**Tools ➤ Launch Design Space Explorer II**) allows you to find optimal project settings for resource, performance, or power optimization goals.

Refer to the following links for information on how to use the tool and what each option means in the GUI:

- Design Space Explorer II Tool
  - Setup Page
  - Project Page
  - Exploration Page
  - Status Page
  - Report Page
- Using Design Space Explorer

B.1. Optimizing Project Settings

Optimize project settings to meet your design goals.

The Quartus Prime Design Space Explorer II iteratively compiles your project with various setting combinations to find the optimal settings for your goals. Alternatively, you can create a project revision or project copy to manually compare various project settings and design combinations.

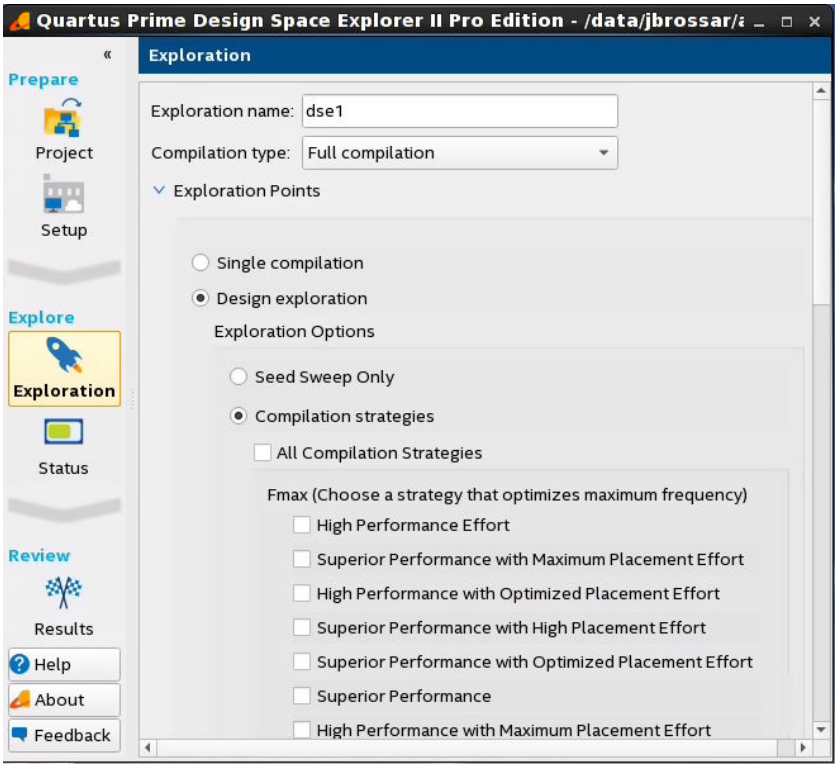
B.1.1. Optimizing Settings with Design Space Explorer II

Design Space Explorer II (DSE II) processes a design using combinations of settings and constraints, and reports the best combination of settings and constraints for the design. You can also take advantage of the DSE II parallelization abilities to compile on multiple computers.

In DSE II, an *exploration point* is a collection of Analysis & Synthesis, Fitter, and placement settings, and a group of exploration points is a *design exploration*. A design exploration can also include different fitter seeds.

DSE II compiles the design using the settings corresponding to each exploration point. When the compilation finishes, DSE II evaluates the performance data against an optimization goal that you specify. You can direct the DSE II to optimize for timing, area, or power. DSE II attempts multiple seeds to identify one meeting your requirements. DSE II can run different compilations on multiple computers in parallel to streamline timing closure.

Figure 72. Design Space Explorer II



You can run DSE II at any step in the design process. However, because large changes in a design can neutralize gains achieved from optimizing settings, Intel recommends that you run DSE II late in the design cycle.

Note: When comparing the results of different seed sweeps with DSE II, changing any of the following variables can cause differences in the compilation results between seed sweeps, resulting in a somewhat different fit in each case:

- The number or type of CPUs that DSE II uses to perform the seed sweeps
- Any change to the operating system
- Any change to source file content or location
- Any change to the Compiler settings or Timing Analyzer settings

For more information, refer to Fitter Seed.

B.1.1.1. DSE II Computing Resources

You can configure DSE II to take advantage of your computing resources to run the design explorations. In the DSE II GUI, the **Setup** page contains the job launch options, and the **Status** page allows you to monitor and control jobs.

DSE II supports running compilations on your local computer or a remote host through LSF, SSH or Torque. For SSH, you can also define a comma-separated list of remote hosts.

If you have a laptop or standard computer, you can use the single compilation feature to compile your design on a workstation with higher computing performance and memory capacity.

When running on a compute farm, you can direct the DSE II to safely exit after submitting all the jobs while the compilations continue to run until completion. Optionally, you can receive an e-mail when the compilations are complete.

If you launch jobs using SSH, the remote host must enable public and private key authentication. For private keys encrypted with a pass phrase, the remote host must run the ssh key agent to decrypt the private key, so the `quartus_dse` executable can access the key.

Note: Windows remote hosts require Cygwin's sshd server and PuTTY.

B.1.1.2. DSE II Optimization Parameters

DSE II provides a collection of predefined exploration spaces that focus on what you want to optimize. Additionally, you can define a set of compilation seeds. The number of explorations points is the number of seeds multiplied by the number of exploration modes.

Note: The availability of predefined spaces depends on the device family that the design targets.

In the DSE GUI, you specify these settings in the **Exploration** page.

B.1.1.3. DSE II Result Management

DSE II compares the compilation results to determine the best Quartus Prime software settings for the design. The **Report** page displays a summary of results.

In an exploration, DSE II selects the best worst-case slack value from among all timing corners across all exploration points. If you want to optimize for worst-case setup slack or hold slack, specify timing constraints in the Quartus Prime software.

Disk Space

By default, DSE II saves all the compilation data. You can save disk space by limiting the type of files that you want to save after a compilation finishes. These settings are in the **Exploration** page, **Results** section.

Reports

DSE II has reporting tools that help you quickly determine important design metrics, such as worse-case slack, across all exploration points.

DSE II provides a performance data report for all points it explores and saves the information in a `project-name.dse.rpt` file in the project directory. DSE II archives the settings of the exploration points in Quartus Prime Archive Files (`.qar`).

B.1.2. Optimizing Settings with Project Revisions

You can save multiple, named project revisions within your Quartus Prime project (**Project > Revisions**). Each project revision captures a unique set of project settings and constraints, while using the same set of logic design files.

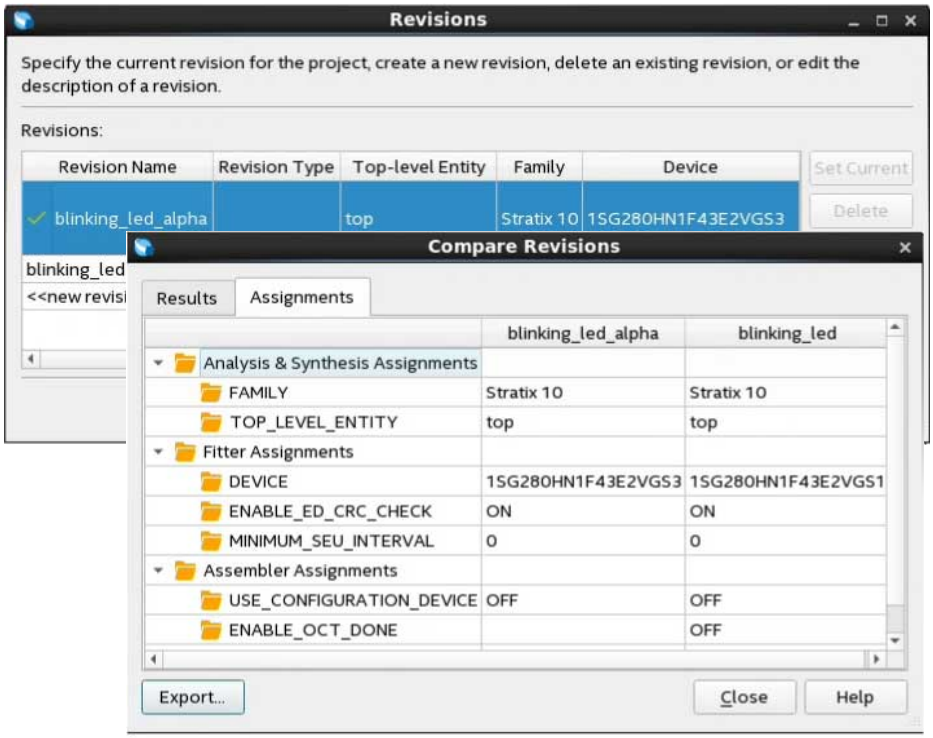
Use revisions to experiment with different settings while preserving the original. Optimize different revisions for separate applications:

- Create a unique revision to optimize a design for different criteria, such as by area in one revision and by f_{MAX} in another revision.
- When you create a new revision the default Quartus Prime settings initially apply.
- Create a revision of a revision to experiment with settings and constraints. The child revision includes all the assignments and settings of the parent revision.

You create, delete, and edit revisions in the **Revisions** dialog box. Each time you create a new project revision, the Quartus Prime software creates a new `.qsf` using the revision name.

To compare each revision's synthesis, fitting, and timing analysis results side-by-side, click **Project > Revisions** and then click **Compare**. In addition to viewing the compilation **Results** of each revision, you can also compare the **Assignments** for each revision. This comparison reveals how different optimization options affect your design.

Figure 73. Comparing Project Revisions



Related Information

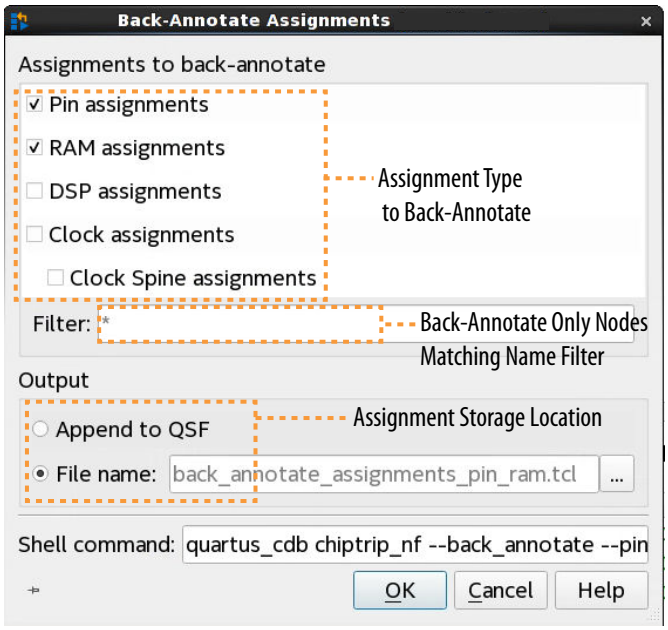
- Project Revision Commands on page 114
- Optimize Settings with Design Space Explorer II
- Design Space Explorer II Tool
- Using Design Space Explorer II (DSE II) Video

B.1.3. Back-Annotating Optimized Assignments

The Compiler maps the elements of your design to specific device resources during fitting. After compilation, you can back-annotate (copy) the Compiler's resource assignments to preserve that same implementation in subsequent compilations. Back-annotation can simplify timing closure by allowing you to lock down placement of your optimized results.

Locking down placement of large blocks related to Clocks, RAMs, and DSPs can produce higher f_{MAX} with less noise. Large blocks like RAMs and DSPs have heavier connectivity than regular LABs, complicating movement during placement. When a seed produces good results from suitable RAM and DSP placement, you can capture that placement with back-annotation. Subsequent compiles can then benefit from the high quality RAM and DSP placement from the good seed.

Figure 74. Back-Annotate Assignments Dialog Box



To back-annotate (copy) the device resource assignments from the last compilation to the project `.qsf` (or to a Tcl file) for use in the next compilation:

1. Run a full compilation, or run the Fitter through at least the **Place** stage.
2. Click **Assignments ➤ Back-Annotate Assignments**.
3. Under **Assignments to back-annotate**, specify whether you want to preserve **Pin assignments**, **RAM assignments**, **DSP assignments**, **Clock assignments**, and **Clock Spine assignments** in the back-annotation.
4. In **Filter**, specify a text string (including wildcards) if you want to filter back-annotated assignments by entity name.
5. Under **Output**, specify whether to save the back-annotated assignments to the `.qsf` or to a Tcl file. A default Tcl file name displays.

Alternatively, you can run back-annotation with the following `quartus_cdb` executable. The **Shell command** field displays the shell command constructed by the options that you specify in the GUI.

```
quartus_cdb chiptrip_nf --back_annotate --pin --ram --dsp --clocks \
    --spines --file "<file>.tcl"
```

Note: Check available arguments by running `quartus_cdb <project> --back_annotate --help`.

Related Information

Back Annotation in Quartus Prime Software

B.2. Running DSE II

Note: Before running DSE II, specify the timing constraints for the design.

This description covers the type of settings that you need to define when you want to run a design exploration. For details about all the options available in the GUI, refer to the Quartus Prime Help.

To perform a design exploration with the DSE II tool:

1. Start the DSE II tool.

   If you have an open project in the Quartus Prime software and launch DSE II, a dialog box appears asking if you want to close the Quartus Prime software. Click **Yes**.
2. In the **Project** page, specify the project and revision that you want to explore.
3. In the **Setup** page, specify whether you want to perform a local or a remote exploration, and set up the job launch.
4. In the **Exploration** page, specify optimization settings and goals.
5. When the configuration is complete, click **Start**.

B.3. Setting Up Remote Farm Using Design Space Explorer II

To launch Design Space Explorer II, in the Quartus Prime Pro Edition GUI, click **Tools ➤ Launch Design Space Explorer II**. Click **Yes** to the message that appears. This closes the Quartus Prime software and launches the Design Space Explorer II.

Under the **Project** tab, click **Open Project** to add your project to the Design Space Explorer II. Refer to Project Page (Design Space Explorer II) for information about all options on this tab.

Use one of the following methods to set up your remote farm:

LSF Remote Farm

Follow these steps to set up your remote machine using the Design Space Explorer II and LSF remote compilation type:

1. Ensure you have set up your LSF environment. If not, request your IT administrator to set up the LSF environment.
2. Once the LSF environment is ready, launch the command line interface and execute the `bsub sleep 60` command.
3. Under the **Setup** tab, select **Remote** compilation type and choose the LSF option from the drop-down list. Populate all mandatory settings under **Specify custom settings for LSF** with the LSF environment-specific information.

   Refer to Setup Page (Design Space Explorer II) for information about all options on this tab.
4. Customize the settings as necessary.
5. Under the **Exploration** tab, expand the **Exploration Points** section.
6. Select **Design exploration**.
7. Under **Exploration Options**, select **Seed Sweep Only**.
8. Under **Seeds**, select **Create**. By default, it must be set to 2 seeds.
9. Click **Start**. Wait until the design exploration is complete.
10. Click the **Results** tab to review the status of the design exploration.

SSH Remote Farm

Follow these steps to set up your remote machine using the Design Space Explorer II and SSH remote compilation type:

1. Install the open-source PuTTY Key Generator tool and launch it.
2. Click **Generate** to generate the public/private key pair.
3. Enter the key passphrase.
4. Click **Save private key** to save the private key with a `.ppk` extension.
5. Click **Save public key** to save the public key as `putty_gen_public_key.pub`.
6. On your machine, change to the SSH directory and copy the contents of the `putty_gen_public_key.pub` file.
7. In the Design Space, under the **Setup** tab, select **SSH** compilation type and specify the following:
   - For **Hostname**, enter the server name.
   - For **private_key**, enter the path to your private key (`.ppk` file).
   - For **SSH Client**, enter the path to the `plink.exe` file. You can find this in the same directory where you installed the PuTTy tool.
   - For **Farm Operating System**, enter your system type (`linux` or `windows`).

   The remaining settings are similar to LSF settings. Refer to Setup Page (Design Space Explorer II) for information about all options on this tab.
8. Under the **Exploration** tab, expand the **Exploration Points** section.
9. Select **Design exploration**.

10. Under **Exploration Options**, select **Seed Sweep Only**.
11. Under **Seeds**, select **Create**. By default, it must be set to 2 seeds.
12. Click **Start**. Wait until the design exploration is complete.
13. Click the **Results** tab to review the status of the design exploration.

C. Document Revision History for Quartus Prime Pro Edition User Guide Getting Started

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Made the following updates in *Introduction to Quartus Prime Pro Edition*: — Included additional steps to the quick start table. — Updated the Quartus Prime software main page image in the table. — Updated the support matrix image to include Agilex 5 device support. — Updated supported features table to include Agilex 5 device support for hyper-aware design flow. — Updated the Intel FPGA developmental tools table to include Agilex 5 device support for PTC. • Revised prerequisite training list in *Prerequisite Knowledge and Training*. • Consolidated the information of design planning into a table in *Planning FPGA Design for RTL Flow*. • Added *Viewing Design Hierarchy and Adding Missing Source Files*. • Removed the Quartus Prime software main page image in *Creating a New FPGA Design Project*. • Revised the IP version upgrade path image in *Upgrading IP Cores*. • Revised *Generating IP Simulation Files* topic entirely. • Revised the note about `.bdf` files in *Managing Logic Design Files*.
2023.12.04	23.4	• Made major reorganization of chapters and topics. • Added additional information to *Introduction to Quartus Prime Pro Edition*. • Renamed *FPGA Basic Design Prerequisites* as *Prerequisite Knowledge and Training* and included a list of trainings. • Revised the information for accessing online design examples in *Creating a New Project from a Design Example*. • Renamed the chapter title "Design Planning" to "Planning FPGA Design for RTL Flow." • Renamed the topic "Plan for Hierarchical and Team-Based Designs" to "Selecting the Design Methodology."

continued...

Document Version	Quartus Prime Version	Changes
		• Added the following: — *Before You Begin* — *Acronyms* — *Navigate Content Through Tasks* — *Selecting the Design Methodology* — *Flat Design Vs. Incremental Block-based Design* — *Partial Reconfiguration Design* — *Related Trainings* — *Migrating Your AMD* Vivado* Project to Quartus Prime Pro Edition* — *Migrating Project From One Device to Another* — *Converting Symbolic BDF Files to Acceptable File Formats* — *Project Path Length Considerations* • Renamed the chapter name from *Migrating to Quartus Prime Pro Edition* to *Selecting a Starting Point for Your Quartus Prime Pro Edition Project*. • Moved *Creating a New FPGA Design Project*, *Migrating Projects Across Operating Systems* and their subtopics from *Managing Quartus Prime Projects* chapter to this chapter. • Chapter renamed as "Working With Intel FPGA IP Cores." • Removed the following topics and added a reference to *Quartus Prime Pro Edition User Guide: Third-party Simulation* where these topics are explained in detail. — *Sourcing Aldec ActiveHDL* or Riviera Pro* Simulator Setup Scripts* — *Sourcing Cadence Incisive* Simulator Setup Scripts* — *Sourcing Cadence Xcelium Simulator Setup Scripts* — *Sourcing QuestaSim* Simulator Setup Scripts* — *Sourcing Synopsys VCS Simulator Setup Scripts* — *Sourcing Synopsys VCS MX Simulator Setup Scripts* • Moved *Creating a New FPGA Design Project*, *Migrating Projects Across Operating Systems* and their subtopics to *Selecting a Starting Point for Your Quartus Prime Pro Edition Project* chapter. • In *Managing Logic Design Files*, added information about converting `.bdf` to `.v` or `.vhd` file. • Removed "Block Diagram/Schematic Design Files (`.bdf`)" in *Project Files to Include In External Revision Control*. • Added *Viewing Parameter Settings From the Project Navigator*. • Added an appendix about *Using Design Space Explorer II* and included related topics. • Revised the Power and Thermal Calculator (PTC) image in *Planning for Device Power Consumption*.
2023.10.02	23.3	• Updated the compilation dashboard image in *Introduction to Quartus Prime Pro Edition* and *Using the Compilation Dashboard*. • Removed OpenCL support from the "Intel Quartus Prime Feature Support Matrix" image in *Selecting an Quartus Prime Software Edition*. • Made a minor correction in *Logic Lock Region Assignment Examples*. • Revised the "Quartus Prime Pro Edition IP Version Upgrade Paths" image in *Upgrading IP Cores*. • Updated the dashboard image in *Using the Compilation Dashboard*.
2023.06.26	23.2	• Updated *Power Analyzer Settings* screenshot for new settings name.

continued...

Document Version	Quartus Prime Version	Changes
2023.04.03	23.1	• Updated product family name to "Intel Agilex 7." • Added note to *Upgrade Project Assignments and Constraints* about new prompt to update operating temperatures in a migrated project. • Updated *Support for the IEEE 1735 Encryption Standard* topic for new installed location of public encryption key. • Updated *Intel Quartus Prime Pro Edition IP Version Upgrade Paths* support chart.
2022.12.12	22.4	• Updated *Plan for the Target Device or Board* topic for board-aware features. • Revised *Applying Preset Parameters for Specific Applications* topic for board-aware features. • Added new *Viewing, Applying, and Deleting IP Presets* topic. • Added new *Example IP Preset File (.qprs)* topic. • Revised *Customizing IP Presets* topic for board-aware features. • Added new *Defining Preset Pin Assignments* section. • Revised *Creating a New FPGA Design Project* for board-aware features. • Added *Using the Board-Aware Flow* topic. • Added *Creating a New Project from a Design Example* topic. • Added *Family, Device & Board Settings* topic. • Added *Accessing Pre-Installed Design Examples* topic. • Added *Accessing Online Design Examples* topic. • Added *Accessing Downloaded Design Examples* topic. • Added *Internet Connectivity Options* topic. • Added *Design Examples Options* topic. • Added *Specifying a Target Board for the Project* topic.
2022.06.20	22.2	• Added new *Top FAQs* navigation to document cover. • Revised *Introduction* to add FPGA definition and device selection footnote. • Added new *FPGA Basic Design Prerequisites* topic. • Added new *Experiment with a Design Example* topic. • Removed obsolete *Simultaneous Switching Noise Analysis* topic from this basic discussion.
2022.03.28	22.1	• Added information about Power and Thermal Calculator in *Plan for Device Power Consumption*. • Removed references to obsolete Advisors from *Optimizing Project Settings* topic. • Added *Viewing Synthesis Warning Messages* topic. • Removed the topic *Automated Problem Reports*.
2021.10.04	21.3	• Added support for Questa*-Intel FPGA Edition simulator. • Removed support for ModelSim - Intel FPGA Edition simulator. • Updated *Quartus Prime Pro Edition IP Version Upgrade Paths* figure for latest versions.
2021.06.21	21.2	• Added *Version-Compatible Compilation Database Support* table. • Added "Promoting Critical Warnings to Errors" topic.
2021.03.29	21.1	• Enhanced *Simulating Intel FPGA Designs* topic with screenshot, links, and additional contextual details. • Updated supported simulator versions and removed support for Cadence Incisive Enterprise* in *Simulator Support* topic. • Revised *Generating IP Simulation Files* topic for new simulation file output options.

continued...

Document Version	Quartus Prime Version	Changes
		• Revised *Using the EDA Netlist Writer* wording for clarity. • Added "Creating Database-Only Archives" topic. • Added "Promoting Critical Warnings to Errors" topic
2020.11.09	20.3	• Revised "Introduction to Intel FPGA IP Cores" topic to include Bridges and Adapters and Intel FPGA Interconnect categories in IP Catalog. Updated IP Catalog image. • Revised wording of "Intel FPGA IP Versioning" topic for clarity. • Added screenshot to "Checking the IP License Status" topic. • Added "IP Version Upgrade Paths" diagram to "Upgrading IP Cores" topic. • Updated IP Port Differences Report image in "Troubleshooting IP or System Upgrade" topic.
2020.09.28	20.3	• Updated GUI screenshot in Introduction. • Updated "Back-Annotate Optimized Assignments" for support of pins, clocks, RAMs, and DSPs.
2020.05.01	20.1	• Added note about `.qar` file requirements to "Design Guidelines for Component Instances" topic.
2019.09.30	19.3	• Added compilation support for Agilex 7 devices. • Added "Checking the IP License Status" topic. • Added details to "Support for the IEEE 1735 Encryption Standard." • Added Intel FPGA IP Versioning" topic. • Added "Disabling Automated Problem Reports" topic. • Added "Suppressing Messages" topic.
2019.05.13	18.1	• Added archives topic. • Updated the keyname and added --help information to "Support for the IEEE 1735 Encryption Standard."
2018.10.24	18.1	• Updated information about obtaining IEEE 1735 Encryption key.
2018.09.24	18.1	• Added screenshot of Quartus Prime Pro Edition GUI. • Moved information about specifying the target board to "Specifying the Target Device or Board" in *Managing Projects* chapter. • Retitled "Creating Design Specifications" to "Create a Design Specification and Test Plan." • Retitled "Selecting Intellectual Property Cores" to "Plan for Intellectual Property Cores." • Retitled "Using Standard Interfaces" to "Plan for Standard Interfaces." Corrected references to Platform Designer. • Retitled "Device Selection" to "Plan for the Target Device." Updated this content to correct Platform Designer names. • Moved "Setting Pin Assignments" to *Managing Projects* chapter as "Generating Pin Assignments for a Target Board." • Retitled "Estimating Power" to "Plan for Device Power Consumption." Reorganized this topic into sections for EPE and Power Analyzer. • Added link to "Simulator Support, *Third-Party Simulation User Guide* • Retitled "Planning for Device Programming or Configuration" to "Plan for Device Programming" • Retitled "Selecting Third-Party EDA Tools" to "Plan for other EDA Tools." • Retitled "Planning for On-Chip Debugging Tools" to "Plan for On-Chip Debugging Tools."

continued...

Document Version	Quartus Prime Version	Changes
		• Retitled *Design Planning with the Intel Quartus Prime Software* to *Design Planning* • Added information about removing assignments from the qsf file that point to legacy output files. • Added statement that the Quartus Prime software installer does not support spaces in the installation path. • Added "Intel FPGA IP Best Practices" topic. • Divided "Introduction to Intel FPGA IP Cores" into separate chapter of *Getting Started User Guide*. • Subdivided "Exporting, Archiving, and Migrating Projects" into separate sections. • Described migration of full chip database in "Exporting a Version-Compatible Compilation Database" topic. • Described automated `.qdb` partition export in "Exporting a Design Partition" topic. • Added "Viewing Quartus Database File Information" topic. • Added "Specifying the Target Device or Board" topic. • Divided "Introduction to Intel FPGA IP Cores" into separate chapter. • Moved "IP Core Best Practices" topic to *Introduction to Intel FPGA IP Cores* chapter. • Moved "Factors Affecting Compilation Results" topic to *Design Compilation: Intel Quartus Prime Pro Edition User Guide*.
2018.05.07	18.0	• Initial release as separate chapter of *Getting Started User Guide.* Separated *Migrating to Quartus Prime Pro Edition* as independent chapter in user guide. • Initial release as separate chapter of *Getting Started User Guide.* Separated *Design Planning* as independent chapter in user guide. • Initial release as separate chapter of *Getting Started User Guide.* Separated *Introduction to Intel FPGA IP Cores* as independent chapter in user guide. • Updated screenshots of IP Catalog and Parameter Editor for latest IP names. • Added note about Generate Combined Simulator Setup Scripts command limitations. • Added information about generation of simulation files for Xcelium* • Initial release as chapter of *Getting Started User Guide.* • Revised "Exporting a Design Partition" topic to add Include entity-bound SDC files for the selected partition option, to add prerequisite steps, and to remove import step covered in separate topic. • Changed title of "Managing Team-Based Designs" to "Exporting, Archiving, and Migrating Projects" and updated content. • Changed title of "Migrating Compilation Results Across Software Versions" to "Exporting the Compilation Database" and updated content. • Changed title of "Exporting the Results Database" to "Exporting a Version-Compatible Design Compilation Database" and updated content. • Changed title of "Importing the Results Database" to "Importing a Version-Compatible Design Compilation Database" and updated content. • Changed title of "Cleaning the Project Database" to "Cleaning the Project Compilation Database." • Updated screenshots of IP Catalog and Parameter Editor for latest IP names.

continued...

Document Version	Quartus Prime Version	Changes
2017.11.06	17.1	• Described Quartus Prime tool name updates for Platform Designer (Qsys), Interface Planner (BluePrint), Timing Analyzer (TimeQuest), Eye Viewer (EyeQ), and Intel Advanced Link Analyzer (Advanced Link Analyzer). • Added Verilog HDL Macro example. • Updated for latest Intel branding conventions. • Added Verilog HDL Macro example. • Updated for latest Intel branding conventions. • Revised product branding for Intel standards. • Revised topics on Intel FPGA IP Evaluation Mode (formerly OpenCore).
2017.05.08	17.0	• Removed statement about limitations for safe state machines. The Compiler supports safe state machines. State machine inference is enabled by default. • Added reference to Block-Based Design Flows. • Removed procedure on manual dynamic synthesis report generation. The Compiler automatically generates dynamic synthesis reports when enabled. • Removed statement about limitations for safe state machines. The Compiler supports safe state machines. State machine inference is enabled by default. • Added note that IP core encryption is supported only in Quartus Prime Pro Edition. • Revised product branding for Intel standards.
2016.10.31	16.1	• Implemented Intel rebranding. • Added reference to Partial Reconfiguration support. • Added to list of Quartus Prime Standard Edition features unsupported by Quartus Prime Pro Edition. • Added topic on Safe State Machine encoding. • Described unsupported Quartus Prime Standard Edition physical synthesis options. • Removed deprecated **Per-Stage Compilation (Beta)** Compilation Flow. • Changed title from "Remove Filling Vectors" to "Remove Unsized Constant". • Implemented Intel rebranding. • Described unsupported Quartus Prime Standard Edition physical synthesis options. • Changed title from "Remove Filling Vectors" to "Remove Unsized Constant". • Removed references to `.qsys` file creation during Quartus Prime Pro Edition stand-alone IP generation. • Added references to `.ip` file creation during Quartus Prime Pro Edition stand-alone IP generation. • Updated IP Core Generation Output files list and diagram. • Indicated distinctions between Quartus Prime Pro Edition and Quartus Prime Standard Edition features. • Added Support for IP Core Encryption topic.
2016.05.03	16.0	• Removed software beta status and revised feature set. • Added topic on Safe State Machine encoding. • Added Generating Dynamic Synthesis Reports. • Corrected statement about Verilog Compilation Unit. • Corrected typo in Modify Entity Name Assignments. • Added description of Fitter Plan, Place and Route stages, reporting, and optimization. • Added **Per-Stage Compilation (Beta)** Compilation Flow.

continued...

Document Version	Quartus Prime Version	Changes
		• Added Platform Designer information. • Added OpenCL and Signal Tap with routing preservation as unique Pro Edition features. • Clarified limitations for multiple Logic Lock instances in the same region. • Added topic on Safe State Machine encoding. • Corrected statement about Verilog Compilation Unit. • Corrected typo in Modify Entity Name Assignments. • Clarified limitations for multiple Logic Lock instances in the same region.
2015.11.02	15.1	• First version of document.

D. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Platform Designer

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Online Version
Send Feedback

683609
2024.04.01

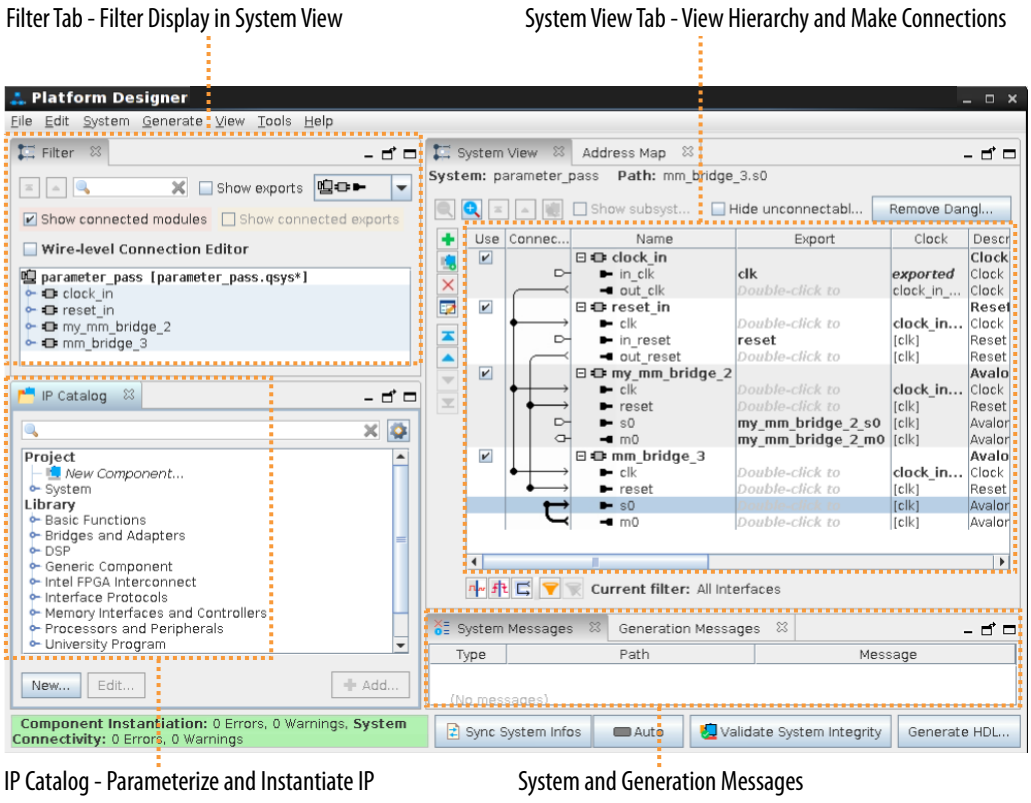
Contents

1. Creating a System with Platform Designer

The Quartus® Prime software includes the Platform Designer system integration tool. Platform Designer simplifies the task of defining and integrating custom IP components (IP cores) into your FPGA design.

Platform Designer automatically creates interconnect logic from high-level connectivity that you specify. The interconnect automation eliminates the time-consuming task of specifying system-level HDL connections.

Figure 1. Platform Designer GUI



Platform Designer allows you to specify interface requirements and integrate IP components within a graphical representation of the system. The Quartus Prime software installation includes the Intel FPGA IP library available from the IP Catalog in Platform Designer.

You can integrate optimized and verified Intel FPGA IP cores into a design to shorten design cycles and maximize performance. Platform Designer also supports integration of IP cores from third-parties, or custom components that you define.

Platform Designer supports a hierarchical framework that offers fast response times for interconnecting large systems and blackbox entities. Platform Designer supports a variety of design entry methods, such as register transfer level (RTL) and schematic entry. Platform Designer supports the creation of your own custom components, as well as generic components that define only the interface and signal connections to the rest of the system.

Platform Designer provides support for the following:

- Create and reuse components—define and reuse custom parameterizable components in a Hardware Component Definition File (`_hw.tcl`) that describes and packages IP components.
- Define generic IP components—instantiate generic IP components without an HDL implementation.
- Incremental generation—optimize and generate IP components incrementally.
- Avalon® to AXI interconnect—Platform Designer generates appropriate types of interconnect logic to handle protocol differences.
- Hierarchical system support—generates a separate `.ip` file that isolates the system from the IP component parameterization. Change parameters of a single IP component without regeneration of other IP components.
- Command-line support—optionally use command-line utilities and scripts to perform functions available in the Platform Designer GUI.
- Up to 64-bit addressing.
- Optimization of interconnect and pipelining within the system and auto-adaptation of data widths and burst characteristics.
- Inter-operation between standard protocols.

1.1. Platform Designer Interface Support

Platform Designer is most effective when you use standard interfaces available in the IP Catalog to design custom IP. Standard interfaces operate efficiently with Intel FPGA IP components, and you can take advantage of the bus functional models (BFMs), monitors, and other verification IP that the IP Catalog provides.

Platform Designer also supports connections between Avalon and AXI interfaces by generating the interconnect logic. This logic enables you to handle the protocol difference. Platform Designer creates the interconnect logic by converting all the protocols to a proprietary packet format. Then, the tool routes the packet through network switches to the appropriate agents. Here, the packet converts to the agent's protocol.(1)

Platform Designer supports the following interface specifications:

- Intel FPGA Avalon Memory-Mapped and Streaming
- Arm* AMBA* 3 AXI (version 1.0)
- Arm AMBA 4 AXI (version 2.0)

(1) This document now refers to the Avalon "host" and "agent," and the AXI "manager" and "subordinate," to replace formerly used terms. Refer to the current *AMBA AXI and ACE Protocol Specification* for the latest AMBA AXI and ACE protocol terminology.

- Arm AMBA 4 AXI-Lite (version 2.0)
- Arm AMBA 4 AXI-Stream (version 1.0)
- Arm AMBA 3 APB (version 1.0)

IP components (IP Cores) can have any number of interfaces in any combination. Each interface represents a set of signals that you can connect within a Platform Designer system, or export outside of a Platform Designer system.

Platform Designer IP components can include the following interface types:

Table 1. IP Component Interface Types

Interface Type	Description
Memory-Mapped	Connects memory-referencing host devices with agent memory devices. Host devices can be processors and DMAs, while agent memory devices can be RAMs, ROMs, and control registers. Data transfers between Avalon Memory Mapped host and agent may be uni-directional (read only or write only), or bi-directional (read and write).
Streaming	Connects Avalon Streaming sources and sinks that stream unidirectional data, as well as high-bandwidth, low-latency IP components. Streaming creates datapaths for unidirectional traffic, including multichannel streams, packets, and DSP data. The Avalon interconnect is flexible and can implement on-chip interfaces for industry standard telecommunications and data communications cores, such as Ethernet, Interlaken, and video. You can define bus widths, packets, and error conditions.
Interrupts	Connects interrupt senders to interrupt receivers. Platform Designer supports individual, single-bit interrupt requests (IRQs). In the event that multiple senders assert their IRQs simultaneously, the receiver logic (typically under software control) determines which IRQ has highest priority, then responds appropriately.
Clocks	Connects clock output interfaces with clock input interfaces. Clock outputs can fan-out without the use of a bridge. A bridge is required only when a clock from an external (exported) source connects internally to more than one source.
Resets	Connects reset sources with reset input interfaces. If your system requires a particular positive-edge or negative-edge synchronized reset, Platform Designer inserts a reset controller to create the appropriate reset signal. If you design a system with multiple reset inputs, the reset controller ORs all reset inputs and generates a single reset output.
Conduits	Connects point-to-point conduit interfaces, or represent signals that you export from the Platform Designer system. Platform Designer uses conduits for component I/O signals that are not part of any supported standard interface. You can connect two conduits directly within a Platform Designer system as a point-to-point connection. Alternatively, you can export conduit interfaces and bring the interfaces to the top-level of the system as top-level system I/O. You can use conduits to connect to external devices, for example external DDR SDRAM memory, and to FPGA logic defined outside of the Platform Designer system.

Related Information

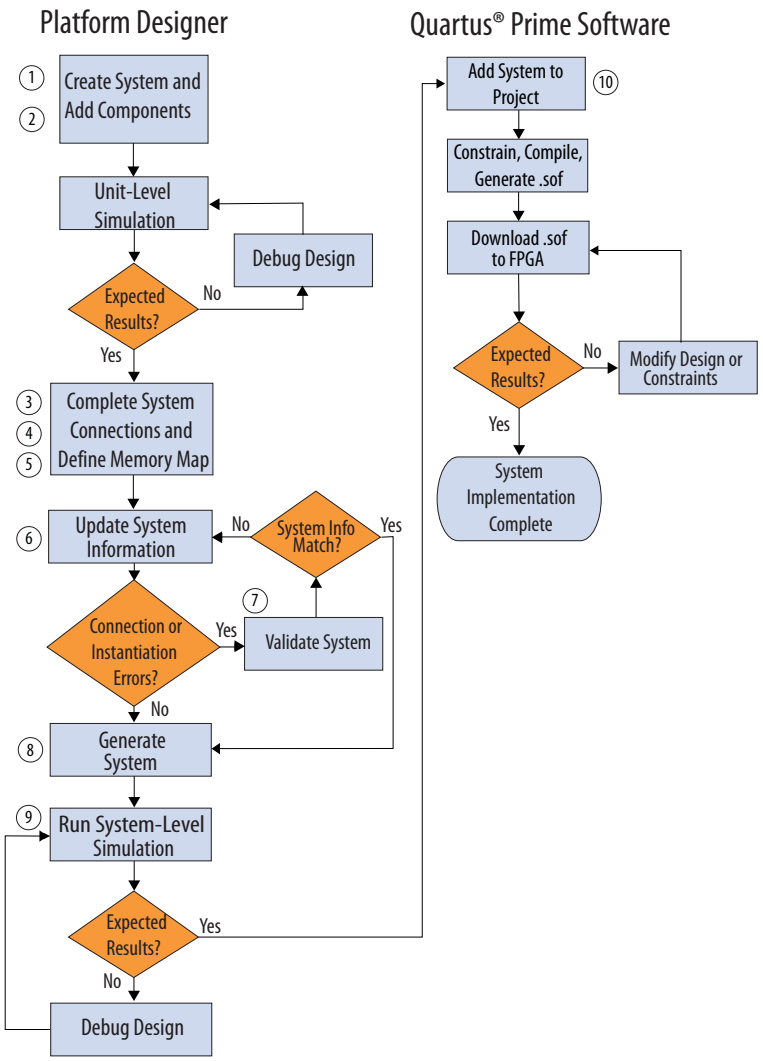
Exporting HDL Parameters to a System on page 189

1.2. Platform Designer System Design Flow

You can use the Platform Designer GUI to quickly create and customize a Platform Designer system for integration with an Quartus Prime project. Alternatively, you can perform many of the functions available in the Platform Designer GUI at the command-line, as Platform Designer Command-Line Utilities on page 441 describes.

When you create a system in the GUI, Platform Designer creates a `.qsys` file that represents the system in your Quartus Prime software project.

Figure 2. Platform Designer System Design Flow



1.3. Creating or Opening a Platform Designer System

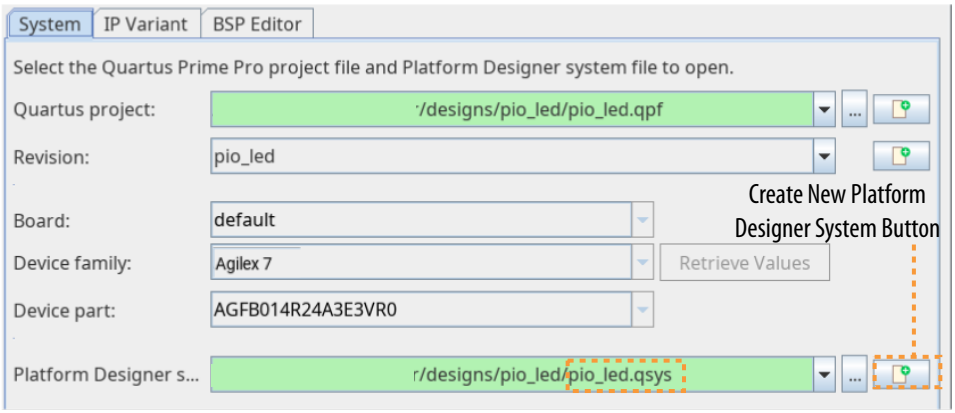
You can launch Platform Designer from the Quartus Prime software to create or open a Platform Designer system.

When you create or open a system, Platform Designer requires that you specify the Quartus Prime project to contain this system. If this project does not yet exist, you can define a new project from within Platform Designer. Alternatively, you can specify an existing project. When you launch Platform Designer with an Quartus Prime project open, Platform Designer automatically specifies that project by default. The target device or board for the system reflects the project settings by default. Refer to Using the Board-Aware Flow in Platform Designer on page 18.

Follow these steps to create or open a Platform Designer system:

1. In the Quartus Prime software, click **File ➤ Open Project** to open the Quartus Prime project that you want to include the Platform Designer system. You can optionally skip this step and launch Platform Designer in view-only mode without opening a project.
2. Click **Tools ➤ Platform Designer**. Platform Designer launches and displays the **Open Project** dialog box automatically.
3. Specify the **Quartus project**. If you have a project open, this project name appears automatically. Otherwise, browse for an existing project, or click the **Create New Quartus Project** button and specify a new project name. Selecting **None** for **Quartus project** opens Platform Designer in view-only mode.

Figure 3. System Tab of Open System Dialog Box



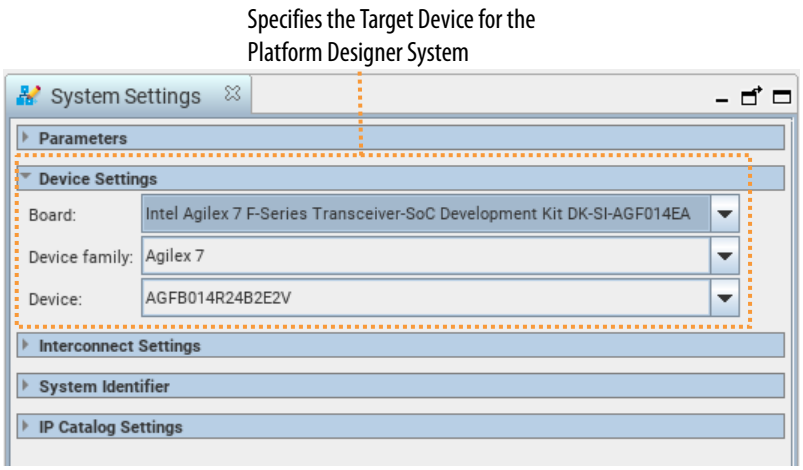
4. Select the **Platform Designer system**, or click the **Create New Platform Designer System** button and specify the name of a new system. Optionally select a specific revision of your project, or click the **Create New Revision** button and define a new project revision.
5. Change the project associated with a Platform Designer system at any time by clicking **File ➤ Select Quartus Project** in Platform Designer.

1.3.1. Specifying the Target FPGA Device or Board for a System

When you create a new Platform Designer system, the system settings automatically reflect the target device or board that the current Quartus Prime project targets. The Platform Designer system generation output is specific to the target Intel® FPGA **Device family** or **Board** specified for the project and system. The available IP components, parameters, and output options for your system vary according to the target **Device family** or **Board**.

Note: The **System Settings** tab now replaces the **Device Family** tab in Platform Designer.

Figure 4. System Settings Tab in Platform Designer



After creating a system, you can modify the target **Device family** or **Board** setting for your system on the Platform Designer **System Settings** tab. If you specify a target **Device family** or **Board** that is different from the current project settings, Platform Designer updates the target device family or board for the project to match the **Device family** or **Board** specification. Platform Designer prompts you to upgrade any IP components that are incompatible with the **Device family** or **Board** that you specify.

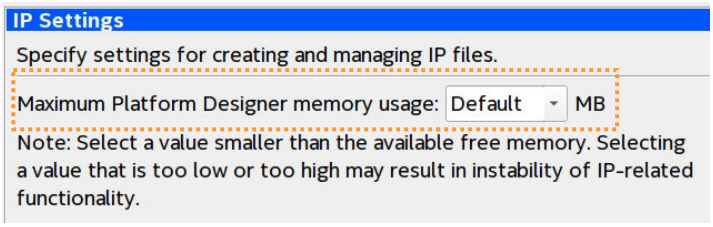
Note: To get started quickly with the board-aware flow, refer to Using the Board-Aware Flow in Platform Designer on page 18.

1.3.2. Specifying Additional Application Memory

If Platform Designer requires more than the default memory to run efficiently, you can increase the amount of application memory.

- From within the Quartus Prime software, increase memory for your Platform Designer system, by clicking **Tools ➤ Options ➤ IP Settings**, and then specifying the amount of memory with the **Maximum Platform Designer memory usage** option.
- From the command-line, you can add an option to increase the memory. For example, the following `qsys-edit` command allows you to open Platform Designer with 2 gigabytes of memory.

Figure 5. Specifying Additional Application Memory for Platform Designer




```
qsys-edit --jvm-max-heap-size=2g
```

1.3.3. Synchronizing IP File References

When you open a system containing IP components, Platform Designer confirms that the list of IP files in your Platform Designer system matches the list of IP files included in the corresponding Quartus Prime project.

The **IP Synchronization Result** dialog box automatically displays any discrepancies between these IP file references in the system.

To manually start a check for IP reference mismatches between the system and corresponding Quartus Prime project:

1. In Platform Designer, click **File ➤ Synchronize IP File References**.
2. View the results of the synchronization. Platform Designer identifies the following types of mismatches with the IP synchronization:

Table 2. IP Synchronization Results

Mismatch Type	Description
Duplicate IP files	The IP files references in the Platform Designer system and the associated Quartus Prime project match. These IP files contain the same name, but are present in different locations. In such cases, the IP files referenced in the Quartus Prime project takes precedence. Platform Designer replaces the IP file reference in the system with the one in the Quartus Prime project during compilation. *Note:* If the Quartus Prime project contains more than one IP of the same file name, Platform Designer retains the first instance and removes all other occurrences of the IP file with the specific name.
Missing IP files	Lists the IP file references missing from Platform Designer system and the corresponding Quartus Prime project. In such cases, Platform Designer allows you to specify the active IP file.
Missing Platform Designer IP files	Lists the IP file references missing from your Platform Designer system that the Quartus Prime project references. If Platform Designer locates a valid reference in the Quartus Prime project, it replaces the missing reference in the Platform Designer system with IP file reference from the Quartus Prime project.
Missing Quartus IP files	Lists the IP file references missing from your Quartus Prime project that the Platform Designer system references. Platform Designer adds the missing IP file reference to the Quartus Prime project. If the project's `.qsf` file already contains reference to the missing IP file, but the file cannot be located in the specified path, Platform Designer removes the reference in the `.qsf` file, and adds the reference to the IP file in the Platform Designer system.

1.3.4. Converting Incompatible Components

If you open a Platform Designer system with incompatible components, Platform Designer prompts you to convert these components to the current Platform Designer format. On conversion, the **Platform Designer Conversion Results** dialog box appears, listing all the converted system and IP source files.

Platform Designer stores the converted `.ip` files inside an `ip` folder, relative to the Platform Designer system file (`.qsys`) location. Platform Designer prefixes the system name to the `.ip` file name. Platform Designer automatically adds these converted files to the associated Quartus Prime project. Ensure that you maintain these `.ip` files, along with your system files.

1.4. Using the Board-Aware Flow in Platform Designer

Platform Designer allows you to create a system that targets a specific development board, rather than only targeting a specific FPGA device. When you target a specific development board, Platform Designer is *aware* of the target board (board-aware) which simplifies the IP parameterization, pin assignments, and exporting interfaces for the system.

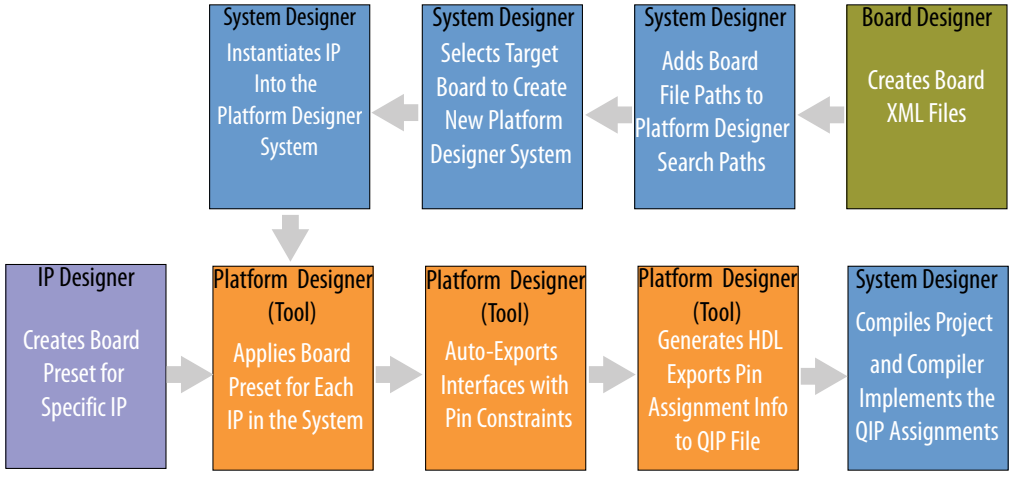
The board-aware flow accelerates the process of appropriately configuring, connecting, and validating IP for the target board by using IP presets together with a board definition file that specifies the details of a target board. You can use (and reuse) the board definition file and IP presets to automatically include the appropriate IP pin assignments, parameters, and exported interfaces for the target development board during system generation.

To use the board definition file and IP presets together, you first specify the board information in a board file (`_board.xml`), and then apply IP presets that are appropriate for the specific development board. IP preset files specify the list of valid parameters and pin assignments that are appropriate for your target board of choice. You can also define your own custom IP presets and board files. When you generate the IP or system using this board-aware flow, Platform Designer performs the following. Refer to Creating IP Presets Targeting Specific Boards on page 45 for details.

- Presets appropriate IP parameters for the target board.
- Exports pin assignment to a `.qip` file read during design compilation.
- Exports the appropriate interfaces for the board.

In the FPGA system design flow, different engineers often handle specific steps of development. For example, a board engineer often creates the board definition file. An IP designer may create the board presets for specific IP. A system designer can use the board file and IP presets to perform design entry. At each stage, you can share the board file and IP presets to reduce any chance of configuration errors from hand-off between developers or projects.

Figure 6. Board-Aware Flow Typical Tasks and Roles



The board-aware flow helps to ensure the proper hand-off, consistency, and reuse of configuration options across multiple projects, developers, and boards.

Related Information

- AN 988: Using the Board-Aware Flow in the Intel Quartus Prime Pro Edition Software
- Board-Aware Flow Scripting Support on page 447

1.4.1. Accessing FPGA Design Examples

You can optionally base your design project on a pre-verified FPGA design example that targets a specific FPGA board or development kit, or you can start with an empty project. Access available design examples using any of the following methods:

- **Pre-installed design examples**—you can immediately access the design examples that install along with the Quartus Prime software installation at: `<quartus>\acds\quartus\common\board_designs`.
- **Online design examples**—you can access design examples hosted online, which include designs from the Intel FPGA Design Store.
- **Downloaded design examples**—you can access previously downloaded design examples, or any design example that you store in a local drive, under downloaded design examples.

1.4.2. Specifying the Target Board for a Platform Designer System

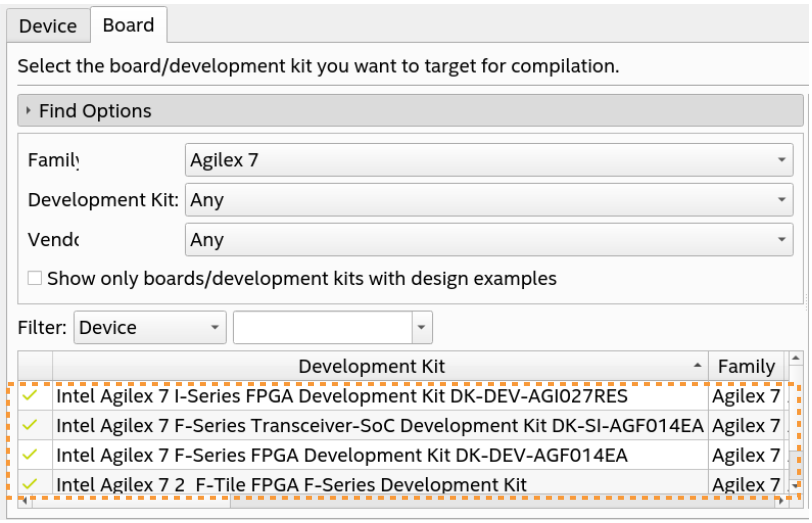
You can specify a target board when you setup any Quartus Prime project. By default, Platform Designer automatically inherits the target board information from the currently open Quartus Prime project.

Specifying the target board for the system, rather than just specifying a target FPGA device, helps to ensure the appropriate IP parameterization, pin assignments, and export of interfaces for the system.

Use any of these methods to specify a target board for the project:

- New Project Wizard (**File ➤ New Project Wizard**).
- Quartus Prime **Home** page, click **Open Example Project** icon.
- Click **File ➤ Open Example Project**.
- **Board** tab of the **Device** dialog Box (**Assignments ➤ Device ➤ Board**).

Figure 7. Board Tab of Device Dialog Box



1.4.2.1. Changing the Target Board for a Platform Designer System

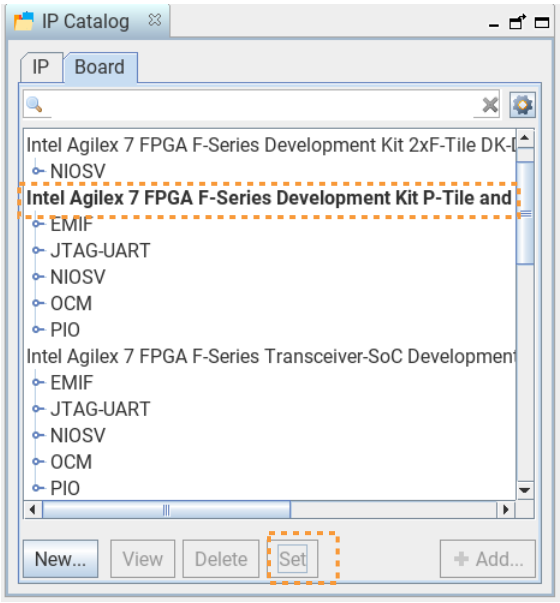
After creating or opening a system, you can use any of the following methods to change the target board for your system:

Figure 8. Change the Target Board in Platform Designer System Settings

Click **System ➤ System Settings** in Platform Designer to change the target board at any time.

Or
In Platform Designer's IP Catalog, click the **Board** tab, then select the target board in the list and click **Set**. The board name appears in bold when set as the target board.

Figure 9. Changing the Target Board in IP Catalog



Alternatively, you can specify the target board by adding board arguments to `qsys-edit` and `qsys-generate` at the command line. The board argument is optional:

```
$qsys-edit --board=[board name]
$qsys-edit --board=[board name] -family=[device family] --part=[device part]
```

`qsys-edit` opens the **Open System** dialog box and automatically sets the board that the argument specifies.

Note: The `--board` option takes precedence over the device `family` and device `part` options. If you specify a `board`, the device `family` and device `part` associated with board automatically update to the board's default device family and part. If you specify the `board`, device `family`, and device `part` , the `board` option with mapped device family and part is selected.

1.4.2.2. Using Board Files

You can define a custom board file (`_board.xml`) that specifies target board information for your system. You can then specify this board as the target for any system. The board file specifies the following information about the target board:

- Board name, such as `Agilex 7 F-Series FPGA Development Kit DK-DEV-AGF014EA`
- Device part, such as AGFB014R24B2E2V
- Device family, such as `Agilex 7`
- Board vendor, such as `Intel`
- Board file version, such as `1.0`

1.4.2.2.1. Creating a New Board File

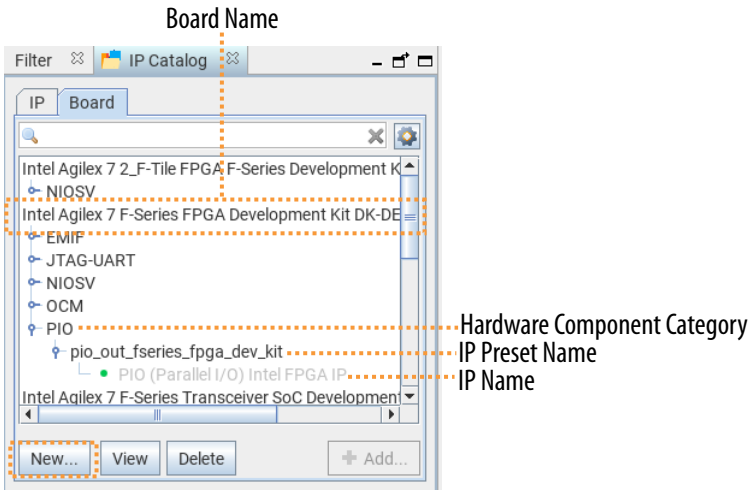
Platform Designer displays all existing board files that have the `_board.xml` extension in the Board Catalog. The Board Catalog is a tab of Platform Designer IP Catalog.

By default, the Board Catalog shows the IPs that you associate with the current selected board file. Under each board file name appears a list of any hardware components associated with the board, such as a button, LED, or JTAG UART. Under each component name appears the IP preset name that applies. Under each preset name appears the IP name to which the preset applies.

To define a new board file in Platform Designer, follow these steps:

1. Click **Tools ➤ Platform Designer**. Platform Designer launches and displays the **Open Project** dialog box automatically.
2. Specify the **Quartus project**. If you have a project open, the project name appears automatically. Otherwise, browse for an existing project, or click the **Create New Quartus Project** button and specify a new project name. If **Device family** or **Device part** are unsynchronized, click **Retrieve Values**.
3. Select the **Platform Designer system**, or click the **Create New Platform Designer System** button and specify the name of a new system.
4. In the Platform Designer IP Catalog, click the **Board** tab. The **Board** tab displays any existing board files (`_board.xml`) in `<quartus>\ip\altera\board_preset_files\`.
5. On the **Board** tab, click the **New** button. The **Create New Board** dialog box appears with options that Create New Board Dialog Box Options on page 23 describes.

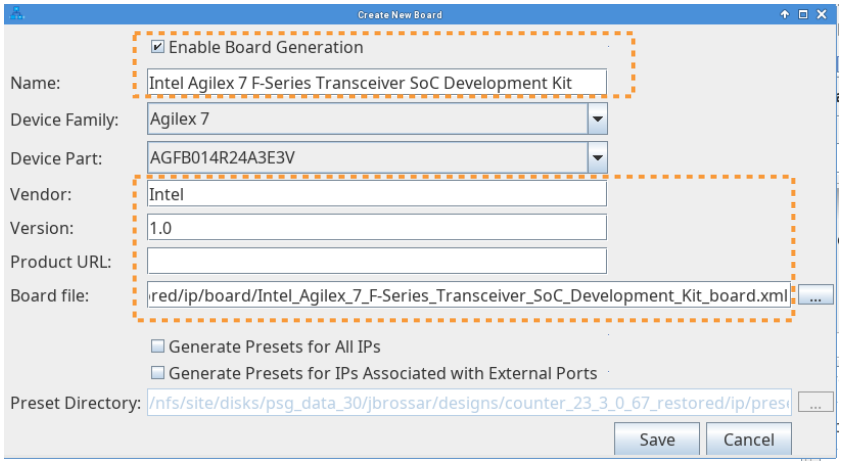
Figure 10. Board Catalog Displays Existing Boards Defined



6. To create a new board file, make sure that **Enable Board Generation** is on.
7. Specify the board **Name**, target **Device family**, **Device part**, board **Vendor**, and board file **Version**. By default, **Device part** and **Device family** reflects the current project settings.
8. For **Product URL**, optionally specify URL of a board product description online.

9. Next to **Board file**, click the browse (**...**) button to specify a **File name**. Platform Designer automatically adds the `_board.xml` extension. Click **OK**.

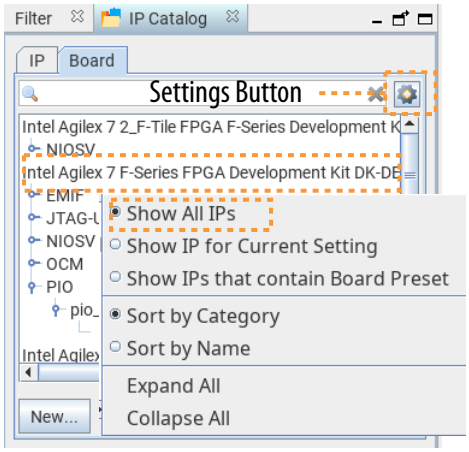
Figure 11. Create New Board Dialog Box



Note: Specifying an existing board file name for **Board file** overwrites the selected board file with the new definition.

10. In the **Create New Board** dialog box, click the **Save** button. If the directory that you specify is new, Platform Designer confirms adding this directory to the search path.
11. To view the new board in the Board Catalog, click the Settings button on the right side of the search field, and select **Show All IPs**.

Figure 12. Agilex™ 7 FPGA DevKit Board Visible in Board Catalog



1.4.2.2.2. Create New Board Dialog Box Options

The **Create New Board** dialog box allows you to specify the details about your target development board in Platform Designer. Platform Designer saves the board data in a board file (`_board.xml`). You define the details of the board and then reuse that board file for other projects that target the same board.

Note: Turning on **Enable Board Generation** in combination with either the **Generate Presets for All IPs** or **Generate Presets for IPs Associated with External Ports** causes the system presets to target the new board that you are creating. Turning on only **Generate Presets for All IPs** or **Generate Presets for IPs Associated with External Ports** with **Enable Board Generation** turned off causes the system presets to target the current board in Platform Designer.

Table 3. Create New Board Dialog Box Settings

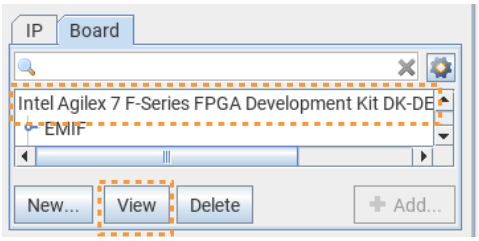
Setting Name	Description
Enable Board Generation	Turn on this option to generate a new board file (`_board.xml`) when you click the **Save** button. Leave this option off to generate only a system presets file (`.qprs`) based on the current system and target board. This option is on by default.
Name	Specifies the name for the board file that appears in the Platform Designer Board Catalog.
Device family	Select the target design family for the Platform Designer system. This setting reflects the current project settings by default.
Device part	Select the target FPGA device part for the Platform Designer system. This setting reflects the current project settings by default.
Vendor	Specifies the vendor of the target board, such as Intel. This setting is optional.
Version	Specifies the version number of the board file, such as 2.0. This setting is optional.
Board file	Specifies the name of the new board file that you are creating.
Preset Directory	Specifies the file name and path of the new system preset file (`.qprs`) that generates when you click the **Save** button. If you instead select an existing preset file for this setting, that file is overwritten with the new preset parameter values.
Generate Presets for All IPs	Turn on this option to enable presets generation for all IPs in a pre-existing Platform Designer system.
Generate Presets for IPs Associated with External Ports	Turn on this option to enable presets generation for IPs with external ports only.
Save	Creates (or overwrites) the board and preset files, according to your specifications in this dialog box.

1.4.2.2.3. Viewing or Deleting Board Files

You can view or delete board files (`_board.xml`) in the Board Catalog. To view the details of an existing board file in the Platform Designer Board Catalog, follow these steps:

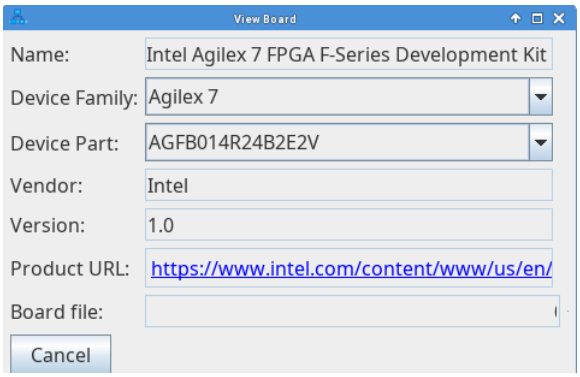
1. In Platform Designer, click the **Board** tab in IP Catalog. (**View ➤ IP Catalog**). The Board Catalog displays any board files (`_board.xml`) in the Platform Designer search path.
2. On the **Board** tab, select the board that you want to view.

Figure 13. Board File in Board Catalog



3. Click the **View** button. The **View Board** dialog box displays the board properties, as Create New Board Dialog Box Options on page 23 describes.

Figure 14. View Board Dialog Box



Deleting a Board File

To delete a board file from the Board Catalog, follow these steps:

1. On the **Board** tab, select the board file that you want to delete.
2. Click the **Delete** button. Click **Yes** to confirm the deletion of the board file.

1.4.3. Generating Board and Preset Files for Existing Systems

You can generate and apply board and preset files to an existing system design that did not originally include board or preset files. This technique allows you to quickly specify appropriate configuration options when you are targeting the same board and IP as the board and preset files. You can apply the board and preset files to the existing system using either the command-line or in Platform Designer.

1.4.3.1. Generating Board and Preset Files for Existing Systems Using Platform Designer

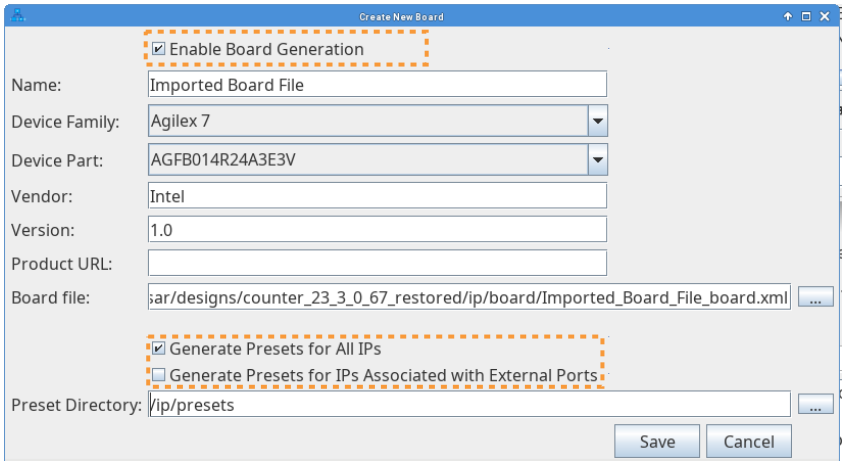
To apply board and preset files to an existing Platform Designer system using the Platform Designer GUI, you first load the pin assignments from the Quartus Prime Settings File (`.qsf`). You then define a new board file that overwrites any existing board file and IP presets. You must ensure that either the **Generate Presets for All IPs** or **Generate Presets for IPs Associated with External Ports** option is on when defining the new board. For details on creating IP presets, refer to Creating IP Presets Targeting Specific Boards.

Note: Before using this flow, you must regenerate the Platform Designer system and run Quartus Prime Analysis and Synthesis.

To apply board and preset files to an existing system using Platform Designer, follow these steps:

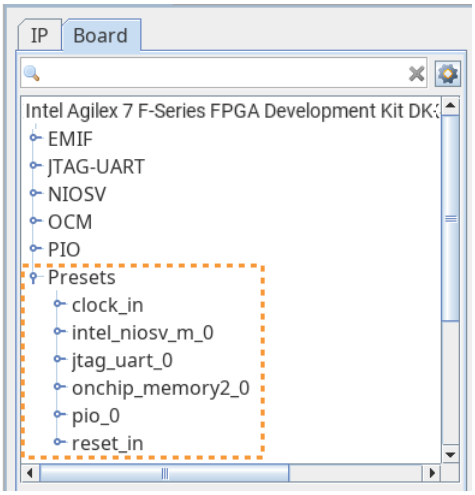
1. In the Quartus Prime software, open and compile the project that contains the target Platform Designer system for board and preset files. The Fitter selects the pin assignments or uses the constraints that you specify in the Pin Planner.
2. In Platform Designer, open the target system for application of board and preset files.
3. Click **File ➤ Load Pin from Quartus Project**. The pin assignments from the current Quartus Prime project load into the Platform Designer system.
4. Confirm the loaded pin assignments in the **Exported Interface** tab, as Editing Pin Assignments for Presets on page 53 describes.
5. Define a new board and presets by clicking **New** in the Platform Designer Board Catalog. The **Create New Board** dialog box opens.

Figure 15. Create New Board Dialog Box



6. Specify options for the new board file, as Creating a New Board File on page 22 describes. When specifying options, turn on the **Enable Board Generation** and either **Generate Presets for All IPs** or **Generate Presets for IPs Associated with External Ports** to overwrite any existing board or preset files for the system.
7. Click **Save** to generate the new board and preset files.
8. View the new system presets in the **Board** tab.

Figure 16. System Presets in Board Tab



1.4.3.2. Generating Presets for Existing Systems with Multiple Instances

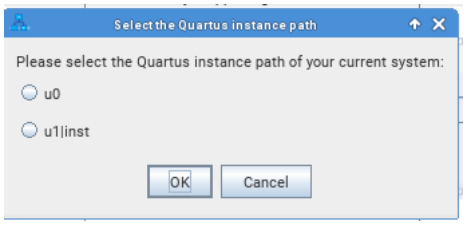
Note: Before using this flow, you must regenerate the Platform Designer system and run Quartus Prime Analysis and Synthesis.

When generating presets for existing Platform Designer systems with two or more separate instances, Platform Designer displays a prompt allowing you to select the relevant instance for the presets.

To generate presets for existing systems with multiple instances, follow these steps:

1. In Platform Designer, open the target system for application of system preset files.
2. Click **File ➤ Load Pin from Quartus Project**.
3. If the system contains more than one instance, select the appropriate instance path and click **OK** to confirm the correct instance path in the dialog box that appears.

Figure 17. Select the Instance Path



The pin assignments from the current Quartus Prime project load into the Platform Designer system.

4. Confirm the loaded pin assignments in the **Exported Interface** tab, as Editing Pin Assignments for Presets on page 53 describes.
5. Define a new board and presets by clicking **New** in the Platform Designer Board Catalog. The **Create New Board** dialog box opens.

6. Specify options for the new board file, as Creating a New Board File on page 22 describes. When specifying options, turn on the **Enable Board Generation** and **Enable System Preset Generation** to overwrite any existing board or preset files for the system.
7. Click **Save** to generate the new board and preset files.
8. View the new system presets in the **Board** tab.

1.4.3.3. Generating Board and Presets for Existing Systems Using Command Line

The command line method of applying board and presets to existing systems implements the same steps in a script that you can perform in the Platform Designer GUI. To apply board and preset files to an existing Platform Designer system using the command line, you must first create a Tcl script that performs the following:

1. Loads the correct Platform Designer package version number with `package require`.
2. Imports the pin assignments from the corresponding Quartus Prime Setting file (`.qsf`) with `load_pin_from_quartus_project`.
3. Saves the system to apply the pin assignment to the system with `save_system`.
4. Exports the system or IP presets and board file with `export_system_preset`.

The following shows an example script that implements these steps:

```
package require -exact qsys <version number>
load_pin_from_quartus_project
save_system
export_system_preset test_preset.qprs
export_board_file "<board name>" <filename_board.xml>
```

Note: The *<version number>* must be 22.4 or later.

The following command runs the example script:

```
$ qsys-script --script=export.tcl --quartus-project=[test_project] \
     --system-file=[test_system.qsys]
```

The following example shows the content of `export.tcl` if there is one instance in a Platform Designer system in the project:

```
package require -exact qsys 22.4
load_pin_from_quartus_project
export_system_preset ip/presets
export_board_file "My Board" test_board.xml
```

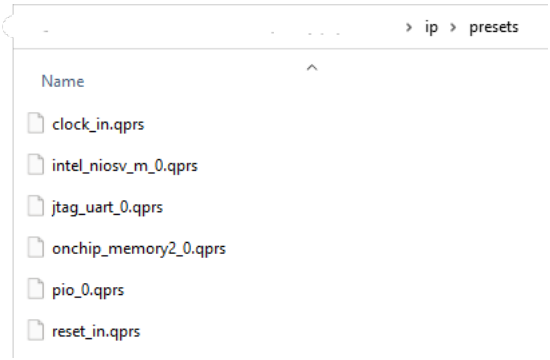
The following example shows the content of `export.tcl` if there is more than one instance in a Platform Designer system in the project:

```
package require -exact qsys 22.4
set instance_paths [get_quartus_instance_path_for_entity]

# select the correct Quartus instance path for your \
    Platform Designer system (e.g using the 1st path)

set my_instance_path [lindex $instance_paths 0]
load_pin_from_quartus_project $my_instance_path
export_system_preset ip/presets
export_board_file "My Board" test_board.xml
```

1.4.3.4. Preset Files Saved

After you create system presets using System Preset Generation, each of the presets for each of the IP in the Platform Designer system save into different `.qprs` files to ensure easier modification (if needed) for preset `.qprs` files. For better organization, you can create a specific folder for presets only, as the following example shows.

Figure 18. Example of Preset Files Saved



1.5. Viewing a Platform Designer System

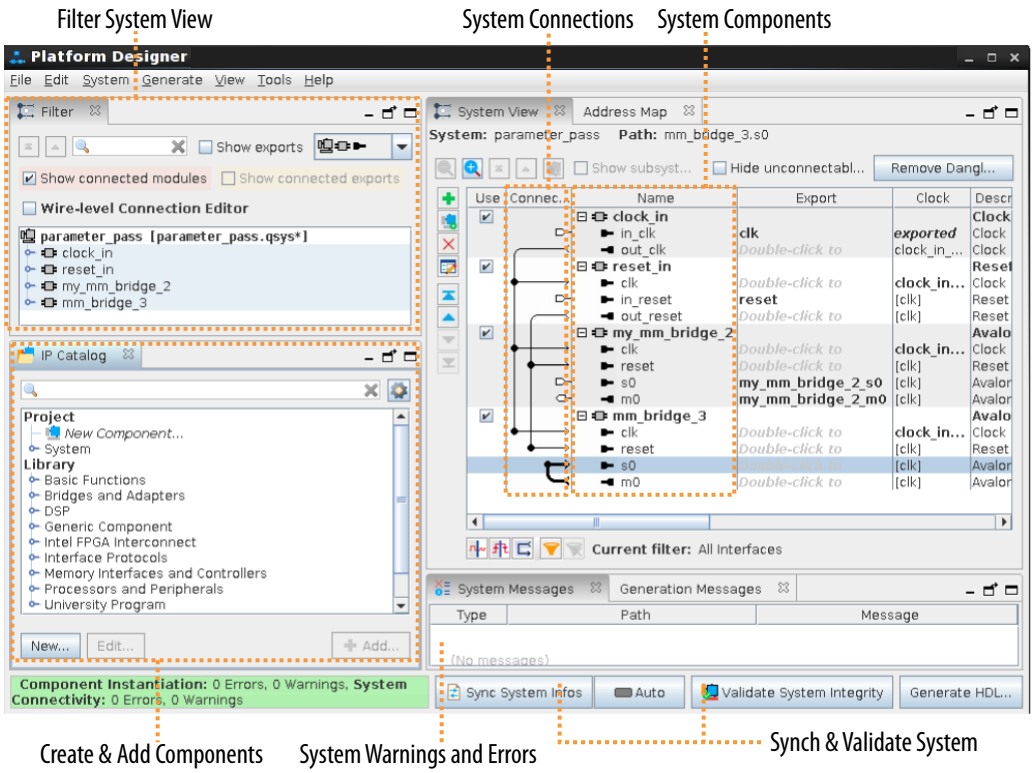
Platform Designer allows you to visualize all aspects of your system. By default, Platform Designer displays the contents of your system in the **System View** tab whenever you open a system. You can also access other tabs that allow you to view and modify various elements of the system.

When you select or edit an item in one Platform Designer tab, all other tabs update to reflect your selection or edit. For example, if you select the `cpu_0` in the **Hierarchy** tab, the **Parameters** tab immediately updates to display `cpu_0` parameters.

Click the View menu to interact with the elements of your system in various tabs.

- The **System View**, **Address Map**, **Exported Interfaces**, and **Details** tabs display in the central pane.
- By default, the **IP Catalog** and **Filter** tabs appear to the left of the **System View** tab.
- **Parameters**, **System Info**, and **Component Instantiation** tabs appear to the right of the **System View** tab when you open them.
- The **System Messages** and **Generation Messages** tabs display in the lower portion of Platform Designer.

Figure 19. Platform Designer GUI



The Platform Designer GUI is fully customizable. You can arrange and display Platform Designer GUI elements that you most commonly use, and then save and reuse useful GUI layouts.

1.5.1. Viewing the System Hierarchy

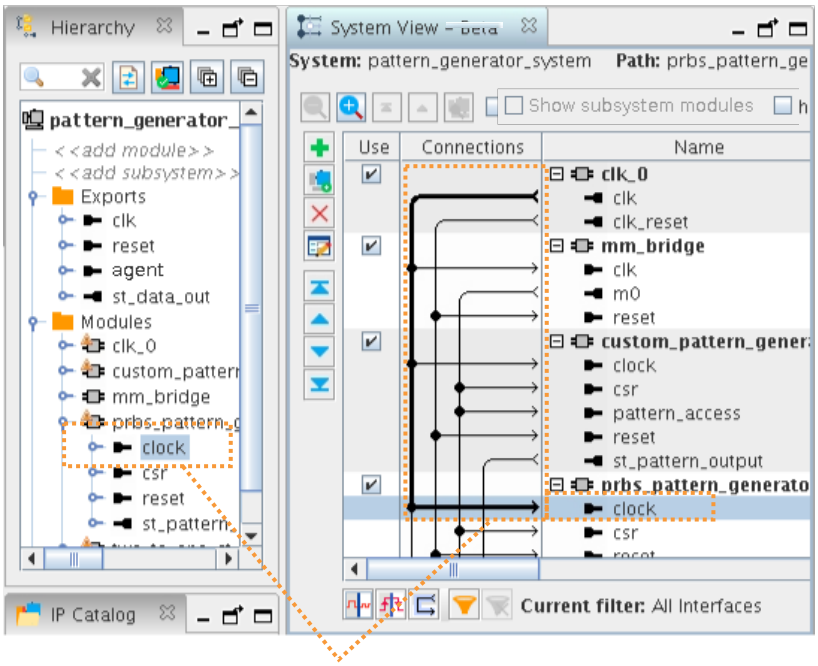
The **Hierarchy** tab hierarchically displays the modules, connections, and exported signals in the current system. You can expand and traverse though the system hierarchy, zoom in for detail, and locate to elements in other Platform Designer panes.

The **Hierarchy** tab provides the following information and functionality:

- Lists connections between components.
- Lists names of signals in exported interfaces.
- Right-click to connect, edit, add, remove, or duplicate elements in the hierarchy.
- Displays internal connections of Platform Designer subsystems that you include as IP components. By contrast, the **System View** tab displays only the exported interfaces of Platform Designer subsystems.

Click the **+** icon to expand any interface in the **Hierarchy** tab to view sub-components, associated elements, and signals for the interface. The **Hierarchy** tab displays a unique icon for each element type in the system. In the example below, the `clock` signal for the `prbs_pattern_generator` is selected in both the **System View** and **Hierarchy** tabs.

Figure 20. Expanding System View in the Hierarchy Tab



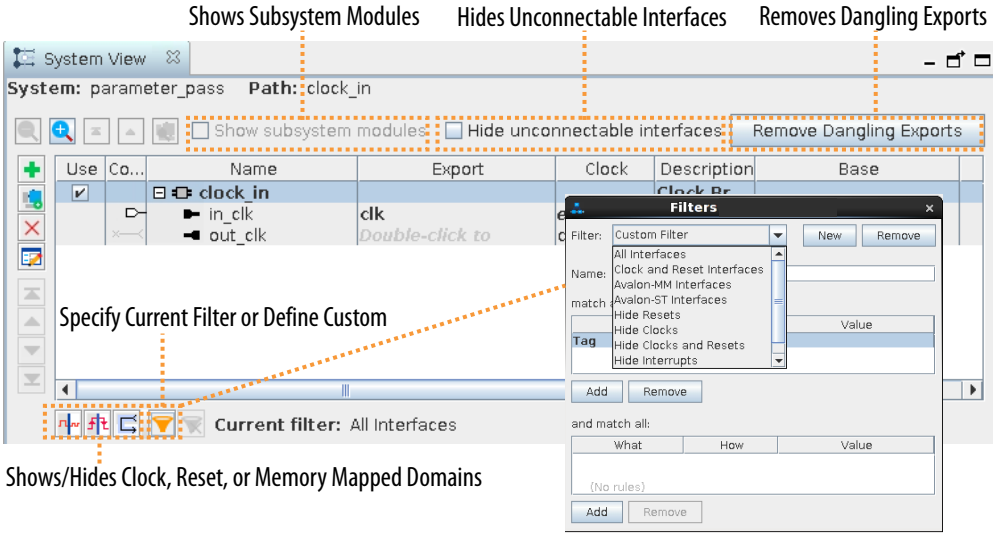
1.5.2. Filtering the System View

You can use the filtering controls in **System View** and the **Filter** tab to change the level of detail in the **System View**. You can filter for various component characteristics, such as component, interface type, or instance name. Filtering the System View allows you to simplify the display and focus only on the items you want.

Table 4. System View Tab Filtering Controls

Filter Control	Description
Filter Button	Defines the type of interfaces that the System View displays. The options are **Clock and Reset**, **Avalon Memory Mapped Interfaces**, **Avalon Streaming Interfaces**, **Hide Resets**, **Hide Clocks**, **Hide Clocks and Resets**, **Hide Interrupts**, or a **Custom Filter** that you define.
Show Memory Mapped domains in the system table button	Shows or hides all Avalon memory mapped domains present in the system in the **System View**.
Show reset domains in the system table	Shows or hides all reset domains in the system in the **System View**.
Show clock domains in the system table	Shows or hides all clock domains in the system in the **System View**.
Show connected modules	Shows or hides the modules that have connections in the **System View**.
Hide unconnectable interfaces	Shows or hides the modules not available for connection in the **System View**.
Remove Dangling Exports	Removes unconnected exported connections from the **System View**.

Figure 21. Filter Controls in System View Tab



The **Filter** tab offers other filtering controls that change the display of components in the **System View**. Select one or more components on the **Filter** tab (**View ➤ Filter**) to display only the selected component in the **System View** tab.

Figure 22. Filter Tab

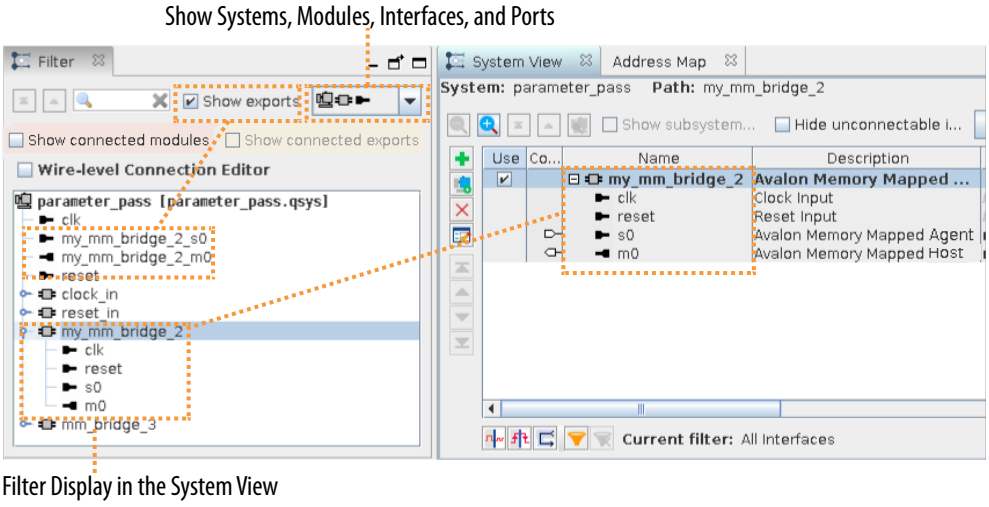


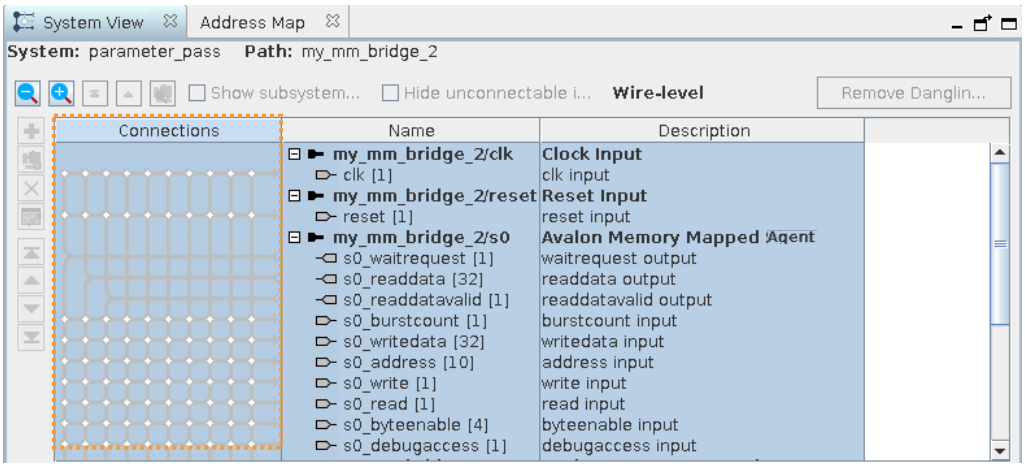
Table 5. Filter Tab Filtering Controls

Filter Control	Description
Search text field	In the **Filter** tab, displays the system, module, interface, and port names that match the text string you enter.
Show exports option	In the **Filter** tab, shows or hides the interfaces that you export.
Tree display list	In the **Filter** tab, specifies the level of detail to display. Show or hide all systems, modules, interfaces, and ports.

continued...

Filter Control	Description
Show connected modules option	In the **System View**, shows or hides the modules that have connections to the items that you select in the **Filter** tab tree.
Show connected exports	In the **System View**, shows or hides the exports that have connections to the items that you select in the **Filter** tab tree.
Wire-level Connection Editor	In the **System View**, shows or hides all wire-level connections in the Connections column.

Figure 23. Wire-level Connection Editor



1.5.3. Viewing Clock and Reset Domains

The Platform Designer **Clock Domains** and **Reset Domains** tabs list the clock and reset domains in the Platform Designer system, respectively.

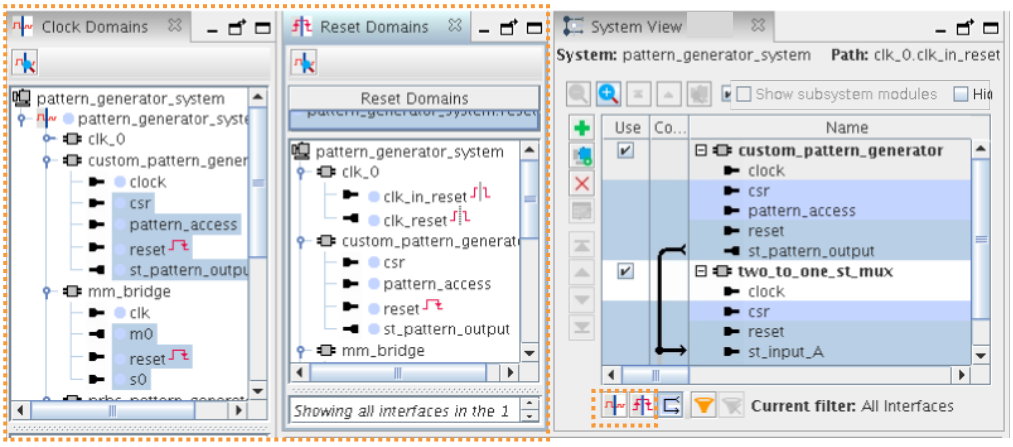
Click **View ➤ Clock Domains** or click **View ➤ Reset Domains** to display these tabs.

Platform Designer determines clock and reset domains by the associated clocks and resets. This information displays when you hover over interfaces in your system.

The **Clock Domains** and **Reset Domains** tabs also allow you to locate system performance bottlenecks. The tabs indicate connection points where Platform Designer automatically inserts clock-crossing adapters and reset synchronizers during system generation. View the following information on these tabs to create optimal connections between interfaces:

- The number of clock and reset domains in the system
- The interfaces and modules that each clock or reset domain contains
- The locations of clock or reset crossings
- The connection point of automatically inserted clock or reset adapters
- The proper location for manual insertion of a clock or reset adapter

Figure 24. Clock Domains, Reset Domains, and System View Tabs



1.5.3.1. Viewing Clock Domains in a System

You can filter the **System View** tab to display a single clock domain, or multiple clock domains. When you select an element in the **Clock Domains** tab, the corresponding selection appears highlighted in the **System View** tab.

Follow these steps to filter and highlight clock domains in the **System View**:

1. Click **View ➤ Clock Domains**.
2. Select any clock or reset domain in the list to view associated interfaces. The corresponding selection appears in the **System View** tab.
3. To highlight clock domains in the **System View** tab, click **Show clock domains in the system table** or at the bottom of the **System View** tab.

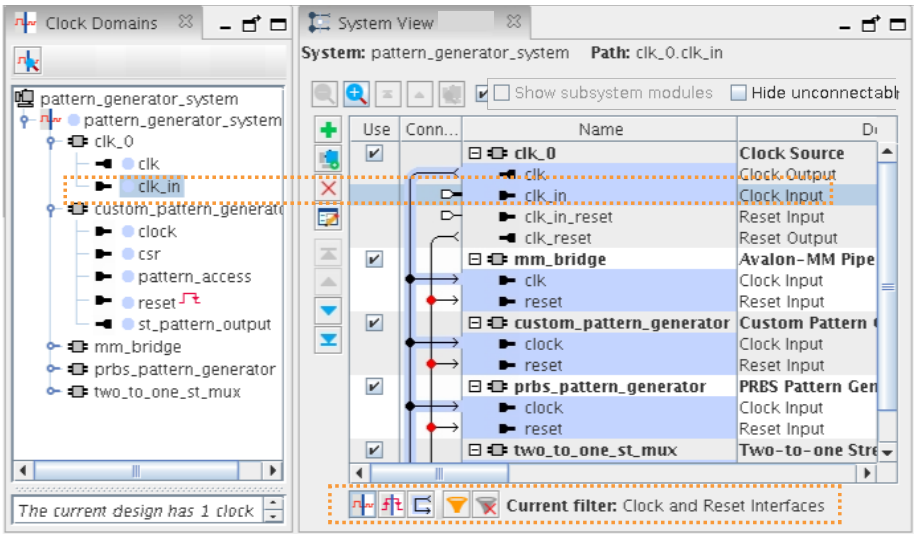
Figure 25. Shows Clock Domains in the System Table

4. To view a single clock domain, or multiple clock domains and their modules and connections, select the clock name or names in the **Clock Domains** tab. The modules for the selected clock domain or domains and connections highlight in the **System View** tab. Detailed information for the current selection appears in the clock domain details pane.

   Note: If a connection crosses a clock domain, the connection circle appears as a red dot in the **System View** tab

5. To view interfaces that cross clock domains, expand the **Clock Domain Crossings** icon in the **Clock Domains** tab, and select each element to view its details in the **System View** tab.

Figure 26. Selected Clock in Clock Domains and System View Tabs



Platform Designer lists the interfaces that cross clock domains under **Clock Domain Crossings**. As you click through the elements, detailed information appears in the clock domain details pane. Platform Designer also highlights the selection in the **System View** tab.

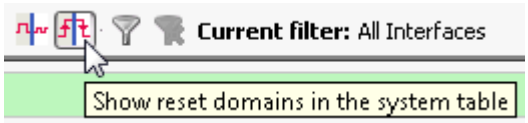
1.5.3.2. Viewing Reset Domains in a System

On the **Reset Domains** tab, you can filter the **System View** tab to display a single reset domain, or multiple reset domains. When you select an element in the **Reset Domains** tab, the corresponding selection appears in the **System View** tab.

Follow these steps to filter and highlight reset domains in the **System View**:

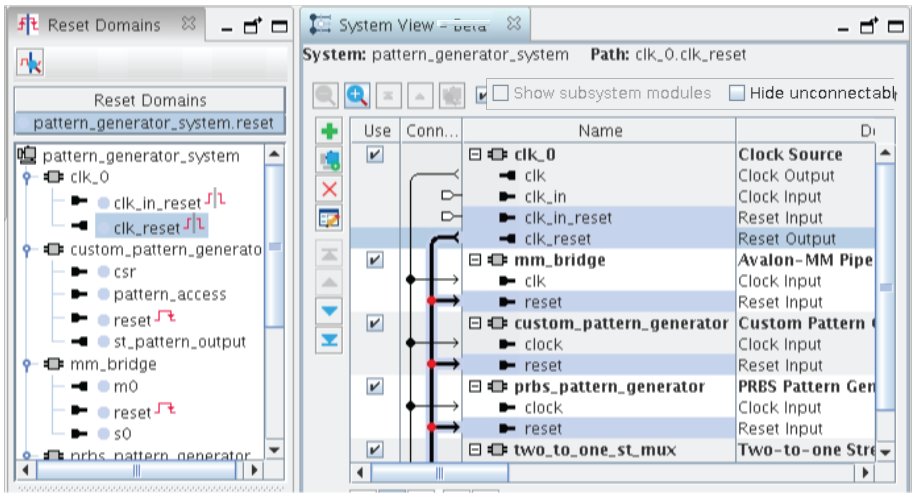
1. To open the **Reset Domains** tab, click **View ➤ Reset Domains**.
2. To show reset domains in the **System View** tab, click the **Show reset domains in the system table** icon in the **System View** tab.

Figure 27. Show Reset Domains in the System Table



3. To view a single reset domain, or multiple reset domains and their modules and connections, click the reset names in the **Reset Domain** tab.

Figure 28. Selected Reset Signal in Reset Domains and System View Tabs



Platform Designer displays your selection according to the following rules:

- When you select multiple reset domains, the **System View** tab shows interfaces and modules in both reset domains.
- When you select a single reset domain, the other reset domains are grayed out, unless the two domains have interfaces in common.
- Reset interfaces appear black when connected to multiple reset domains.
- Reset interfaces appear gray when they are not connected to all of the selected reset domains.
- If an interface is contained in multiple reset domains, the interface is grayed out.

Detailed information for your selection appears in the reset domain details pane. Red dots in the **Connections** column between reset sinks and sources indicate auto insertions by Platform Designer during system generation, for example, a reset synchronizer. Platform Designer decides when to display a red dot with the following protocol, and ends the decision process at first match.

- Multiple resets fan into a common sink.
- Reset inputs are associated with different clock domains.
- Reset inputs have different synchronicity.

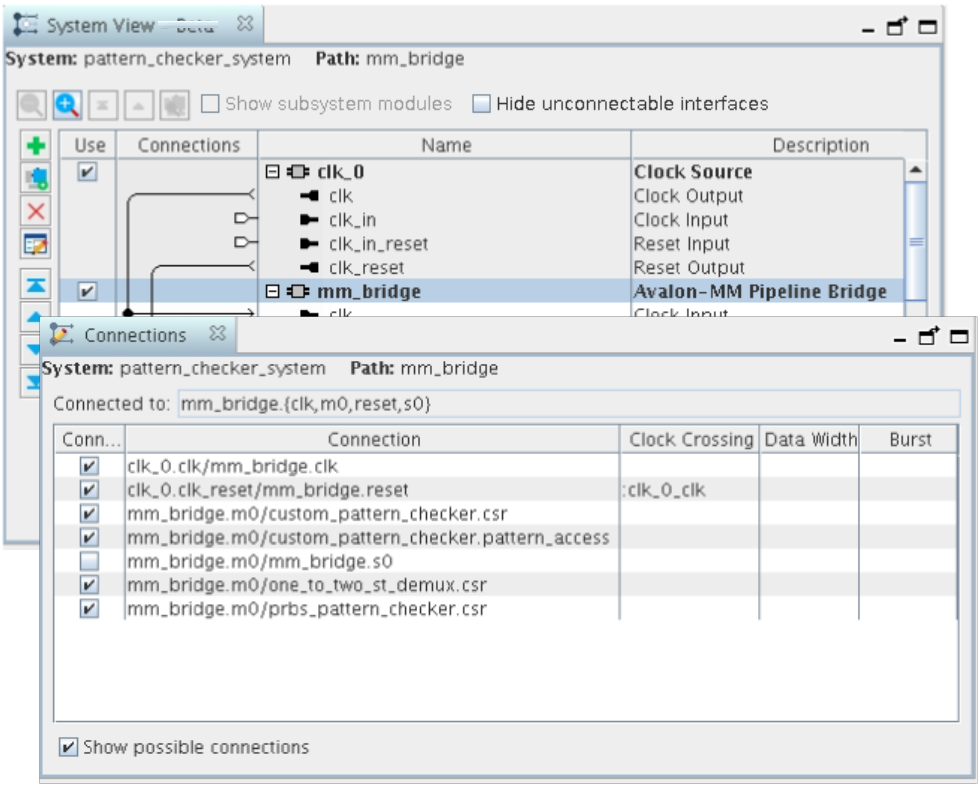
1.5.4. Viewing System Connections

The **Connections** tab allows you to connect or un-connect every connection in the Platform Designer system.

Click **View ➤ Connections** to display this tab.

If you connect or unconnect modules on the **Connections** tab, the connection immediately updates in the **System View** tab. You can also make connections in the **System View** tab directly.

Figure 29. Connections tabs in Platform Designer



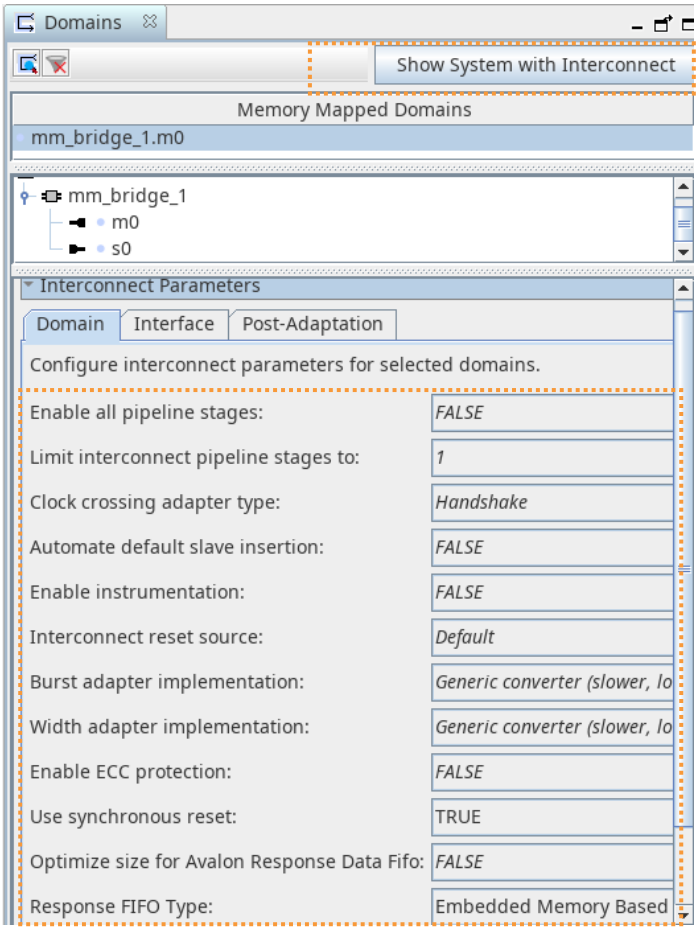
1.5.5. Viewing Avalon Memory-Mapped Domains in a System

The **Domains** tab displays a list of all the Avalon memory mapped domains in the system, allowing you to specify interconnect parameters. When you select a domain in the **Domains** tab, the corresponding selection highlights in the **System View** tab.

Click **View ➤ Domains** to display this tab.

- Filter the **System View** tab to display a single Avalon domain, or multiple domains. Further filter your view with selections in the **Filters** dialog box.
- To rename an Avalon memory-mapped domain, double-click the domain name. Detailed information for the current selection appears in the Avalon domain details pane.
- On the **Domain** tab, specify interconnect parameters, as Specifying Interconnect Parameters on page 71 describes.
- To enable and disable the highlighting of the Avalon domains in the **System View** tab, click the domain control tool at the bottom of the **System View** tab.

Figure 30. Domains Tab



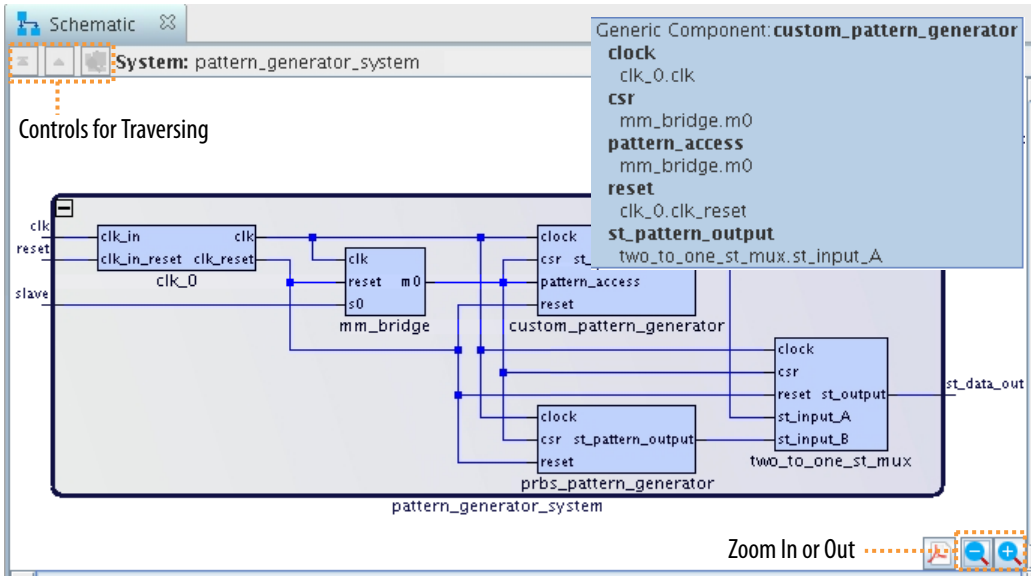
1.5.6. Viewing the System Schematic

The **Schematic** tab displays a schematic representation of the current Platform Designer system. You can zoom into a component or connection to view more details. You can use the image handles in the right panel to resize the schematic image.

Click **View ➤ Schematic** to display this tab.

If your selection is a subsystem, You can use the **Move to the top of the hierarchy** **Move up one level of hierarchy**, and **Drill into a subsystem to explore its contents** buttons to traverse the schematic of a hierarchical system.

Figure 31. Schematic Tab



Related Information

Editing a Subsystem on page 110

1.5.7. Customizing the Platform Designer Layout

You can arrange your workspace by dragging and dropping, and then grouping tabs in an order appropriate to your design development, or close or dock tabs that you are not using.

Dock tabs in the main frame as a group, or individually by clicking the tab control in the upper-right corner of the main frame. Tool tips on the upper-right corner of the tab describe possible workspace arrangements, for example, restoring or disconnecting a tab to or from your workspace.

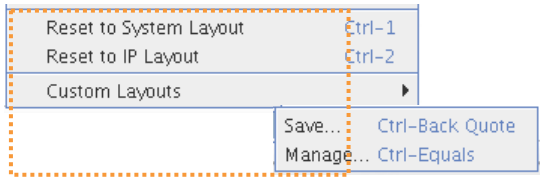
When you save your system, Platform Designer also saves the current workspace configuration. When you re-open a saved system, Platform Designer restores the last saved workspace.

The **Reset to System Layout** command on the View menu restores the workspace to its default configuration for Platform Designer system design. The **Reset to IP Layout** command restores the workspace to its default configuration for defining and generating single IP cores.

Follow these steps to customize and save the Platform Designer layout:

1. Click items on the View menu to display and then optionally dock the tabs. Rearrange the tabs to suit your preferences.
2. To save the current Platform Designer window configuration as a custom layout, click **View ➤ Custom Layouts ➤ Save**. Platform Designer saves your custom layout in your project directory, and adds the layout to the custom layouts list, and the `layouts.ini` file. The `layouts.ini` file determines the order of layouts in the list.

Figure 32. Platform Designer View Menu and Layouts



3. Use any of the following methods to revert to another layout:
   - To revert the layout to the default system design layout, click **View ➤ Reset to System Layout**. This layout displays the **System View**, **Address Map**, **Interconnect Requirements**, and **Messages** tabs in the main pane, and the **IP Catalog** and **Hierarchy** tabs along the left pane.
   - To revert the layout to the default system design layout, click **View ➤ Reset to IP Layout**. This layout displays the **Parameters** and **Messages** tabs in the main pane, and the **Details**, **Block Symbol**, and **Presets** tabs along the right pane.
   - To reset your Platform Designer window configuration to a previously saved layout, click **View ➤ Custom Layouts**, and then select the custom layout.
   - Press Ctrl+3 to quickly change the Platform Designer layout.
4. To manage your saved custom layouts, click **View ➤ Custom Layouts**. The **Manage Custom Layouts** dialog box opens and allows you to apply a variety of functions that facilitate custom layout management. For example, you can import or export a layout from or to a different directory.

1.5.8. Changing the Platform Designer Font

Click **Tools ➤ Options ➤ Fonts** to change the font name, style, and size in Platform Designer to suit your viewing preferences. A **Sample** pane displays a font example. Click **Finish** to apply the setting.

Note: Some GUI elements may require refresh (reopening) before font changes are visible. Platform Designer prompts you to restart Platform Designer after changing font settings so that font changes appear throughout.

You can click the **Reset Font** button to reset the Platform Designer font to the default value.

1.6. Adding IP Components to a System

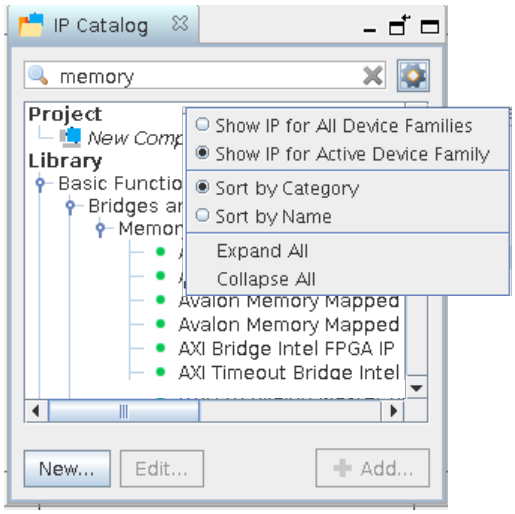
You can add Intel FPGA IP components to a system from the IP Catalog in Platform Designer. The IP Catalog launches a parameter editor for specifying options and generating the component's HDL. Your Platform Designer system can contain a single instance of an IP component, or multiple, individually parameterized variations of multiple or the same IP components.

When you first add Intel FPGA IP components to a system, Platform Designer automatically adds the IP as a generic component (except for HPS IP components). Generic components allow you to define only the interface and signal connections to the rest of the system, without immediately defining the HDL implementation.

To parameterize, and instantiate an IP component in a Platform Designer system:

1. Type some of the component's name in the IP Catalog search box to find the IP by name or category.

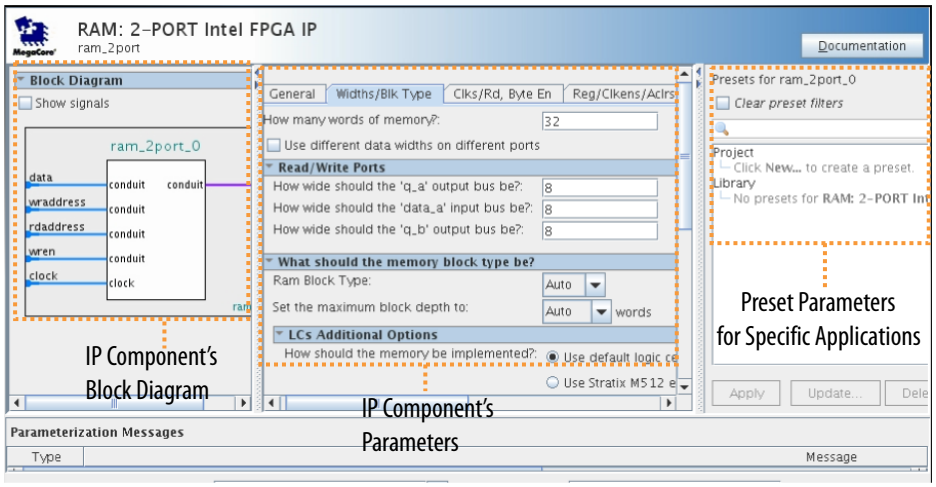
Figure 33. Platform Designer IP Catalog



2. Double-click any component to launch the parameter editor. The **Parameterization Messages** tab displays any parameterization errors.
3. After specifying parameters, click **Finish** to instantiate the component in the system. The IP component appears in the **System View** and **Component Instantiation** tabs. Platform Designer creates a corresponding `.ip` file for the IP component on instantiation, and stores the file in the <ip> folder in the project directory.

   Platform Designer instantiates a generic component in place of the actual IP core with a reference to the HDL entity name, module and interface assignments, compilation library, HDL ports, interfaces, and system-info parameters.

Figure 34. Parameter Editor



1.6.1. Modifying IP Parameters

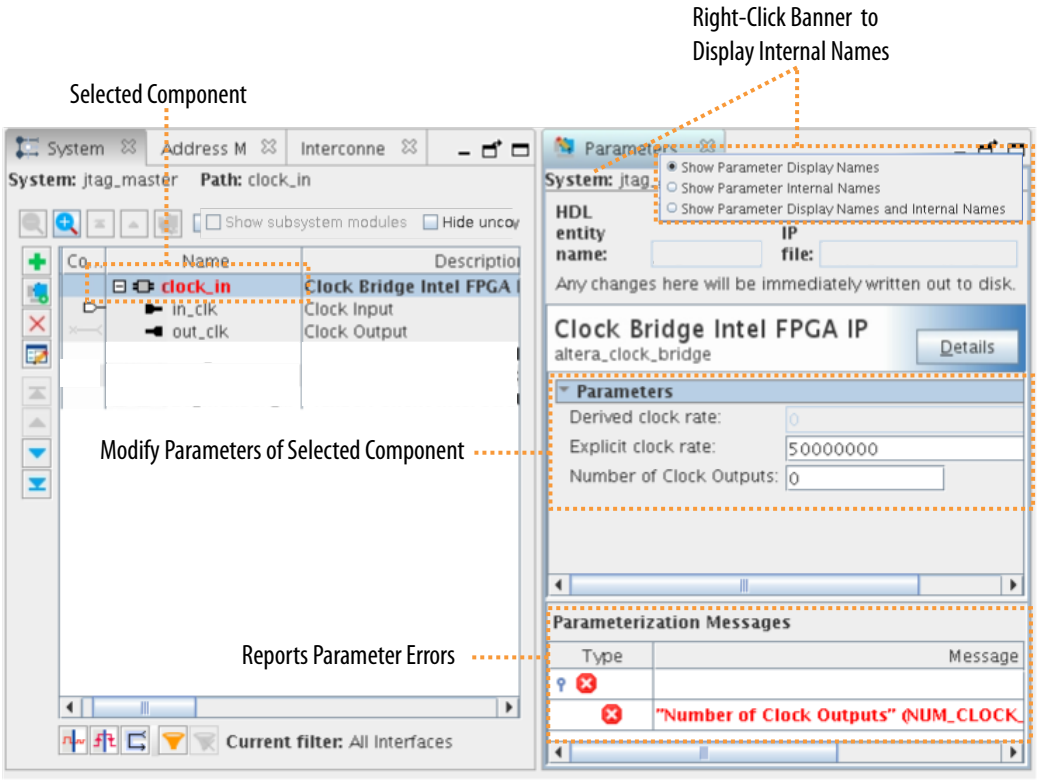
The **Parameters** tab allows you to view and edit the current parameter settings for IP components in your system.

To display a components parameters on the **Parameters** tab:

1. click **View ➤ Parameters**.
2. Select the component in the **System View** or **Hierarchy** tabs.

- **Parameters** field—adjust the parameters to align with your design requirements, including changing the name of the top-level instance.
- Component Banner—displays the hierarchical path for the component and internal names. Displays the HDL entity name and the IP file path for the selected IP component. Right-click in the banner to display internal parameter names for use with scripted flows.
- **Details**—displays links to detailed information about the component.
- **Parameterization Messages**—displays parameter warning and error messages about the IP component.

Figure 35. Platform Designer Parameters Tab



Changes that you make in the **Parameters** tab affect your entire system, and dynamically update other open tabs in Platform Designer. Any change that you make on the **Parameters** tab, automatically updates the corresponding `.ip` file that stores the component's parameterization.

If you create your own custom IP components, you can use the Hardware Component Description File (`_hw.tcl`) to specify configurable parameters.

Note: If you use the `ip-deploy` or `qsys-script` commands rather than the Platform Designer GUI, you must use internal parameter names with these parameters.

1.6.1.1. Viewing Component or Parameter Details

The **Details** tab provides information for a component or parameter that you select. Platform Designer updates the information in the **Details** tab as you select different components.

To view a component's details:

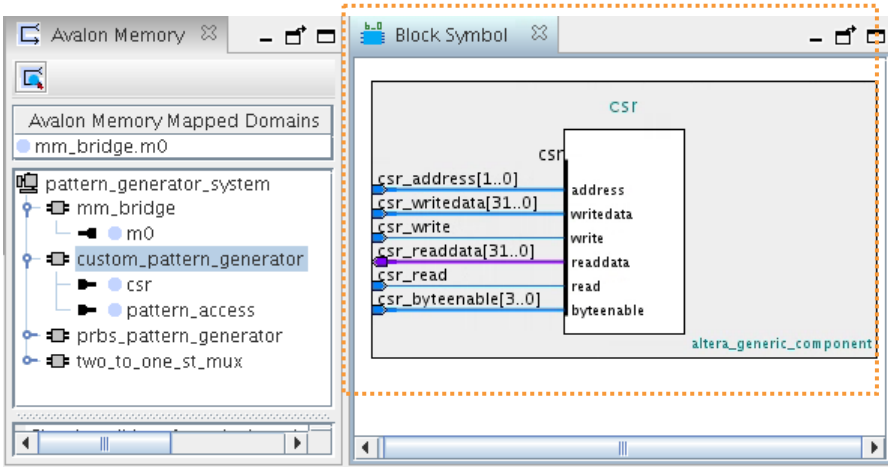
1. Click the parameters for a component in the parameter editor, Platform Designer displays the description of the parameter in the **Details** tab.
2. To return to the complete description for the component, click the header in the **Parameters** tab.

1.6.1.2. Viewing a Component's Block Symbol

The **Block Symbol** tab displays a symbolic representation of any component you select in the **Hierarchy** or **System View** tabs. The block symbol shows the component's port interfaces and signals. The **Show signals** option allows you to turn on or off signal graphics.

The **Block Symbol** tab appears by default in the parameter editor when you add a component to your system. When the **Block Symbol** tab is open in your workspace, it reflects changes that you make in other tabs.

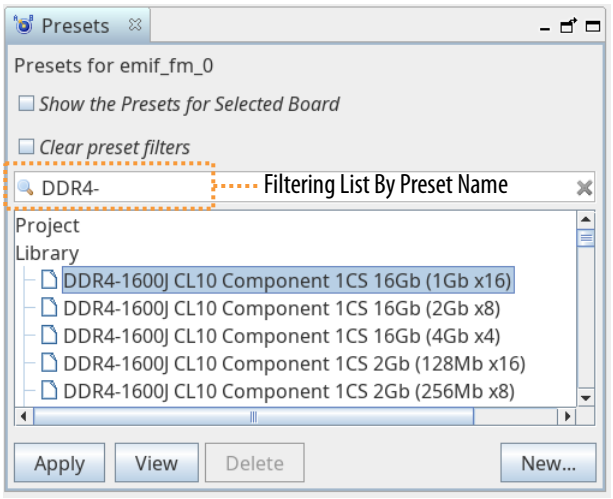
Figure 36. Block Symbol Tab



1.6.2. Applying Preset Parameters for Specific Applications

The **Preset** tab displays the names of available preset settings for an IP component. A preset is a specific collection of parameter settings that are appropriate for a specific protocol, application, or board. Double-click the preset name (or click **Apply**) to instantly apply the parameter values defined in the preset to the current IP instance.

Figure 37. Selecting Preset Parameters



1.6.2.1. Customizing IP Presets

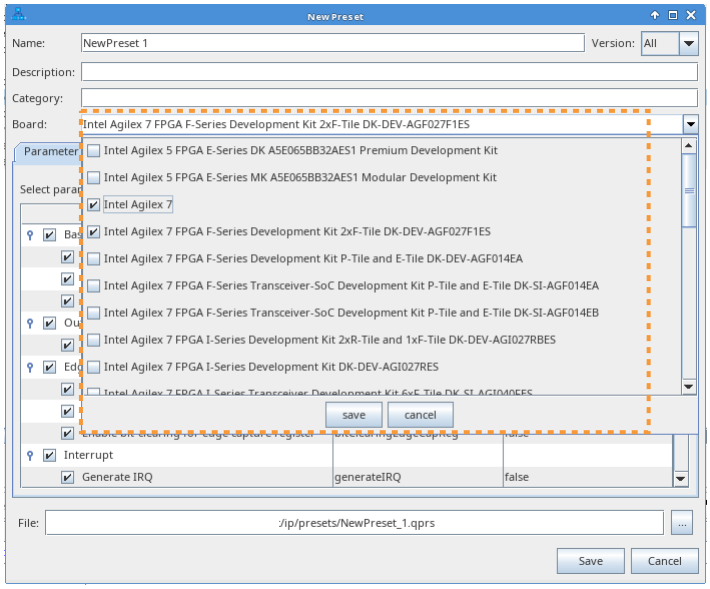
You can optionally define and save a custom set of parameter settings as an IP preset, and then apply the preset whenever you add an instance of the IP component to any system.

Follow these steps to save a custom IP preset:

1. In IP Catalog, double-click any component to launch the parameter editor.
2. To search for a specific preset to base initial settings, type a partial preset name in the search box.
3. In the **Presets** tab, click **New** to specify the **Preset name** and **Preset description**.
4. In the **Board** dropdown, specify the target board. The **Default** setting specifies the current board as the target board for this preset.

   Note: You can specify multiple boards for a preset, provided that the preset parameters and assignments are applicable to all boards in the preset.
5. Under **Select parameters to include in the preset**, enable or disable the parameters you want to include in the preset.
6. Specify the path for the **Preset file** that preserves the collection of parameter settings. The location of the new `.qprs` preset file is added to the IP search path automatically.

Figure 38. Create New Preset

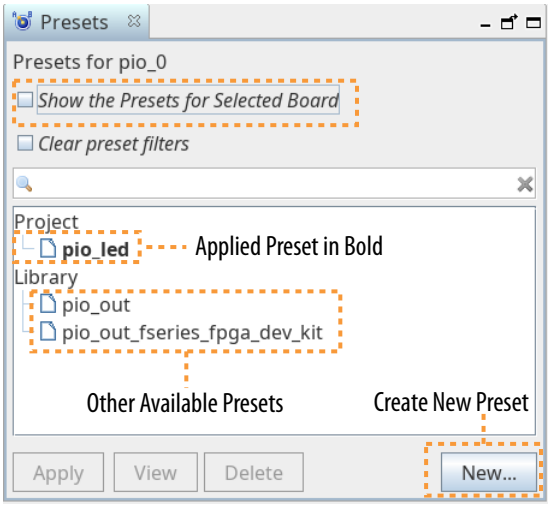


7. Click **Save**.
8. To apply the preset to an IP component, click **Apply**. Preset parameter values that match the current parameter settings appear in bold.

1.6.3. Creating IP Presets Targeting Specific Boards

Use the **Presets** tab in Platform Designer to define a custom group of preset parameter settings and pin assignments appropriate for the target board.

Figure 39. Presets Tab Displays Available IP Presets



You can apply the presets to IP, view and delete presets, and filter presets by board name in the **Presets** tab. If you click on any IP in the **System View**, the **Presets** tab displays the associated presets for each board. Turn off **Show the Presets for**

Selected Board to display all available presets for the IP. Turn off this option to view and apply preset files that do not contain board and pin information located under the **Library**.

Note: You cannot edit a preset in the **Preset** tab. Rather than editing an existing preset, create a new preset.

When you define an IP preset in the **Presets** tab, an underlying `.qprs` file stores the following information about the preset:

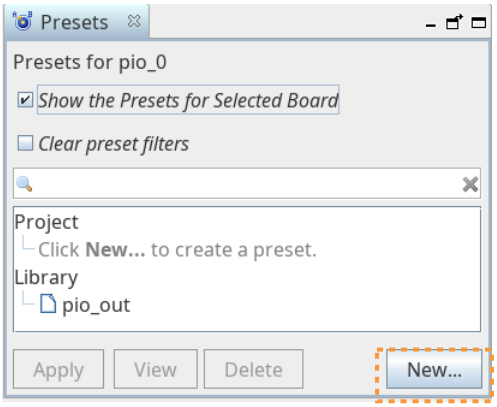
- Preset name
- Preset description
- Preset category
- Supported board
- Parameter settings
- Pin assignments

1.6.3.1. Creating IP Presets

To create an IP preset with appropriate parameters and pin assignments for the target board, follow these steps:

1. In the Platform Designer **System View**, select the IP that you want to create presets for and view the **Presets** tab (**View > Presets**).
2. In the **Presets** tab click the **New** button to define a new IP preset file for the IP. The **New Preset** dialog box appears.

Figure 40. Presets Tab



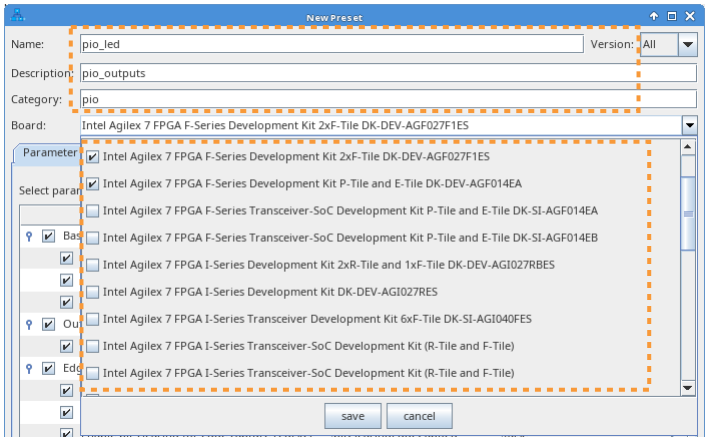
3. Enter a **Preset name**, **Preset description**, and **Category**.

Figure 41. New Preset Dialog Box

4. In the **Board** list, specify the target board. The **Default** setting specifies the current board as the target board for this preset.

   Note: You can specify multiple boards for a preset, provided that the preset parameters and assignments are applicable to all boards in the preset.

Figure 42. Create New Preset



5. On the **Parameter Settings** tab, enable or disable the appropriate parameters for your target board.
6. To specify pin location and I/O standard assignments for the preset, click the **Pin Assignments** tab.
7. For the exported interfaces, turn on the **external_connection** checkbox and enter the exported signal name in the **Exported Name** cell.

   Note: You can change the interface and signal names by double-clicking cells under **Exported Name**. Alternatively, you can type the pin locations and I/O standard details in the cell without using the **Board** dropdown.
8. Select the appropriate **Pin Location** and **IO Standard** for each exported signal. Refer to Defining Preset Pin Assignments.
9. By default, the **Preset file** setting suggests a `.qprs` file name based on the **Preset name**.
10. Click **Save**. The new IP preset appears in the **Presets** tab. Manage presets in the **Presets** tab, as Viewing, Applying, and Deleting IP Presets on page 50 describes.

Figure 43. Pin Assignments in New Preset Dialog Box

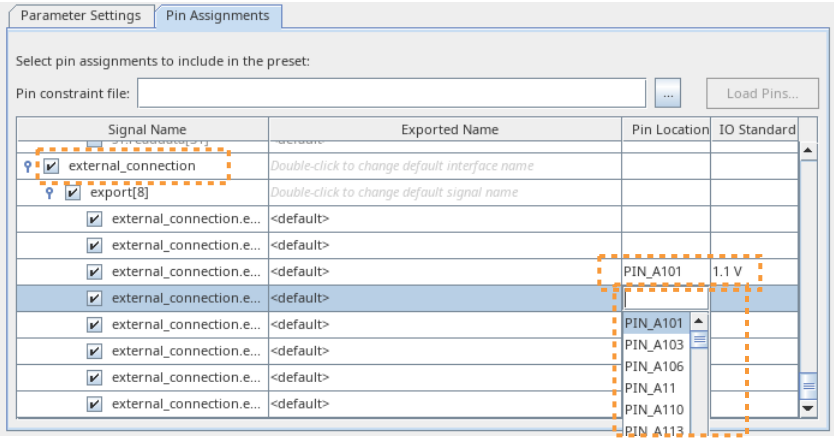
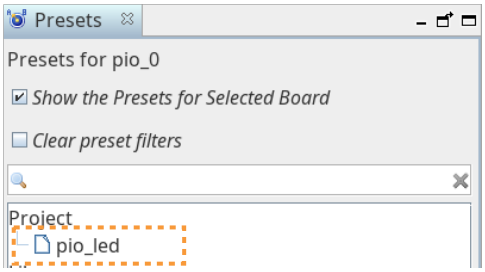


Figure 44. New pio_led Preset Appears in Presets Tab



1.6.3.2. Defining Preset Pin Assignments

You can define pin assignments that are included as part of an IP preset. When you apply the IP preset to an IP instance, the pin assignments export during the IP or system's HDL generation.

You define preset pin assignments in the **Pin Assignments** tab of the **New Preset** dialog box, or in a Pin Assignments File (`.tcl`) that you create.

1.6.3.2.1. Defining Preset Pin Assignments in Pin Assignments Tab

The **Pin Assignments** tab allows you to specify the **Exported Name** of the signals, to select the appropriate **Pin Location,** and to select the appropriate **IO Standard** for the target board.

By default, the **Exported Name** name takes the form of:

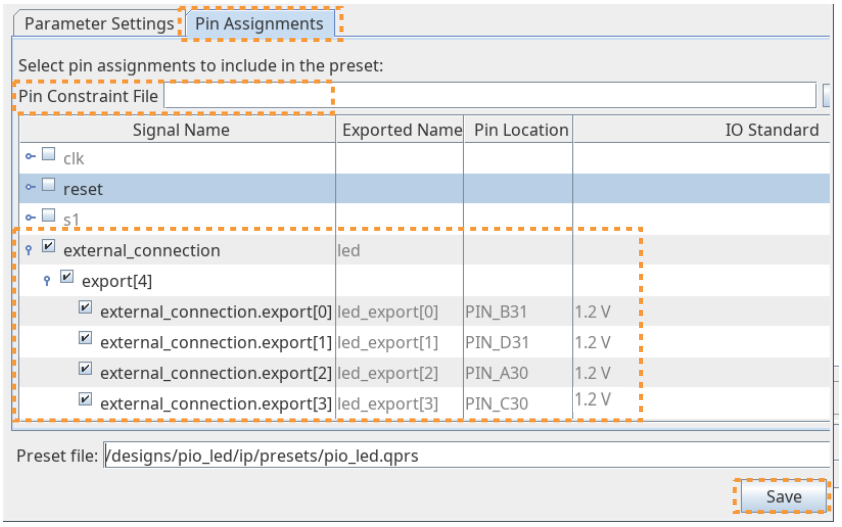
```
module_name + interface_name + pin_role
```

For example:

```
pio0_external_connection_export[0]
```

You can change the **Exported Name** by double-clicking on the **Exported Name** for the interface and typing a new name. All of the signals of the interface then update automatically to reflect the name you specify.

Figure 45. Pin Assignments Tab

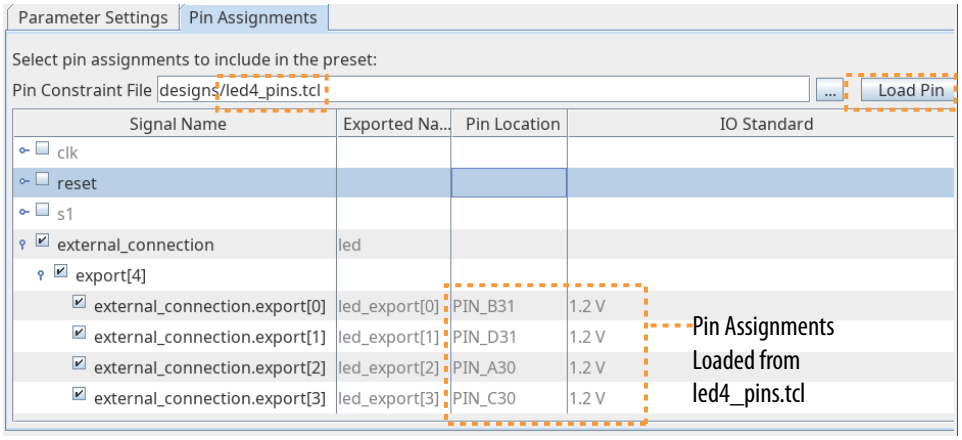


For example, typing `led` for the **external_connection** interface updates the signals of the interface to `led_export[n]`. The **external_connection** is the interface name, and **external_connection_export(0)** is the signal name.

1.6.3.2.2. Defining Preset Pin Assignments in a Pin File

Alternatively, you can specify the pin assignments in a Pin Constraints File (`.tcl`), which can be more efficient for projects with many ports. You specify this `.tcl` file as the **Pin Constraint File** on the **Pin Assignments** tab, and then click **Load Pin**. The **Pin Location** and **IO Standard** update per the loaded pin assignments.

Figure 46. Loading Pin Assignments from Tcl File



The following shows the contents of an example Pin Constraints File (`.tcl`):

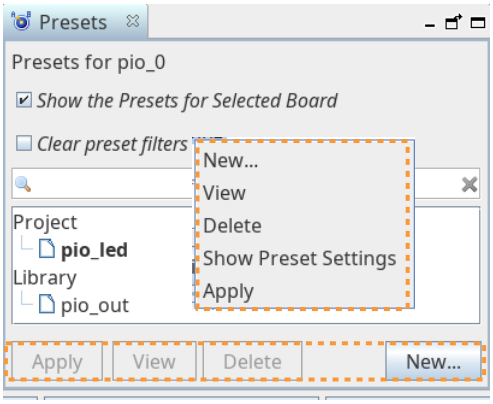
```
set_instance_assignment -to "led_export[0]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[0]" "PIN_B31"
set_instance_assignment -to "led_export[1]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[1]" "PIN_D31"
set_instance_assignment -to "led_export[2]" -name IO_STANDARD "1.2 V"
```

```
set_location_assignment -to "led_export[2]" "PIN_A30"
set_instance_assignment -to "led_export[3]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[3]" "PIN_C30"
```

1.6.3.3. Viewing, Applying, and Deleting IP Presets

You can view the properties of a preset, apply a preset, or delete any existing preset in the **Presets** tab.

Figure 47. View, Apply, and Delete Presets in Presets Tab

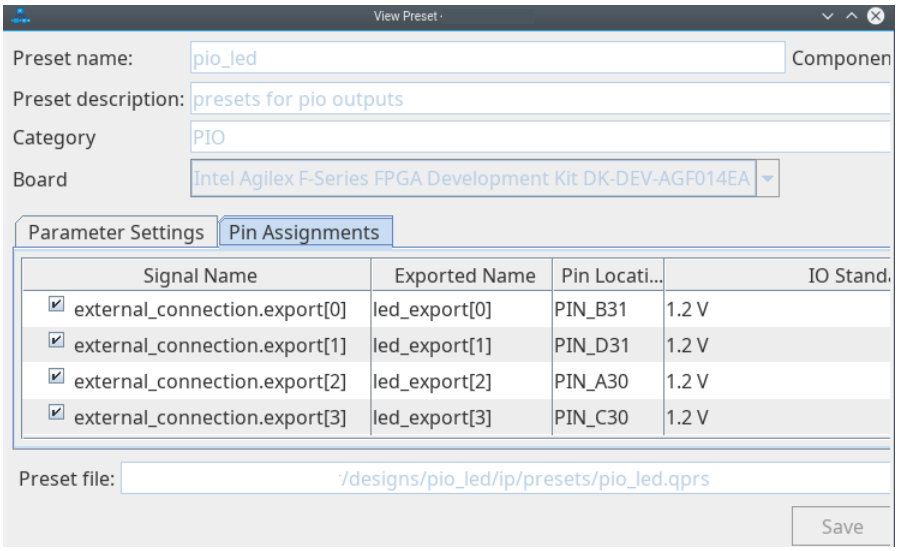


Note: Right-click a preset to access the same **View**, **Apply**, and **Delete** preset functions in the context menu.

Viewing Presets

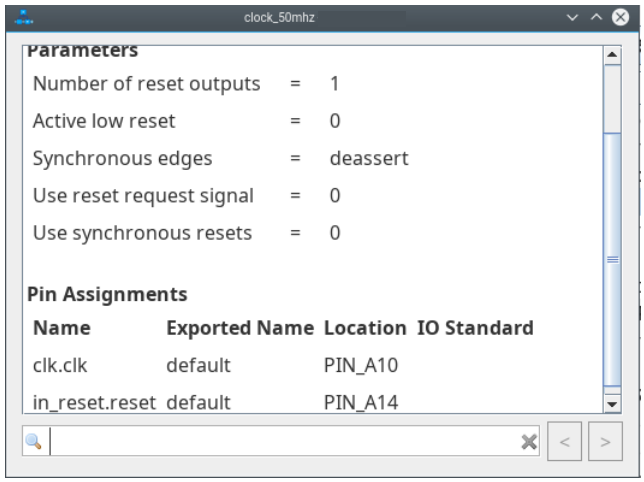
Click the **View** button to show the preset properties in the read-only **Update Preset** dialog box.

Figure 48. View Button Opens View Preset Dialog Box



Right-click a preset and click **Show Preset Settings** to view a searchable report of the preset settings.

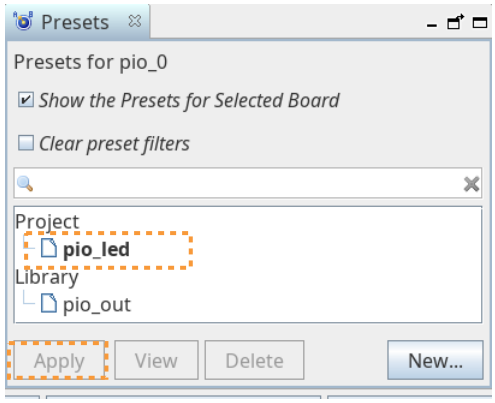
Figure 49. Show Preset Settings Searchable Report



Applying Presets to IP Instances

Click the **Apply** button (or double-click) to apply the IP preset to the currently selected IP. Applied presets appear in bold text.

Figure 50. Applied Presets Appear in Bold Text



Deleting Presets from the System

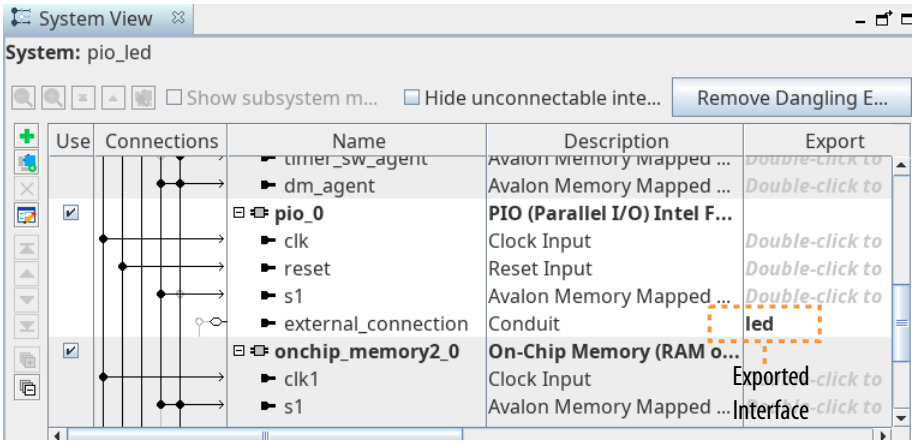
Click the **Delete** button to delete the current preset from the Platform Designer system.

1.6.3.4. Auto-Exporting IP Preset Interfaces and Pins

Once you apply a preset with pin assignments to an IP instance, Platform Designer automatically exports the preset interfaces that have pin constraints to external systems.

Note: If an IP with presets is not within the top-level Platform Designer system, you must export those interfaces as well.

Figure 51. Auto-Exported LED Interface IP Preset



After you click **Generate HDL** for the IP or system with pin assignment presets, Platform Designer automatically exports the pin assignments to the system `.qip` file for subsequent design compilation in the Quartus Prime software.

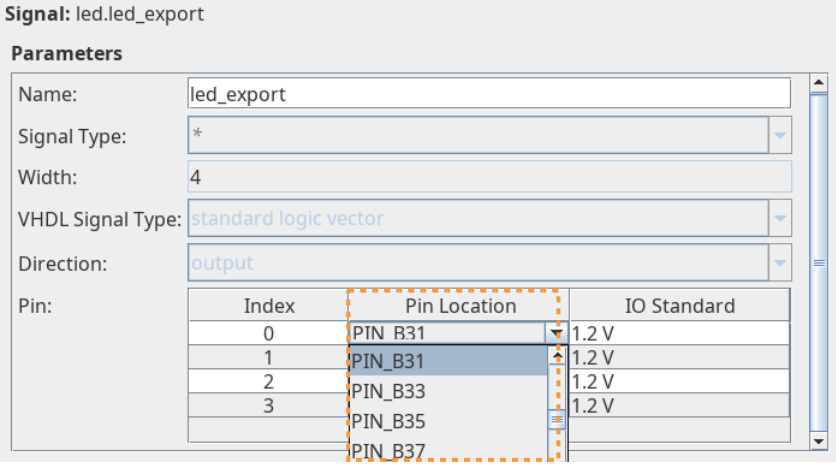
The following shows an example of the pin assignment Tcl commands exported to the system `.qip` file:

```
set_instance_assignment -to "led_export[0]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[0]" "PIN_B31"
set_instance_assignment -to "led_export[1]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[1]" "PIN_D31"
set_instance_assignment -to "led_export[2]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[2]" "PIN_A30"
set_instance_assignment -to "led_export[3]" -name IO_STANDARD "1.2 V"
set_location_assignment -to "led_export[3]" "PIN_C30"
```

1.6.3.5. Editing Pin Assignments for Presets

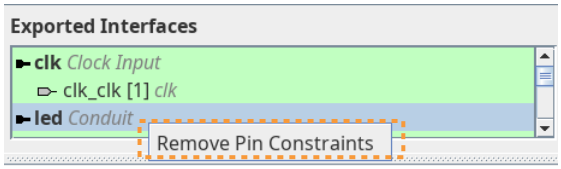
After you apply presets to IP, you can later view, edit, and remove those pin assignments for selected signals in the **Exported Interfaces** tab in Platform Designer. This action does not change the preset, but only changes the pin assignments of the exported system signals. To edit pin assignments for an existing IP preset, follow these steps:

Figure 52. Changing Preset Pin Location Assignments



1. In Platform Designer, click **View ➤ Exported Interfaces**. The **Exported Interfaces** tab displays the exported interfaces in the system.
2. Under **Exported Interfaces**, select a specific exported signal. The **Parameters** field displays the signal and editable **Pin** assignment information. The interface **Name** and signal name match the `exported_interface_name` and `exported_signal_name` values in the preset `.qprs` file.
3. To change the pin location or I/O standard for the signal, select a different **Pin Location** or **IO Standard** in the **Pin** field.
4. To remove any pin constraint, right-click the signal interface name under **Exported Interfaces**, and then click **Remove Current Pin Constraints**.

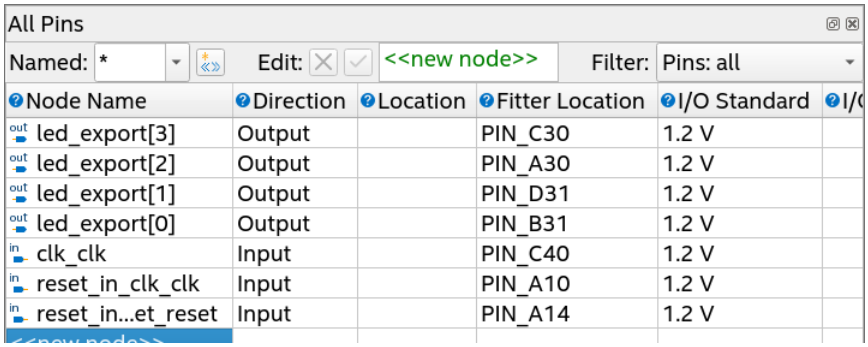
Figure 53. Remove Current Pin Constraints



To change only the exported port name, you can do so in the **System View** tab. You must still make detail changes in the **Exported Interfaces** tab.

5. After design compilation, review the exported preset pin assignments in the Quartus Prime software Pin Planner.

Figure 54. Exported Interface Pins in Pin Planner After Design Compilation

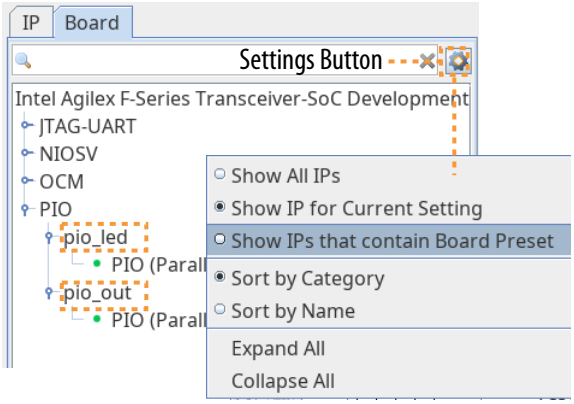


1.6.3.6. Viewing IP Presets In Board Catalog

You can easily view all presets defined for a target board in the Board Catalog in Platform Designer. The Board Catalog lists the name of each board defined, followed by a hierarchical list of the associated components, presets, and IP of the board.

You can use the search field and the adjacent Settings button to filter the list of list of IP to match your needs:

Figure 55. Viewing pio_led and pio_out Presets in Board Catalog



- Click the Settings button, and then select **Show IPs that contain Board Preset**. The Board Catalog displays only the IPs that have presets for the target board. Under the IP name, the available presets for that IP appear.

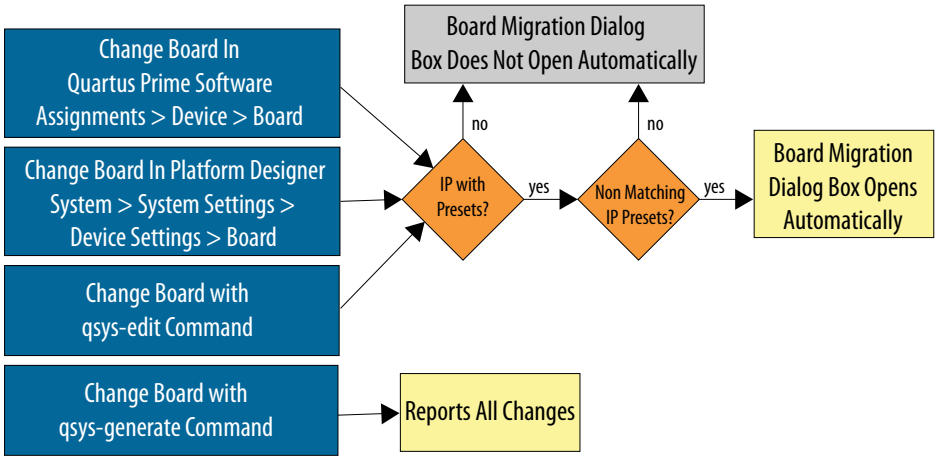
1.6.4. Applying Presets After Migrating a Board

You can save time by applying available presets when you migrate your design to a different board. The **Board Migration** dialog box helps you to quickly apply appropriate presets to each IP in the Platform Designer system to ensure consistency and accuracy of the system on the migrated board, even for hierarchical systems.

The **Board Migration** dialog box allows you to readily view a comparison of the parameters and pin assignments in the current system, versus the selected presets.

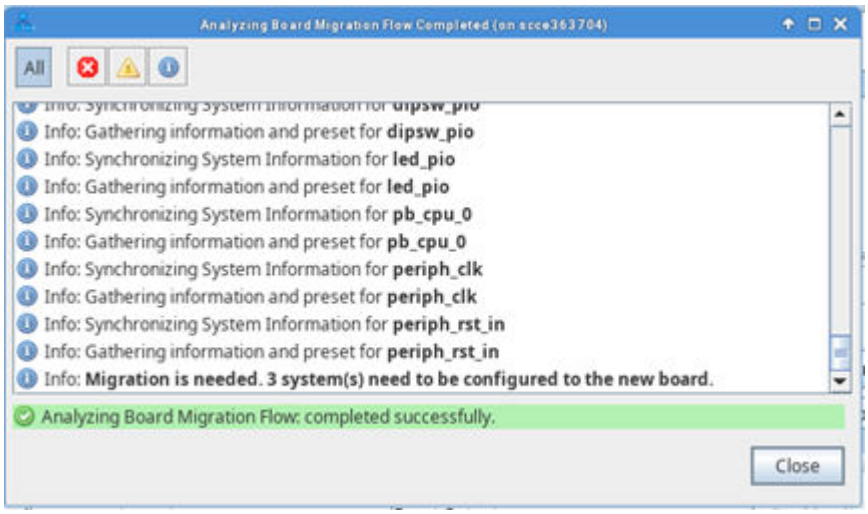
You can access the **Board Migration** dialog box using any of the following methods:

Figure 56. Opening the Board Migration Dialog Box



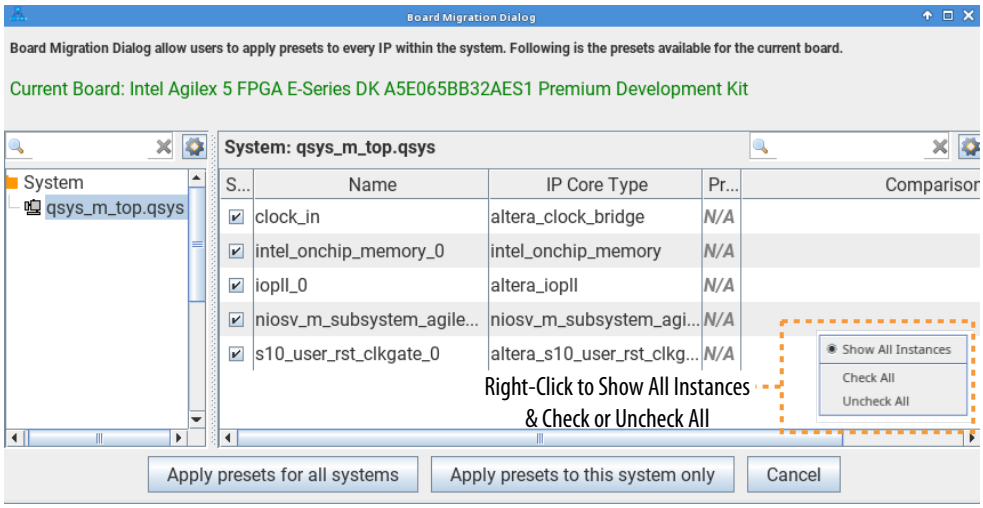
1. The **Board Migration** dialog box opens automatically using any of the following methods:
   - Click **Assignments ➤ Device ➤ Board** tab in the Quartus Prime Pro Edition software.
   - Click **System ➤ System Settings ➤ Device Settings ➤ Board** in Platform Designer.
   - Use the `--board` option with the qsys-edit command.
2. Upon completing step 1, the **Analyzing Board Migration** window appears showing the progress of system analysis and synchronization for the new target board. Analysis messages eventually indicate either that **Migration is needed** or **No migration is needed** to align with the target board. Click the **Close** button when system migration analysis is compete.

Figure 57. Analyzing Board Migration Window



If migration is needed, the **Board Migration** dialog box opens automatically displaying the systems and subsystems in the design you are migrating.

Figure 58. Board Migration Dialog Box



3. By default, the **Board Migration** dialog box displays only IP that do not have presets matching with the current configuration. To view all IP or systems, right-click in the dialog box and then click **Show All Instances**.

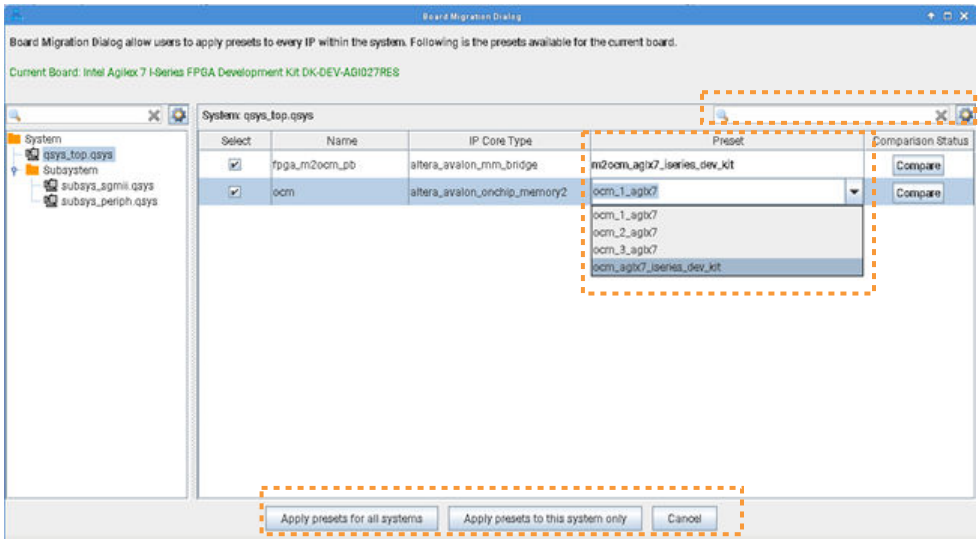
Note: Using the `--board` option with `qsys-generate` command does not launch the Board Migration dialog box. Rather, an information message appears suggesting that you to launch Platform Designer and use the **Board Migration** dialog box to determine the presets that apply to the new targeted board. The Board Migration dialog box does not open for migration of designs containing IP subsystems.

1.6.4.1. Comparing and Applying IP Presets

Follow these steps to compare the parameter and pin assignment differences of your current board configuration with your chosen board IP preset in the **Board Migration** dialog box and apply the desired presets:

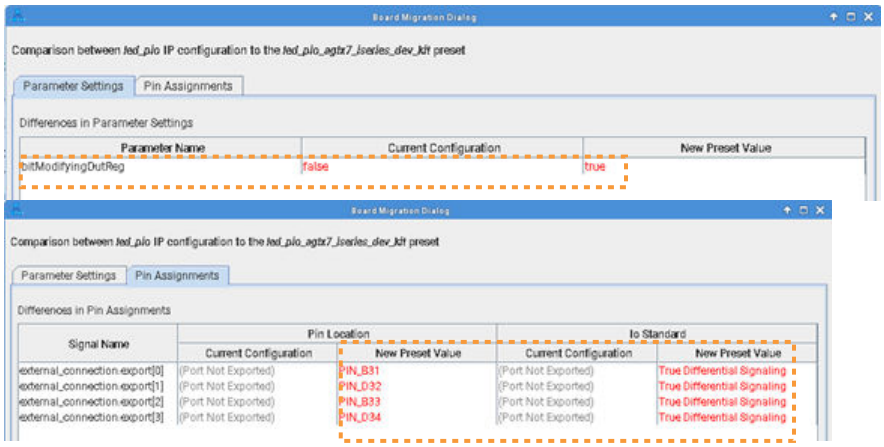
1. In the **Preset** column, type or select the preset name for the IP that you want to compare against the current configuration. If you have multiple IP or systems, type the IP or system name in the search bar to narrow the non-case-sensitive search.
2. To begin comparison, click the **Compare** button. Following comparison, the **Parameter Settings** and **Pin Assignments** tabs show the comparison results (differences) for parameter settings and pin assignments, respectively.

Figure 59. Setup Comparisons



IP with matching configurations and presets display **Matched** in green under **Comparison Status**. IP with no presets available display **N/A** in the **Preset** column.

Figure 60. Pin Assignment and Parameter Setting Comparison Results



3. When the presets match the board configuration as desired, click either the **Apply Presets to This System Only** button or the **Apply Presents for All Systems** button. The presets apply to your system according to your specifications.

Figure 61. IP with No Presets Available Display N/A

1.6.5. Adding Third-Party IP Components

You can add third-party IP components created by Intel partners to your Platform Designer system. Third-party partner IP components have interfaces that Platform Designer supports, such as Avalon memory mapped or AMBA AXI. Third-party partner IP components can also include timing and placement constraints, software drivers, simulation models, and reference designs.

To locate supported third-party IP components on the Intel web page, follow these steps:

1. From the Intel website, navigate to the *Find IP* page, and then click Find IP on the tool.
2. Use the **Search** box and the **End Market**, **Technology**, **Devices** or **Provider** filters to locate the IP that you want to use.
3. Click **Enter**.
4. Sort the table of results for the **Platform Designer Compliant** column. You cannot use non-compliant components in Platform Designer.
5. Click the IP name to view information, request evaluation, or request download.
6. After you download the IP files, add the IP location to the IP search path to add the IP to IP Catalog, as IP Search Path Recursive Search on page 58 describes.

Related Information

Find Intel FPGA and Partner IP

1.6.5.1. IP Search Path Recursive Search

The Quartus Prime software automatically searches and identifies IP components in the IP search path. The search is recursive for some directories, and only to a specific depth for others. During a recursive descent search, whenever search finds a `_hw.tcl` or `.ipx` file, search does not descend further.

In the following list of search locations, ** indicates a recursive descent.

Table 6. IP Search Locations

Location	Description
`PROJECT_DIR/*`	Finds IP components and index files in the Quartus Prime project directory.
`PROJECT_DIR/ip/**/*`	Finds IP components and index files in any subdirectory of the `/ip` subdirectory of the Quartus Prime project directory.

1.6.5.1.1. IP Search Path Precedence

If the Quartus Prime software recognizes two IP cores with the same name, the following search path precedence rules determine the resolution of files:

1. Project directory.
2. Project database directory.
3. Project IP search path specified in **IP Search Locations**, or with the SEARCH_PATH assignment for the current project revision.
4. Global IP search path specified in **IP Search Locations**, or with the SEARCH_PATH assignment in the `quartus2.ini` file.
5. Quartus software libraries directory, such as <*Quartus Installation*>`\libraries`.

1.6.5.1.2. IP Component Description Files

The Quartus Prime software identifies parameterizable IP components in the IP search path for the following files:

- Component Description File (`_hw.tcl`)—defines a single IP core.
- IP Index File (`.ipx`)—each `.ipx` file indexes a collection of available IP cores. This file specifies the relative path of directories to search for IP cores. In general, `.ipx` files facilitate faster searches.

1.6.5.2. Defining the IP Search Path with Index Files

You can create an IP Index File (`.ipx`) to specify a path that Platform Designer searches for IP components.

You can optionally specify the search path in a `user_components.ipx` file, in addition to **Tools ➤ Options ➤ IP Catalog Search Locations**. The `user_components.ipx` file allows you to add locations independent of the default search path.

A <*path*> element in a `.ipx` file specifies a directory where Platform Designer can search for IP components. A <*component*> entry specifies the path to a single component. <*path*> elements allow wildcards in definitions. An asterisk matches any file name. If you use an asterisk as a directory name, it matches any number of subdirectories.

Example 1. Path Element in an .ipx File

```
<library>
        <path path="…<user directory>" />
        <path path="…<user directory>" />
        …
        <component … file="…<user directory>" />
        …
</library>
```

A <*component*> element in an `.ipx` file contains several attributes to define a component. If you provide the required details for each component in an `.ipx` file, the startup time for Platform Designer is less than if Platform Designer must discover the files in a directory.

Example 2. Component Element in an .ipx File

The example shows two *<component>* elements. Note that the paths for file names are specified relative to the `.ipx` file.

```
<library>
  <component
    name="A Platform Designer Component"
    displayName="Platform Designer FIR Filter Component"
    version="2.1"
    file="./components/qsys_filters/fir_hw.tcl"
   />
  <component
    name="rgb2cmyk_component"
    displayName="RGB2CMYK Converter(Color Conversion Category!)"
    version="0.9"
    file="./components/qsys_converters/color/rgb2cmyk_hw.tcl"
   />
</library>
```

Note: You can verify that IP components are available with the `ip-catalog` command.

Related Information

Create an .ipx File with ip-make-ipx on page 449

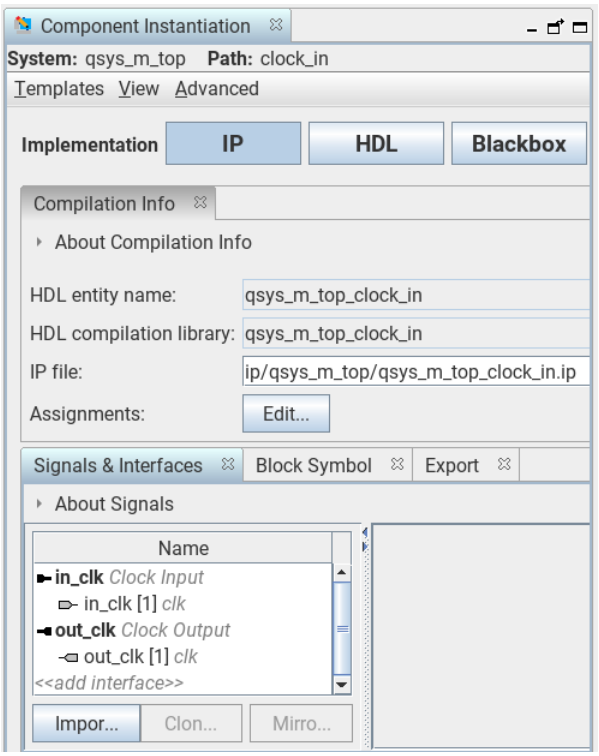
1.6.6. Specifying IP Component Instantiation Options

When you instantiate an Intel FPGA IP component in a system, Platform Designer instantiates the IP as a generic component that contains references to the HDL entity name, module and interface assignments, compilation library, HDL ports, interfaces, and system-info parameters. You can specify options that control the appearance of a component in the system.

To specify options that control the appearance of IP details and symbol in the system, follow these steps:

1. To open the **Component Instantiation** tab, click **View ➤ Component Instantiation**.
2. For **Implementation Type**, select the **IP** (Default), **HDL**, or **Blackbox** type. Component Implementation Type Options on page 61 defines these types.
3. Under **Compilation Info**, specify the **HDL Entity name** and **HDL compilation library** name for the implementation. These values are fixed for the **IP Implementation Type**.
4. In the **Signals & Interfaces** tab, define the port boundary of the component. Click ***<<add interface>>*** or ***<<add signal>>*** to add the interfaces and signals.
5. Optionally, click the **Block symbol** tab to visualize the signals and interfaces added in the **Signals & Interfaces** tab.
6. Optionally, in the **Export** tab you can export the signals and interfaces of an IP component as an IP-XACT file or a `_hw.tcl` file.

Figure 62. Component Instantiation Tab



Note: Platform Designer supports importing and exporting files in IP-XACT 2009 format and exporting IP-XACT files in 2014 format.

1.6.6.1. Component Implementation Type Options

Table 7. Component Implementation Type Options

Implementation Type	Description
IP	The default implementation type for any new component. Platform Designer reads the **IP Implementation Type** to perform the following functions: • Runs background checks against the port widths between the IP component and the `.ip` file to ensure continuity. • Scans the `.ip` file for the error flag to determine if any component has parameterization errors. • Checks for system-info mismatches between the IP file and the IP component in the system, and prompts resolution with IP instantiation warnings in the **Instantiation Messages** tab.
HDL	Allows you to define a component in your system from existing RTL. You can populate the signals and interfaces parameters of the generic component from an RTL file.
Blackbox	Defines a component that represents only the signal and interface boundary of an entity, without providing the component's implementation. You then provide the implementation of the component for processing with the Quartus Prime software or an RTL simulator.

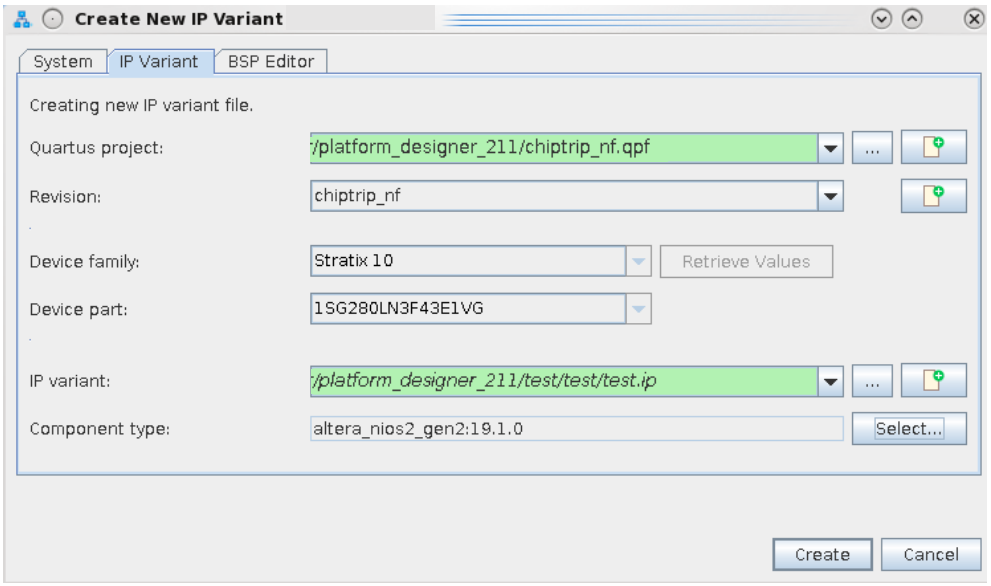
1.6.7. Creating or Opening an IP Core Variant

In addition to creating a system, Platform Designer allows you to define a stand-alone IP core variant that you can add to your Quartus Prime project or to a Platform Designer system.

Follow these steps to define an IP core variant in Platform Designer:

1. In Platform Designer, click **File ➤ New IP Variant**.
2. On the **IP Variant** tab, specify the **Quartus project** to contain the IP variant.
3. Specify any of the following options:
   - **Revision**—optionally select a specific revision of a project.
   - **Device family**—when defining a new project or **None**, allows you to specify the target FPGA device family. Otherwise this field is non-editable and displays the **Quartus project** target device family. Click **Retrieve Values** to populate the fields.
   - **Device part**—when defining a new project or **None**, allows you to specify the target FPGA device part number. Otherwise this field is non-editable and displays the **Quartus project** target device part number.
4. Specify the **IP variant** name, or browse for an existing IP variant.
5. For **Component type**, click **Select** and select the IP component from the IP Catalog.
6. Click **Create**. The IP parameter editor appears. Specify the parameter values that you want for the IP variant.
7. To generate the IP variant synthesis and optional simulation files, click **Generate HDL**, specify **Generation Options**, and click **Generate**. Refer to Generation Dialog Box Options on page 90 for generation options.

Figure 63. Platform Designer IP Variant Tab

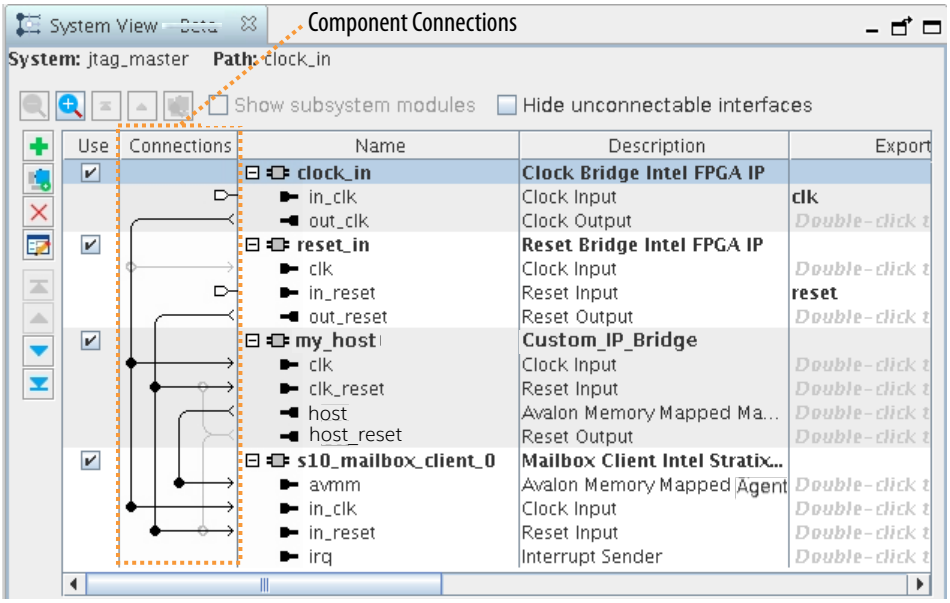


1.7. Connecting System Components

You must appropriately connect the components in your Platform Designer system. The **System View** and **Connections** tabs allow you to connect and configure IP components quickly. Platform Designer supports connections between interfaces of compatible types and opposite directions.

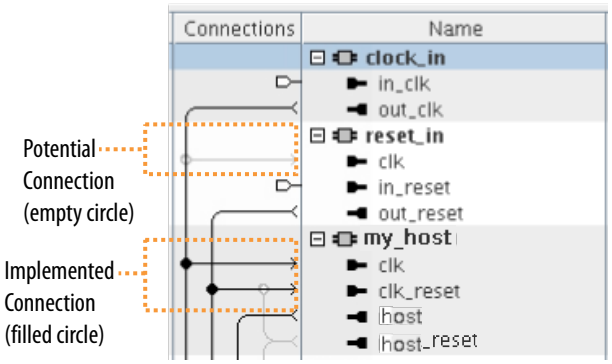
For example, you can connect a memory-mapped host interface to an agent interface, and an interrupt sender interface to an interrupt receiver interface. You can connect any interfaces exported from a Platform Designer system within a parent system.

Figure 64. Connections Column in the System Contents Tab



Platform Designer uses the high-level connectivity you specify to instantiate a suitable HDL fabric to perform the needed adaptation and arbitration between components. Platform Designer generates and includes this interconnect fabric in the RTL system output. Potential connections between interfaces appear as gray interconnect lines with an open circle icon at the intersection of the potential connection.

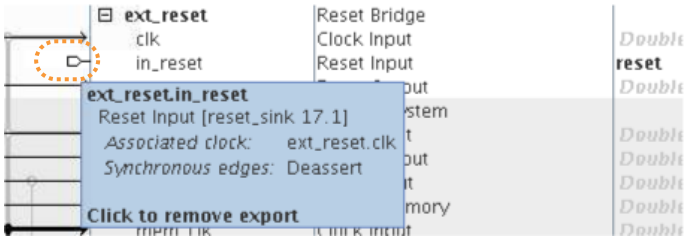
Figure 65. Potential and Implemented Connections in System View



To implement a connection, follow these steps:

1. Click inside an open connection circle to implement the connection between the interfaces. When you make a connection, Platform Designer changes the connection line to black, and fills the connection circle. Clicking a filled-in circle removes the connection.
2. To display the list of current and possible connections for interfaces in the **Hierarchy** or **System View** tabs, click **View ➤ Connections**.

Figure 66. Connection Display for Exported Interfaces



3. Perform any of the following to modify connections:
   - On the **Connections** tab, enable or disable the **Connected** column to enable or disable any connection. The **Clock Crossing**, **Data Width**, and **Burst** columns provide interconnect information about added adapters that can result in slower f_{MAX} or increased area utilization.
   - On the **System View** tab, right-click in the **Connection** column and disable or enable **Allow Connection Editing**.
   - On the **Connections** tab view and make connections for exported interfaces. Double-click an interface in the **Export** column to view all possible connections in the **Connections** column as pins. To restore the representation of the connections, and remove the interface from the **Export** column, click the pin.

1.7.1. Platform Designer 64-Bit Addressing Support

Platform Designer interconnect supports up to 64-bit addressing for all Platform Designer interfaces and IP components, with a range of: `0x0000 0000 0000 0000` to `0xFFFF FFFF FFFF FFFF`, inclusive.

The address parameters appear in the **Base** and **End** columns in the **System View** tab, on the **Address Map** tab, in the parameter editor, and in validation messages. Platform Designer displays as many digits as needed in order to display the top-most set bit, for example, 12 hex digits for a 48-bit address.

A Platform Designer system can have multiple 64-bit hosts, with each host having its own address space. You can share agents between hosts, and hosts can map agents to different addresses. For example, one host can interact with agent `0` at base address `0000_0000_0000`, and another host can see the same agent at base address `c000_000_000`.

Quartus Prime debugging tools provide access to the state of an addressable system via the Avalon memory mapped interconnect. These tools are also 64-bit compatible, and process within a 64-bit address space, including a JTAG to Avalon host bridge.

Platform Designer supports auto base address assignment for Avalon memory mapped components. In the **Address Map** tab, click **Auto Assign Base Address**.

1.7.1.1. Support for Avalon Memory Mapped Non-Power of Two Data Widths

Platform Designer requires that you connect all multi point Avalon memory mapped connections to interfaces with data widths that are equal to powers of two.

Platform Designer issues a validation error if an Avalon memory mapped host or agent interface on a multi-point connection is parameterized with a non-power of two data width.

Note: Avalon memory mapped point-to-point connections between an Avalon memory mapped host and an Avalon memory mapped agent are an exception, you can set their data widths to a non-power of two.

1.7.2. Connecting Hosts and Agents

Specify connections between host and agent components in the **Address Map** tab. You specify the address range that each memory-mapped host uses to connect to an agent in the Platform Designer system.

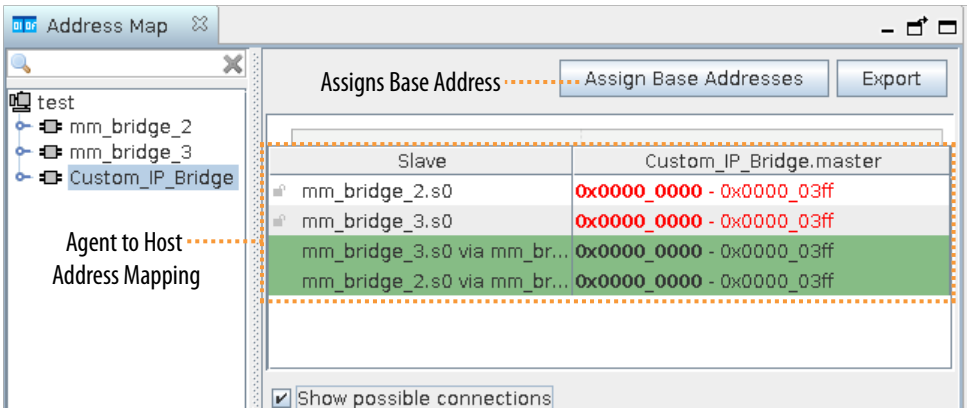
The **Address Map** tab shows the agents on the left, the hosts across the top, and the address span of the connection in each cell. If there is no connection between a host and an agent, the table cell is empty. In this case, use the **Address Map** tab to view the individual memory addresses for each connected host.

Platform Designer enables you to design a system where two hosts access the same agent at different addresses. If you use this feature, Platform Designer labels the **Base** and **End** address columns in the **System View** tab as "mixed" rather than providing the address range.

To create or edit a connection between host and agent IP components:

1. In Platform Designer, click the **Address Map** tab.
2. Locate the table cell that represents the connection between the host and agent.
3. Type in a base address, or update the current base address in the cell. The base address of an agent component must be a multiple of the address span of the component.

Figure 67. Address Map Tab for Connection Hosts and Agents



1.7.3. Connecting NoC IP in Platform Designer

Agilex™ 7 M-Series FPGAs support an integrated Network-on-Chip (NoC) to facilitate high-bandwidth data movement between the FPGA core logic and memory resources, such as HBM2e and DDR5 memories.

In the regular NoC compilation flow, you specify connections between NoC IP in the NoC Assignment Editor after running Analysis & Elaboration. You also specify address mapping and performance requirements for these connections in the NoC Assignment Editor. These assignments are required to perform RTL simulation of your design.

An optional early RTL simulation flow is available where you can specify NoC IP connection and addressing in Platform Designer. In this early simulation flow, you can perform RTL simulation after generating HDL for your Platform Designer system, but before running Analysis & Elaboration and making assignments in NoC Assignment Editor.

Note: If you use this early RTL simulation flow, you must still run Analysis & Elaboration and make connection and addressing assignments in NoC Assignment Editor before performing compilation.

If you want to enable early RTL simulation for your design, follow the steps below to specify NoC IP connectivity and addressing in Platform Designer. If you run Analysis & Elaboration and use the NoC Assignment Editor to make these assignments before running simulation, these steps are unnecessary.

1. Connect the AXI4 NoC manager ports to appropriate AXI4 NoC subordinate ports in the Platform Designer **System View** tab. The AXI4 NoC manager ports are on the NoC Initiator Intel FPGA IP. The AXI4 NoC subordinate ports are on the High Bandwidth Memory (HBM2E) Interface Agilex 7 FPGA IP and on the External Memory Interfaces Agilex 7 Intel FPGA IP.
2. Click the **Address Map** tab in Platform Designer to assign starting addresses for each NoC initiator to target connection. If an initiator connects to multiple targets, ensure that each target has a unique starting address. The end address is auto calculated based on the memory span.
3. Save you system and click **Generate HDL**. Platform Designer stores the NoC connectivity and addressing as assignments in the project `.qsf`.

   Note: This connectivity and addressing is not present in the HDL netlist that Platform Designer generates.

When using the Platform Designer flow for specifying NoC connectivity and addressing, the design is ready for RTL simulation after generating HDL for the Platform Designer system. For details on simulation, refer to *Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide*.

Related Information

Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide

1.7.4. Wire-Level Connectivity

Wire-level connectivity enables you to manipulate wire-level connections in the system level view of Platform Designer. For example, you can enter a Verilog style syntax expression to drive an input port of an IP component. You can implement wire-level connectivity with the Platform Designer GUI or with the `qsys-script` utility.

After applying the expression, the port you specify moves from the current interface into a **Wire-Level Endpoint** interface. The new interface name appends `_wirelevel` to the existing interface name. If you remove the wire-level expression, the port restores to the original interface. However, not all interfaces are restorable to legal interfaces after certain ports change. Moving a port from its original interface might result in validation errors on the original interface.

After you move a port to a **Wire-Level Endpoint** interface, wire-level expressions must drive all bits in the vector. You cannot connect ports contained within this new interface type to any other interfaces.

The following general rules apply to wire-level expressions:

- Wire-level connectivity is only available on optional input ports.
- Wire-level expressions can consist of input, output, and bi-directional ports, constant values, and logic terms using standard Verilog syntax.
- Wire-level expressions can only consist of ports within the same level of hierarchy. If you require elements from a higher or lower hierarchy, you must export the appropriate elements to the same hierarchical context.
- You can apply multiple expressions to a single input port unless they collide or cause bus contention.
- You must resolve validation errors occurring on the original interface for the interface to function correctly.

Platform Designer validates the wire-level expressions and provides messages for syntax, port existence, and other systematic errors. This validation includes the following:

- Validation of Verilog syntax.
- Warning if any sub-operator elements don’t match bit size.
- Warning if resulting combined bit size does not match the driven input port.
- Validation that all module and port names exist.
- Validation that all ports in a wire-level interface are input ports.
- Validation that all wire-level expressions drive each input port within a wire-level interface.
- Validation of no bus-contention, meaning that no one wire is driven by more than one expression.
- In a composed `_hw.tcl` module, validation that all ports driven by wire-level expressions are not in any connection.
- In a composed `_hw.tcl` module, validation that all ports driven by wire-level expressions are not exported.

After you define wire-level expressions, generate the system to create the Verilog files. When the Platform Designer GUI or `qsys-script` utility applies the expression, the expression inserts into the Verilog wrapper file that generates for your system. When you apply the wire-level connections with composed `_hw.tcl` commands, the wire-level expression inserts in the IP component's Verilog wrapper file.

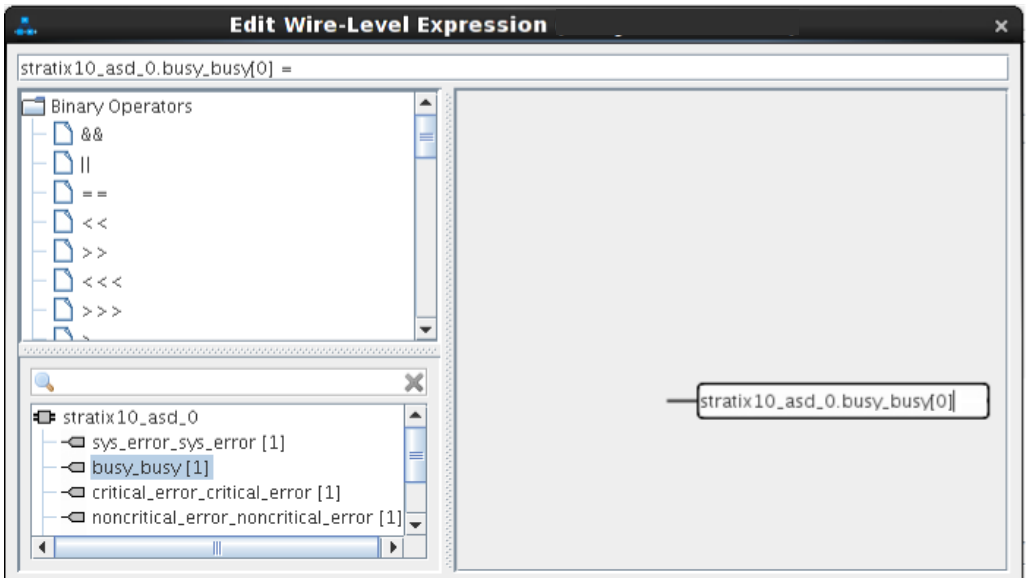
1.7.4.1. Editing Wire-Level Expressions

After you add a wire-level expression to an optional input port, you can add, edit, or remove wire-level expressions and connections in the Platform Designer GUI.

Follow these steps to edit wire-level expressions in the Platform Designer GUI:

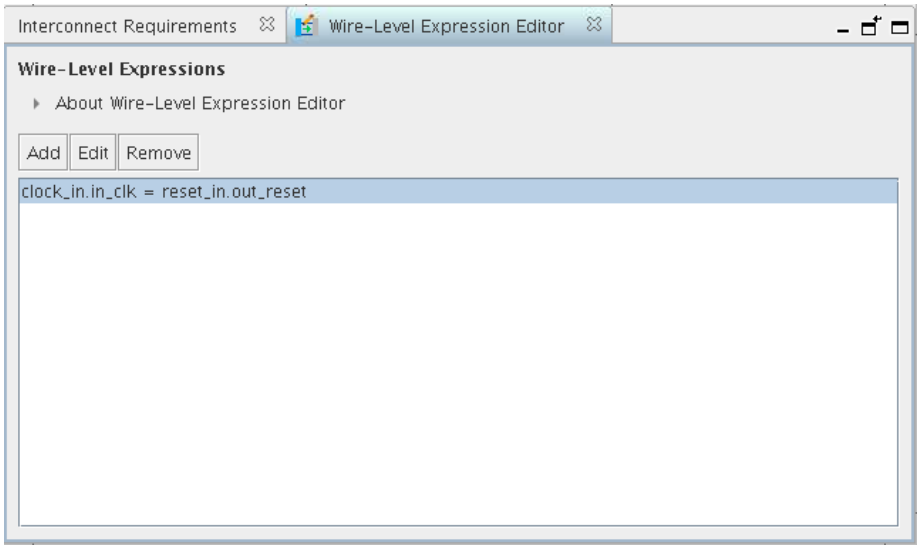
1. To specify a new wire-level expression, right-click an input port in the **Hierarchy** tab and click **Add Wire-Level Expression**. The **Edit Wire-Level Expression** dialog box appears.
2. To construct the expression, drag operators or ports from the list of operators or ports, and drop them into the expression field. Refer to Wire-Level Expression Syntax on page 69 for a list of legal operators.
3. Click the text field at the top of the **Edit Wire-Level Expression** dialog box and press the Down Arrow key to enable the expression assistant. The assistant provides a context sensitive list of available operators at the cursor position.

Figure 68. Edit Wire-Level Expression Dialog Box



4. Modify the elements of the expression in the workspace:
   - To add a value to an expression, right-click a node and select **Insert Value**.
   - Double-click on a value to enter a numeric value or port name.
   - Click an operator node to change the operator type.
   - Reorder nodes or move nodes between operators by dragging them.
5. To manage all wire-level expressions, click **View ➤ Wire-Level Expression Editor**. The **Wire-Level Expression Editor** allows you to add new wire-level expressions, edit, or remove existing wire-level expressions.

Figure 69. Wire-Level Expression Editor



1.7.4.2. Wire-Level Expression Syntax

The wire-level expression derives from Verilog syntax. The following is an example and list of legal operators and elements that you can use for wire-level expressions.

Example Expressions:

```
foo1.port1[5:0] = foo2.port1[5:0]
foo3.port1[8:4] = foo5.port1[4:0] & 5'b10101
foo6.port1[0] = 'b1
foo7.port1 = foo8.port1
foo9.port1[0] = ~foo10.port1[0]
foo10.port1[3:0] = foo11.port2[1:0] + 4'b1100
foo12:port1[3:0] = {4{0}}
foo13.port1[7:0] = {foo14.port1[3:0], 4'b0011}
```

Table 8. Ports

Port	Description
<instance_name>.<port_name>	Whole port
<instance_name>.<port_name>[*x*]	Wire *x* of port
<instance_name>.<port_name>[*y*:*x*]	Wires *x* to *y* of port. Port ranges must be in decreasing order, for example `a[1:0]`.
<constant base x values>	For example: 1, 'b1, 4'hf, 4'o7, 32'd9

Table 9. Operators (Bitwise)

Operator	Description
~	Negation
&	AND
|	OR
~&	NAND

continued...

Operator	Description
~|	NOR
^	XOR
~^	XNOR

Table 10. Operators (Logical)

Operator	Description
?	Conditional
!	Negation
&&	AND
||	OR

Table 11. Operators (Relational, Equality, and Shift)

Operator	Description
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To
==	Equal To
!=	Not Equal To
<<	Shift Left
>>	Shift Right

Table 12. Operators (Mathematical)

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Table 13. Operators (Other)

Operator	Description
{integer {*x*}}	Replication of *x*
{*x*, *y*, ...}	Concatenation

1.7.4.3. Adding or Removing Ports from Wire-Level Endpoint Interfaces

You can quickly add or remove ports from wire-level interfaces.

Follow these steps to add or remove ports from wire-level endpoint interfaces:

1. To move the port to a wire-level endpoint interface, in the **Hierarchy** tab, right-click a port and then click **Move Port to Wire-Level Interface**. After you move a port to a wire-level endpoint interface, you can view and edit it in the **Component Instantiation** tab.
2. To remove the port from a wire-level endpoint interface, in the **Hierarchy** tab, right-click a port and then click **Remove Port from Wire-Level Interface**.

1.7.4.4. Scripting Wire-Level Expressions

Platform Designer supports system scripting commands to apply wire-level expressions to input ports in IP components.

The following commands function with the `qsys-script` utility or in a `_hw.tcl` file to set, retrieve, or remove an expression on a port:

```
set_wirelevel_expression <instance_or_port_bit> <expression>
get_wirelevel_expressions <instance_or_port_bit>
remove_wirelevel_expressions <instance_or_port_bit
```

These commands require a string that you compose from the left-handed and right-handed components of the expression. Platform Designer reports errors in syntax, existence, or system hierarchy.

Related Information

- Wire-Level Connection Commands on page 633
- set_wirelevel_expression on page 633
- get_wirelevel_expressions on page 634
- remove_wirelevel_expressions on page 635

1.8. Specifying Interconnect Parameters

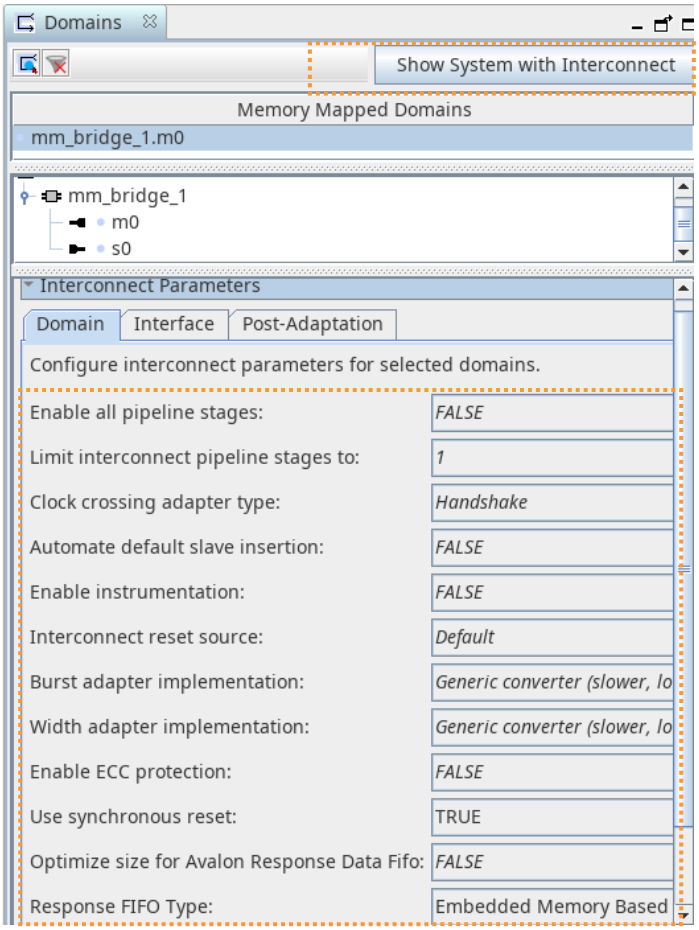
Specify system-wide or domain-specific interconnect parameters on the **Domains** tab. Interconnect parameters allow you to customize the implementation of the system interconnection.

The **Domains** tab displays the memory mapped interfaces in your system, and allows you to specify legal values for available system-level or interface parameters. While specifying parameters, you can click **Show System with Interconnect** to view a schematic preview of the Platform Designer complete system interconnect.

Follow these steps to specify interconnect parameters:

1. Open or create a Platform Designer system.
2. Click **View ➤ Domains**.
3. Under **Memory Mapped Domains**, select one or more domains.

Figure 70. Domains Tab



4. On the **Domain** tab, specify values for interconnect parameter settings, as Interconnect Parameters on page 73 describes.
5. To view a preview of the Platform Designer complete system interconnect, click **Show System with Interconnect**. Refer to Previewing the System Interconnect on page 74 for detailed description of this preview.
6. To manually add pipeline stages to the interconnect schematic, refer to Add Pipeline Stages to the Interconnect Schematic on page 330.

Related Information

- Avalon Interface Specifications
- Correcting Platform Designer System Timing Issues on page 76
- Interconnect Pipelining on page 328
- Burst Adapter on page 273
- Width Adaptation on page 271
- Platform Designer Interconnect on page 251
- Reset Interfaces on page 316
- Domains and Interfaces on page 483

1.8.1. Interconnect Parameters

The following parameters are available on the **Domains** tab:

Table 14. Interconnect Parameters

Option	Description
Enable all pipeline stages	• **FALSE**—default setting. The **Limit interconnect pipeline stages to** value and post-adaptation assignments control pipeline insertion. • **TRUE**—enables all pipeline stages within this domain. Ignores the **Limit interconnect pipeline stages to** and any post-adaptation assignment. Also disables manual editing of pipelines in the schematic. Although this setting ignores post-adaptation assignments, the assignments still remain in the design.
Limit interconnect pipeline stages to	Specifies the maximum number of pipeline stages that Platform Designer can insert in each command and response path to increase the f_{MAX} at the expense of additional latency. You can specify between 0 and 4 as the maximum number of pipeline stages to insert. The default value is 1. A value of 0 indicates no pipelines and a combinational datapath. When you specify 1 or greater, Platform Designer inserts *up to* the number you specify, depending on availability of pipeline locations. *Note:* If certain adapters or IP components are not present in the interconnect, or if there are not enough pipelineable locations in the interconnect, Platform Designer does not add all of the pipeline stages specified. Click **Show System with Interconnect** to view the number of stages added for a particular domain.
Clock crossing adapter type	Specifies the default implementation for automatically inserted clock crossing adapters: • **Handshake**—this adapter uses a simple handshaking protocol to propagate transfer control signals and responses across the clock boundary. This methodology uses fewer hardware resources because each transfer is safely propagated to the target domain before the next transfer can begin. The **Handshake** adapter is appropriate for systems with low throughput requirements. • **FIFO**—this adapter uses dual-clock FIFOs for synchronization. The latency of the FIFO-based adapter is a couple of clock cycles more than the handshaking clock crossing component. However, the FIFO-based adapter can sustain higher throughput because it supports multiple transactions at any given time. FIFO-based clock crossing adapters require more resources. The **FIFO** adapter is appropriate for memory-mapped transfers requiring high throughput across clock domains. • **Auto**—if you select **Auto**, Platform Designer specifies the **FIFO** adapter for bursting links, and the **Handshake** adapter for all other links.
Automate default slave insertion	Directs Platform Designer to automatically insert a default Avalon agent or AXI subordinate for undefined memory region accesses during system generation.(2)
Enable instrumentation	When you set this option to TRUE, Platform Designer enables debug instrumentation in the Platform Designer interconnect, which then monitors interconnect performance in the system console.
Interconnect reset source	Select **Default** or a specific reset signal in your design.
Burst adapter implementation	Allows you to choose the converter type that Platform Designer applies to each burst. • **Generic converter (slower, lower area)**—default setting. Controls all burst conversions with a single converter that is able to adapt incoming burst types. This results in an adapter that has lower f_{MAX}, but smaller area. • **Per-burst-type converter (faster, higher area)**—controls incoming bursts with a particular converter, depending on the burst type. This results in an adapter that has higher f_{MAX}, but higher area. This setting is useful when you have AXI managers or subordinates and you want a higher f_{MAX}.

continued...

(2) This document now refers to the Avalon "host" and "agent," and the AXI "manager" and "subordinate," to replace former terms. Refer to the current *AMBA AXI and ACE Protocol Specification* for the latest AMBA AXI and ACE protocol terminology.

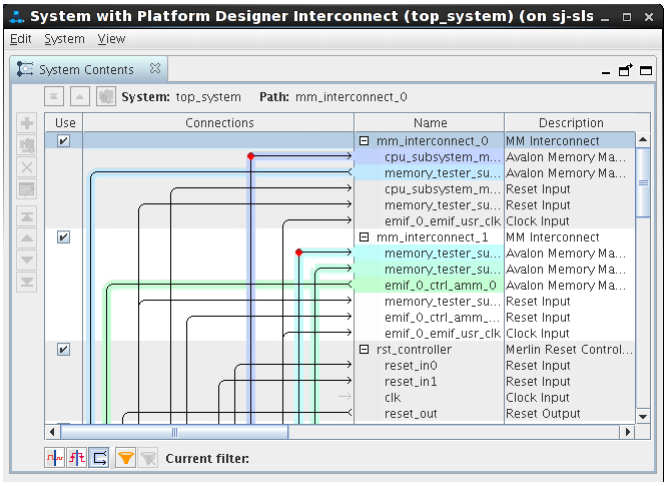
Option	Description
Width adapter implementation	• **Generic converter (slower, lower area)**—default. Controls all width adaptations with a single converter that is able to adapt incoming widths. This results in an adapter that has lower f_{MAX}, but smaller area. • **Optimized converter (faster, higher area)**—controls width adaptations with a particular converter, depending on the width. This results in an adapter that has higher f_{MAX}, but higher area. This setting is useful when you have AXI managers or subordinates and you want a higher f_{MAX}.
Enable ECC protection	Specifies the default implementation for ECC protection for memory elements. • **FALSE**—default. Disables ECC protection for memory elements in the Platform Designer interconnect. • **TRUE**—enables ECC protection for memory elements. Platform Designer interconnect sends uncorrectable errors arising from memory as DECODEERROR (DECERR) on the Avalon response bus. For more information about Error Correction Coding (ECC), refer to Error Correction Coding (ECC) in Platform Designer Interconnect on page 331.
Use synchronous reset	When set to **TRUE**, all registers in the interconnect use synchronous reset. Assert the reset for at least 16 cycles and start transactions 16 cycles after deassertion of the reset. This period allows all the IP components to reset and come out of the reset state. To avoid deadlocks in the interconnect, reset hosts and agents simultaneously. If hosts and agents have different resets, agents must reset only after responding to all necessary transactions. The **Use synchronous reset** option is enabled by default for Hyperflex® architecture devices, but is disabled by default for all other devices. Enabling synchronous reset for the interconnect does not enable synchronous reset for IP components in the system. You must separately enable the synchronous reset parameter for any component.
Optimize size for Avalon Response Data Fifo	When enabled, Platform Designer does not increase the size of the Avalon interface agent read FIFO to the next power of 2. You can enable this setting to decrease the size of the FIFO if needed.
Response FIFO Type	Specifies the implementation method for the Platform Designer response FIFO that stores information for processing of response in the interconnect, such as width adaptation and routing to the Avalon host or AXI manager. The following options are available: • **Register Based**—default option that implements the response FIFO using registers. When **Register Based** is selected, FIFO uses the LUT's to hold responses. This consumes more LUT's, but the minimum response latency is 1. • **Embedded Memory Based**—implements the response FIFO using M20K memory blocks, rather than LUTs. When you specify **Embedded Memory Based**, the minimum response latency is 3, and FIFO consumes far fewer LUT resources, but consumes M20K memory resources.

1.8.2. Previewing the System Interconnect

You can review a graphical representation of the Platform Designer interconnect before you generate the system. The System with Platform Designer Interconnect window Platform Designer's conversion of connections between interfaces to interconnect logic during system generation.

To open the System with Platform Designer Interconnect window, click **System ➤ Show System With Platform Designer Interconnect**, or click the **Show System With Interconnect** button in the **Domains** tab.

Figure 71. System with Platform Designer Interconnect window



The System with Platform Designer Interconnect window has the following tabs:

- **System Contents**—displays the original instances in your system, as well as the inserted interconnect instances. Connections between interfaces are replaced by connections to interconnect where applicable.
- **Schematic**—displays a schematic representation that shows the multiple interconnects together as a complete system.
- **Hierarchy**—displays a system hierarchical navigator, expanding the system contents to show modules, interfaces, signals, contents of subsystems, and connections.
- **Parameters**—displays the parameters for the selected element in the **Hierarchy** tab.
- **Memory-Mapped Interconnect**—allows you to select a memory-mapped interconnect module and view its internal command and response networks. You can also insert pipeline stages to achieve timing closure.

The **System Contents**, **Hierarchy**, and **Parameters** tabs are read-only. Edits that you apply on the **Memory-Mapped Interconnect** tab are automatically reflected on the **Interconnect Requirements** tab.

The **Memory-Mapped Interconnect** tab in the System with Platform Designer Interconnect window displays a graphical representation of command and response datapaths in your system. Datapaths allow you precise control over pipelining in the interconnect, as Add Pipeline Stages to the Interconnect Schematic on page 330 describes. Platform Designer displays separate figures for the command and response datapaths. You can access the datapaths by clicking their respective tabs in the **Memory-Mapped Interconnect** tab.

Each node element in a figure represents either a host or agent that communicates over the interconnect, or an interconnect sub-module. Each edge is an abstraction of connectivity between elements, and its direction represents the flow of the commands or responses.

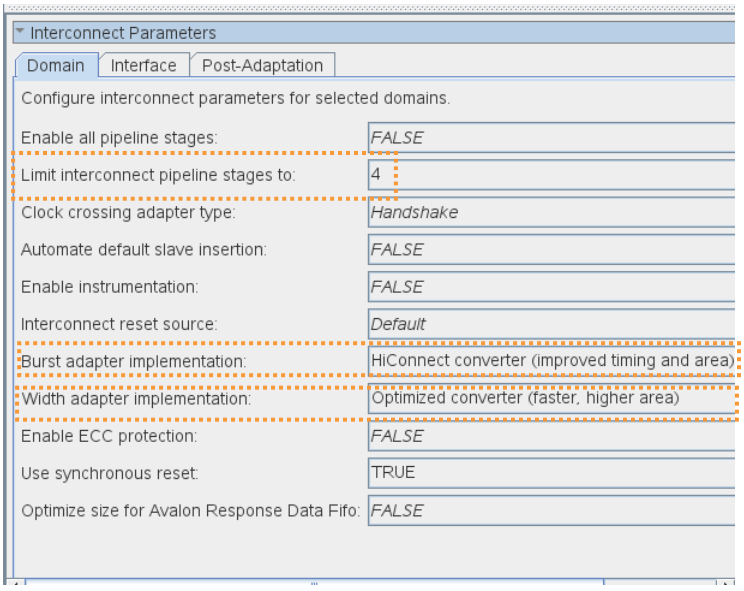
Click **Highlight Mode** (**Path**, **Successors**, **Predecessors**) to identify edges and datapaths between modules. Turn on **Show Pipelinable Locations** to add greyed-out registers on edges where pipelining is allowed in the interconnect.

1.9. Correcting Platform Designer System Timing Issues

You can help alleviate Platform Designer system timing issues by adjusting the following interconnect parameters on the **Domains** tab:

- **Limit interconnect pipeline stages to**—increase the use of interconnect pipeline stages in each command and response path by increasing the value of this option up to **4**. Platform Designer can insert up to four pipeline stages, depending on availability of pipeline locations, at the possible expense of additional latency and area. The default value is **1** pipeline stage.
- **Burst adapter implementation**—enable the **HiConnect converter (Improved timing and area)** or **Per-burst-type HiConnect converter (faster, higher area)** implementation to control incoming bursts with optimized burst adapters. These settings produce an adapter that has improved timing, but may require more device resources for implementation.
- **Width adapter implementation**—enable the **Optimized converter (faster, higher area)** implementation to control width adaptation with a particular converter. This setting produces an adapter that has higher f_{MAX}, but requires more device resources for implementation.

Figure 72. Correcting Platform Designer System Timing Issues with Domains Tab Settings



Note: To manually add pipeline stages to the interconnect schematic, refer to Add Pipeline Stages to the Interconnect Schematic on page 330.

Related Information

- Specifying Interconnect Parameters on page 71
- Interconnect Pipelining on page 328
- Burst Adapter on page 273
- Width Adaptation on page 271

1.10. Specifying Signal and Interface Boundary Requirements

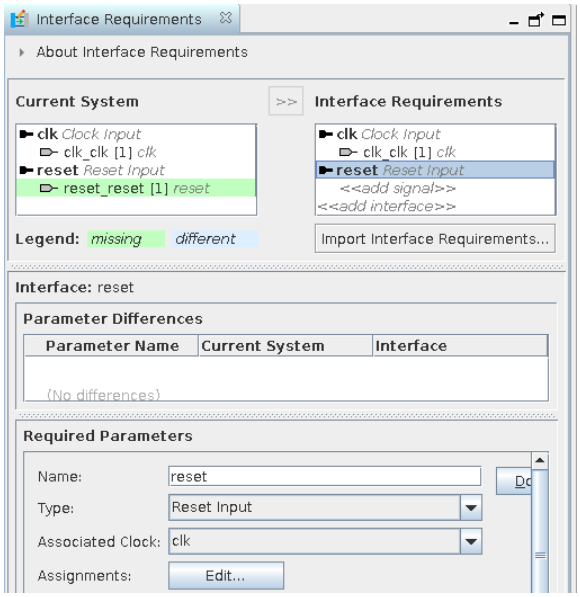
If you export an interface that does not match the interface requirements of the system, Platform Designer generates component instantiation errors. You must match all the exported interfaces with the interface requirements of the system.

The **Interface Requirements** tab allows you to assign a component's top-level HDL module signals to an interface, specify the expected signal and interface boundary requirements for the interface, and to resolve any interface requirement mismatches. You can also modify the signal names in an exported interface.

1. Click **View ➤ Interface Requirements**.
2. To load the interface requirements from a Platform Designer system, click **Import Interface Requirements** in the **Interface Requirements** table. Select the `.ipxact` representation of the Platform Designer system.
3. To manually add new interface or signal requirements, click *<<add interface>>* or *<<add signal>>* in the **Interface Requirements** table.
4. To correct the mismatches, select the missing or mismatched interface or signal in the **Current System** table and click **>>**.

   Note: Platform Designer highlights the mismatches between the system and interface requirements in blue, and highlights the missing interfaces and signals in green.
5. To rename an exported signal or interface, use any of the following methods:
   - Double-click the signal or interface in **Current System** table.
   - Select the signal or interface in the **Current System** table and press F2.
   - Select the signal or interface in the **Current System** table and rename from the **Current Parameters** pane at the bottom of the tab. The **Current Parameters** pane displays all the parameters of the selected interface or signal.

Figure 73. Interface Requirements Tab



Related Information

Editing Exported Interface Signal Names on page 78

1.10.1. Interface Requirements Tab Fields

The Platform Designer **Interface Requirements** tab contains the following fields.

Table 15. Interface Requirements Tab Fields

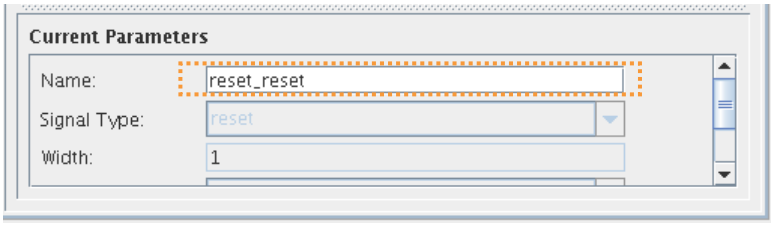
Name	Description
Current System	Displays all the exported interfaces in the current Platform Designer system. Add or remove the interfaces in the **Current System** by adding or removing components from the **System View** tab.
Interface Requirements	This table shows all the interface requirements set for the current Platform Designer system.
Parameter Differences	This table lists the **Parameter Name**, **Current System Value**, and **Interface Requirement Value** for the selected mismatched interface. *Note:* The **Interface Requirements** tab highlights in blue the signals and interfaces that are the same, but have different parameter values. Selecting a blue item populates the **Parameter Differences** table.
Import Interface Requirements	This button allows you to populate the **Interface Requirements** table from an IP-XACT(3) file representing a generic component or an entire Platform Designer system.
Parameters	This table lists the signal and interface parameters for the selected interface. You can view the table as **Current Parameters** when you select an interface or signal from the **Current System** table, and as **Required Parameters** when you select the signal or interface from **Interface Requirements** table. You can modify the name of the exported signal or interface from this table. For more information about how to edit the name of an exported signal or interface, refer to *Edit the Name of Exported Interfaces and Signals*.

1.10.2. Editing Exported Interface Signal Names

To rename an exported signal or interface:

- Double-click the signal or interface in **Current System** table.
- Select the signal or interface in the **Current System** table. Edit the interface or signal's parameters in the **Current Parameters** pane.

Figure 74. Editing the Name of Exported Interfaces and Signals



Note: All other parameters in the **Current Parameters** except **Name** are read-only for the current system.

(3) Platform Designer supports importing and exporting files in IP-XACT 2009 format and exporting IP-XACT files in 2014 format.

1.11. Configuring Platform Designer System Security

Specify system or interconnect **Security** requirements on the **Domains** tab.

Platform Designer interconnect supports the Arm TrustZone® security extension. The Platform Designer Arm TrustZone security extension includes secure and non-secure transaction designations, and a protocol for processing between the designations.(4)

The AXI AxPR0T protection signal specifies a secure or non-secure transaction. When an AXI manager sends a command, the AxPR0T signal specifies whether the command is secure or non-secure. When an AXI subordinate receives a command, the AxPR0T signal determines whether the command is secure or non-secure. Determining the security of a transaction while sending or receiving a transaction is a run-time protocol.

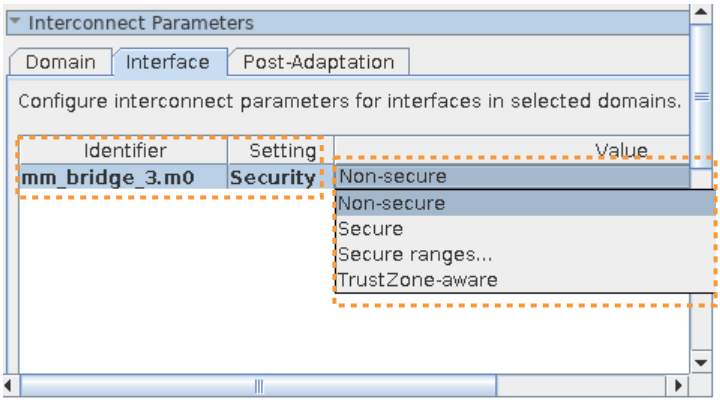
AXI managers and subordinates can be TrustZone-aware. All other host and agent interfaces, such as Avalon memory mapped interfaces, are non-TrustZone-aware.

The Avalon specification does not include a protection signal. Consequently, when an Avalon host sends a command, there is no embedded security and Platform Designer recognizes the command as non-secure. Similarly, when an Avalon agent receives a command, the agent always accepts the command and responds.

To set compile-time security support for non-TrustZone-aware components:

1. To begin creating a secure system, add Avalon hosts and agents and AXI managers and subordinates to your system, as Adding IP Components to a System describes.
2. Make connections between the hosts and agents and between the managers and subordinates in your system, as Connecting Hosts and Agents describes.
3. Click **View ➤ Domains**.

Figure 75. Security Settings in Domains Tab



4. To specify security requirements for an interconnect, click the **Interface** tab under **Interconnect Parameters**,

(4) This document now refers to the Avalon "host" and "agent," and the AXI "manager" and "subordinate," to replace formerly used terms with inclusive language. Refer to the current *AMBA AXI and ACE Protocol Specification* for the latest AMBA AXI and ACE protocol terminology.

5. Click the **Add** button.
6. In the **Identifier** column, select the interconnect in the **new_target** cell.
7. In the **Setting** column, select **Security**.
8. In the **Value** column, select the appropriate **Secure**, **Non-Secure**, **Secure Ranges**, or **TrustZone-aware** security for the interface. Refer to System Security Options for details of each option.
9. After setting compile-time security options for non-TrustZone-aware manager and subordinate interfaces, you must identify those managers that require a default subordinate before generation, as Specifying a Default Avalon Agent or AXI Subordinate describes.

Related Information

- Platform Designer Interconnect on page 251
- Platform Designer System Design Components on page 367

1.11.1. System Security Options

Table 16. Security Options

Option	Description
Secure	The Avalon host or AXI manager sends only secure transactions, and the Avalon agent or AXI subordinate receives only secure transactions. Platform Designer treats transactions from a secure manager as secure. Platform Designer blocks non-secure transactions to a secure agent or subordinate and routes to the default agent or subordinate.
Non-Secure	The host or manager sends only non-secure transactions, and the agent or subordinate receives any transaction, secure or non-secure. Platform Designer treats transactions from a non-secure host or manager as non-secure. Platform Designer allows all transactions, regardless of security status, to reach a non-secure agent or subordinate.
Secure Ranges	Applies to only the agent or subordinate interface. Allows you to specify secure memory regions for an agent or subordinate. Platform Designer blocks non-secure transactions to secure regions and routes to the default agent or subordinate. The specified address ranges within the agent or subordinate's address span are secure, all other address ranges are not. The format is a comma-separated list of inclusive-low and inclusive-high addresses, for example, `0x0:0xfff,0x2000:0x20ff`
TrustZone-aware	TrustZone-aware managers have signals that control the security status of their transactions. TrustZone-aware subordinates can accept these signals and handle security independently. The following applies to secure systems that mix secure and non-TrustZone-aware components: • All AXI, AMBA 3 AXI, and AMBA 3 AXI-Lite managers are TrustZone-aware. • You can set AXI, AMBA 3 AXI, and AMBA 3 AXI-Lite subordinates as TrustZone-aware, secure, non-secure, or secure range ranges. • You can set non-AXI host interfaces as secure or non-secure. • You can set non-AXI agent interfaces as secure, non-secure, or secure address ranges.

1.11.2. Specifying a Default Avalon Agent or AXI Subordinate

If an AXI manager issues "per-access" or "not allowed" transactions, your design must contain a default subordinate. Per-access refers to the ability of a TrustZone-aware AXI manager to allow or disallow access or transactions.

You can achieve an optimized secure system by partitioning your design and carefully designating secure or non-secure address maps to maintain reliable data. Avoid a design that includes a non-secure AXI manager or Avalon host that initiates transactions to a secure subordinate or agent resulting in unsuccessful transfers, within the same hierarchy.

A transaction that violates security is rerouted to the default subordinate or agent and subsequently responds to the AXI manager or Avalon host with an error. The following rules apply to specifying a default subordinate or agent:

- You can designate any AXI subordinate or Avalon agent as the default subordinate or agent.
- You can share a default subordinate or agent between multiple AXI managers or Avalon hosts, respectively.
- You should have one default subordinate or agent for each interconnect domain.
- An interconnect domain is a group of connected memory-mapped managers and subordinates or hosts and agents that share the same interconnect. The `altera_error_response_slave` component includes the required TrustZone features.
- *Note:* If you do not specify a value for the **Default Slave** option, and the **Automate default slave insertion** option is off, Platform Designer automatically assigns the AXI subordinate or Avalon agent in the system. Platform Designer automatically assigns the AXI subordinate or Avalon agent that has largest address span within the memory map for the issuing AXI manager or Avalon host. In the case of multiple, large AXI subordinates or Avalon agents that have the same address span, Platform Designer selects the AXI subordinate or Avalon agent at the lowest base offset.

To designate a subordinate or agent interface as the default subordinate or agent for non-TrustZone-aware interfaces, follow these steps:

1. Specify interconnect security settings, as Configuring Platform Designer System Security on page 79 describes.
2. In the **System View**, right-click any column and turn on the **Security** and **Default Slave** columns.
3. In the **System View** tab, turn on the **Default Slave** option for the subordinate or agent interface. A manager or host can have only one default subordinate or agent.

Figure 76. Security and Other Columns

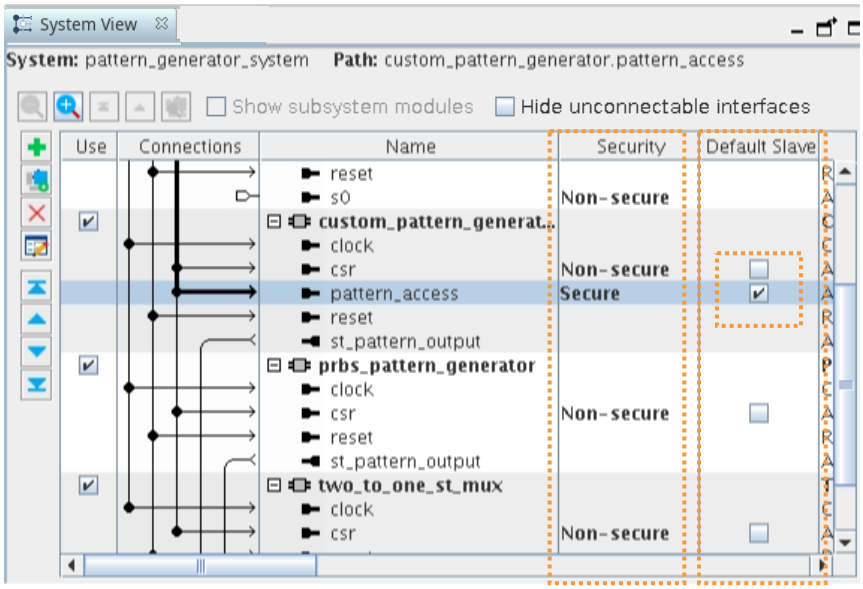


Table 17. Secure and Non-Secure Access Between Manager or Host, Subordinate or Agent, and Memory Components

Transaction Type	TrustZone-aware manager	Non-TrustZone-aware manager/host Secure	Non-TrustZone-aware manager/host Non-Secure
TrustZone-aware subordinate memory	OK	OK	OK
Non-TrustZone-aware subordinate/ agent (secure)	Per-access	OK	Not allowed
Non-TrustZone-aware subordinate/ agent (non-secure)	OK	OK	OK
Non-TrustZone-aware memory (secure region)	Per-access	OK	Not allowed
Non-TrustZone-aware memory (non-secure region)	OK	OK	OK

Related Information

- Error Response Slave Intel FPGA IP on page 390
- Designating a Default Agent on page 396

1.11.3. Accessing Undefined Memory Regions

Access to an undefined memory region occurs when a transaction from an AXI manager or Avalon host targets a memory region unspecified in the AXI subordinate or Avalon agent memory map. To ensure predictable response behavior when this condition occurs, you must specify a default subordinate or agent, as Specifying a Default Avalon Agent or AXI Subordinate on page 80 describes.

You can designate any memory-mapped subordinate or agent as a default subordinate or agent. You may have only one default subordinate or agent for each interconnect domain in your system. Platform Designer then routes undefined memory region accesses to the default subordinate or agent, which terminates the transaction with an error response.

Note: If you do not specify a value for the **Default Slave** option, and the **Automate default slave insertion** option is off, Platform Designer automatically assigns the AXI subordinate or Avalon agent in the system. Platform Designer automatically assigns the AXI subordinate or Avalon agent that has largest address span within the memory map for the issuing AXI manager or Avalon host. In the case of multiple, large AXI subordinates or Avalon agents that have the same address span, Platform Designer selects the AXI subordinate or Avalon agent at the lowest base offset.

Accessing undefined memory regions can occur in the following cases:

- When there are gaps within the accessible memory map region that are within the addressable range of subordinate or hosts, but are not mapped.
- Accesses by a manager or host to a region mapped to that manager or host that does not belong to any subordinates or agents.
- When a non-secured transaction is accessing a secured subordinate. This applies to only subordinates that are secured at compilation time.
- When a read-only subordinate or agent is accessed with a write command, or a write-only subordinate or agent is accessed with a read command.

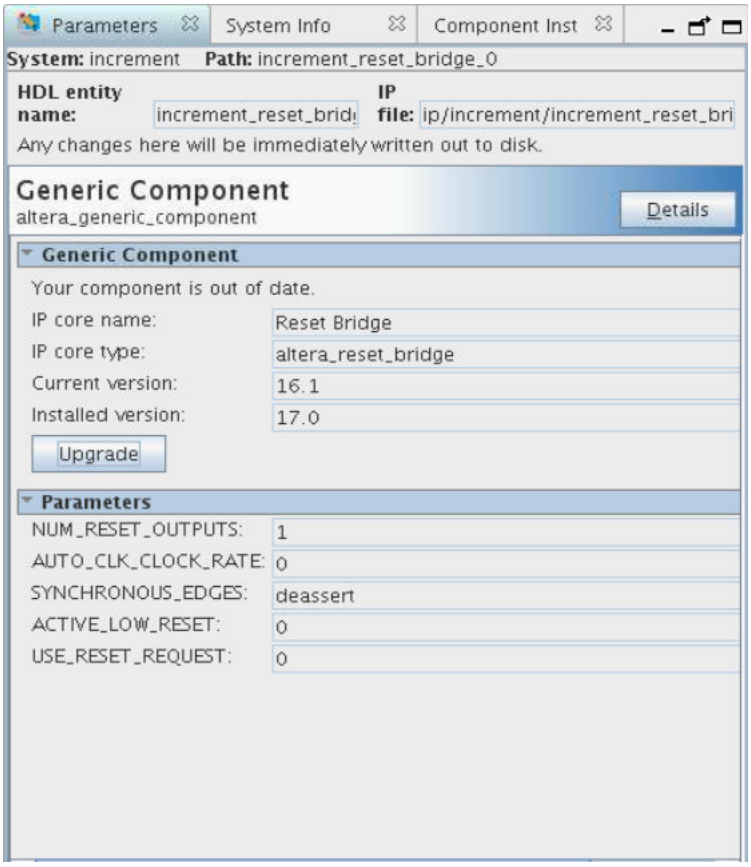
1.12. Upgrading Outdated IP Components in Platform Designer

When you open a Platform Designer system containing outdated IP components, you can retain and use the RTL of previously generated IP components within the Platform Designer system. If Platform Designer is unable to locate the IP core's original version, you cannot re-parametrize the IP core without upgrading the IP core to the latest version. However, Platform Designer allows you to view the parametrization of the original IP component without upgrading.

To upgrade individual IP components in your Platform Designer system:

1. Click **View ➤ Parameters**.
2. Select the outdated IP component in the **Hierarchy** or the **System View** tab.
3. Click the **Parameters** tab. This tab displays information on the current version, as well as the installed version of the selected IP component.
4. Click **Upgrade**. Platform Designer upgrades the IP component to the installed version, and deletes all the RTL files associated with the IP component.

Figure 77. Upgrade IP Component in your Platform Designer System



To upgrade an IP component from the command-line, type the following:

```
qsys-generate --upgrade-ip-cores <ip_file>
```

To upgrade all the IP components in your Platform Designer system, open the associated project in the Quartus Prime software, and click **Project ➤ Upgrade IP Components**.

1.13. Synchronizing System Component Information

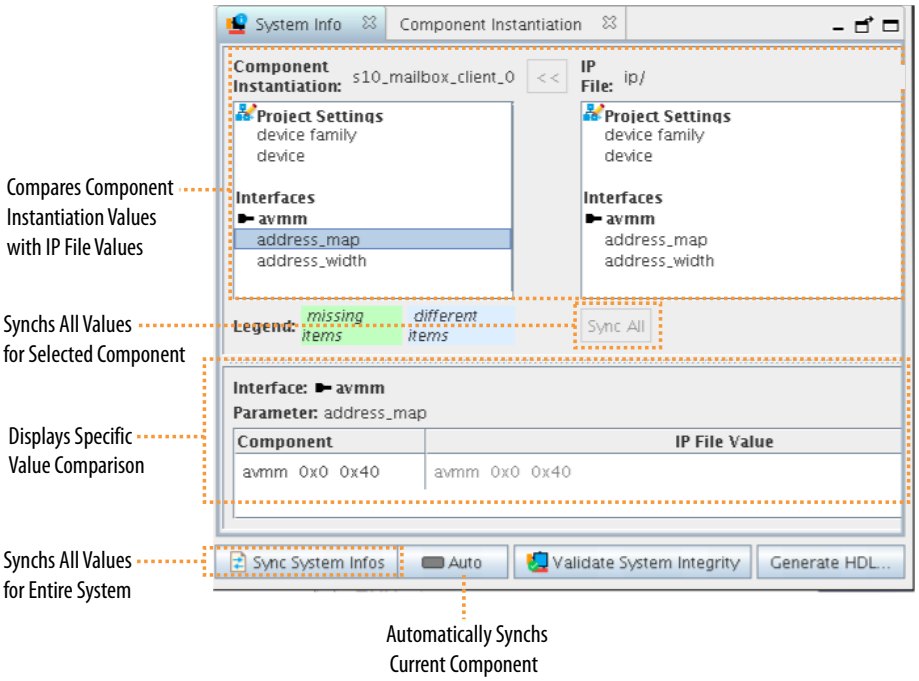
When a component instantiation values do not match the component's corresponding `.ip` file, Platform Designer reports these mismatches as Component Instantiation Warnings in the **System Messages** tab.

You must synchronize any mismatches between the component instantiation, and the component's corresponding `.ip` prior to system generation.

Follow these steps to synchronize one or more components in your system:

1. Select the mismatched signal or interface in the **System View** tab, and then click **View ➤ System Info**. Alternatively, you can double-click the corresponding Component Instantiation Warning in the **System Messages** tab.

Figure 78. System Info Tab



2. View any component mismatches in the **System Info** tab. Select individual interfaces, signals, or parameters to view the specific value differences in the **Component** and **IP file** columns. Value mismatches between the **Component Instantiation** and the **IP file** appear in blue. Missing elements appear in green.
3. To synchronize the **Component Instantiation** and **IP file** `.ip` values in the system, perform one or more of the following:
   - Select a specific mismatched parameter, interface, or signal and click **>>** to synchronize the items.
   - Click **Sync All** to synchronize all values for the current component.
   - Click **Sync System Infos** to synchronize all IP components (except "Missing Items") in the current system at once.

   Note: The following guidelines apply to synchronizing system elements that Platform Designer identifies as "Different Items" and "Missing Items":
   - You can synchronize "Different Items" but not "Missing Items."
   - For "Missing Items," you must replace the IP component in the Platform Designer system to ensure that the number and names of the parameters match.

1.13.1. System Info Tab Fields

Table 18. System Info Tab Fields

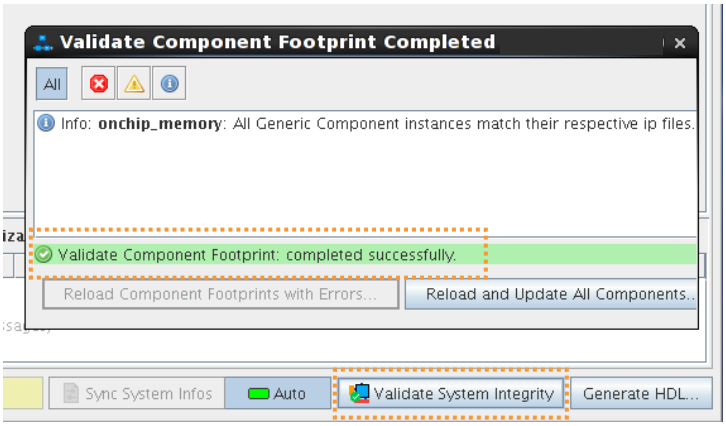
Name	Description
Component Instantiation	Lists the signals and interfaces for the selected component with respect to the component instantiation. Value mismatches between the **Component Instantiation** and the **IP file** appear in blue. Missing elements appear in green.
IP file	Lists the signal and interface information with respect to the `.ip` file. Value mismatches appear in blue. Missing elements appear in green.
Component Column	Displays the selected interface parameter value with respect to the **Component Instantiation**.
IP File Value	Displays the selected interface parameter value with respect to the **IP file**.
>>	Manually synchronizes the selected mismatch between signals and interfaces in the **Component Instantiation** and the **IP file**.
Sync All	Synchronizes **Component Instantiation** and **IP file** mismatches in the current system.

1.14. Validating System Integrity

You can use any of the following methods to validate Platform Designer system integrity.

- To perform system integrity check for the entire system, click the **Validate System Integrity** button at the bottom of main Platform Designer window. If validation finds errors, click **Reload and Update All Components** to reload signal and interface values from the corresponding IP component file.

Figure 79. Validating System Integrity



- View any errors and warnings on the **System Messages** tab. Double-click the warning or error messages to locate the issue in the **System View** or **Parameters** tab to correct the issue. Platform Designer generates the following types of system validation errors and warnings:

Table 19. System Messages Types in Platform Designer

System Messages Types	Description
Component Instantiation Warning	Indicates the mismatches between system information parameters or IP core parameterization errors. A system information parameters mismatch refers to the mismatch between an IP component's system parameter expectations and the component's saved system information parameters in the corresponding `.ip` file. For example: • Interface types do not match • Interface is missing • Port has been moved to another interface • Port role has changed • Interface assignment is mismatched • Interface assignment is missing
Component Instantiation Error	Indicates the mismatches between HDL entity name, compilation library, or ports which results in downstream compilation errors. The component instantiation errors always indicate the fundamental mismatches between generated system and interconnect fabric RTL. For example: • Port is missing from the ip file • Port is missing from instantiation • Port direction has changed • Port HDL type has changed • Port width has changed • Interface Parameter is mismatched • Interface Parameter is missing
System Connectivity Warning	Platform Designer system connectivity warnings.
System Connectivity Error	Platform Designer system connectivity errors.

1.14.1. Validating the System Integrity of Individual Components

To validate the system integrity for your IP components:

1. Select the IP component in the **System View** tab.
2. Right-click and select **Validate Component Footprint** to check for any mismatches between the IP component and its `.ip` file representation.
3. If there are any errors, click **Reload Component Footprint** to reload the signals and interfaces for the component from the `.ip` file.

1.15. Preserving System Elements for Debug

The Compiler optimizes the module, interface, and port names in your Platform Designer system during synthesis and place-and-route. Unless preserved, the module, interface, and port names in your system may not exist in the post-fit netlist after optimizations. For example, synthesis can merge duplicate registers, or add tildes (~) to net names that fan-out from a node.

You can use any of the following methods to prevent synthesis from performing optimizations on specific modules, interfaces, or port names, allowing the names to persist into the post-fit netlist for Signal Tap and other debug monitoring.

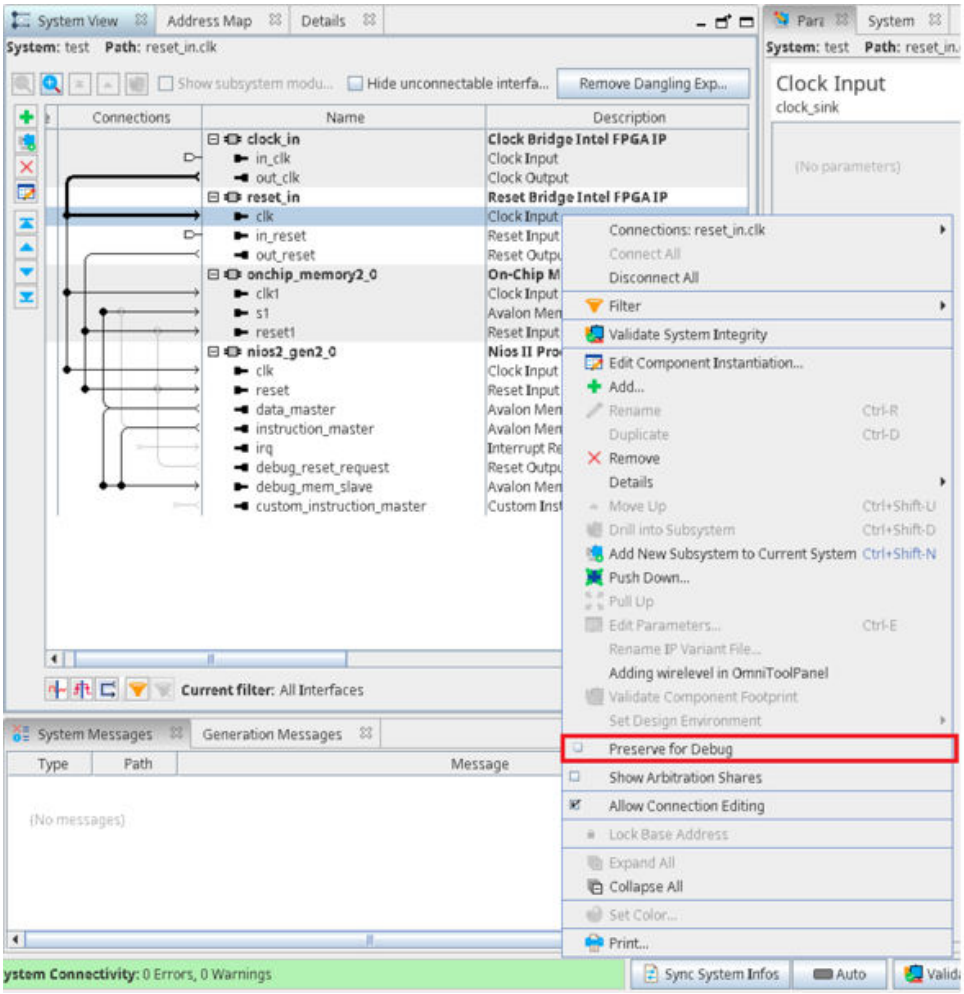
1. Enable preserve for debug using any of the following methods:

- On the **System View** tab, right-click any module, interface or port, and click **Preserve for Debug** in the context menu.
- On the **Filter** tab, right-click any module, interface, or port, and click **Preserve for Debug** in the context menu. A green icon color in the **Filter** tab indicates that preservation is applied.

Applying **Preserve for Debug** adds corresponding assignments to the `.qip`. Once applied, you can verify the assignments are correct in the `.qip`.

2. Recompile the design to apply the **Preserve for Debug** and view the preserved nodes following compilation.

Figure 80. Preserve for Debug from System View



Note: For more information about preserving signals, refer to *Preserving Registers During Synthesis*, in *Hyperflex Architecture High-Performance Design Handbook*. Specifying preservation synthesis attributes may increase device resource utilization or decrease timing performance when the attribute is active.

1.16. Generating a Platform Designer System

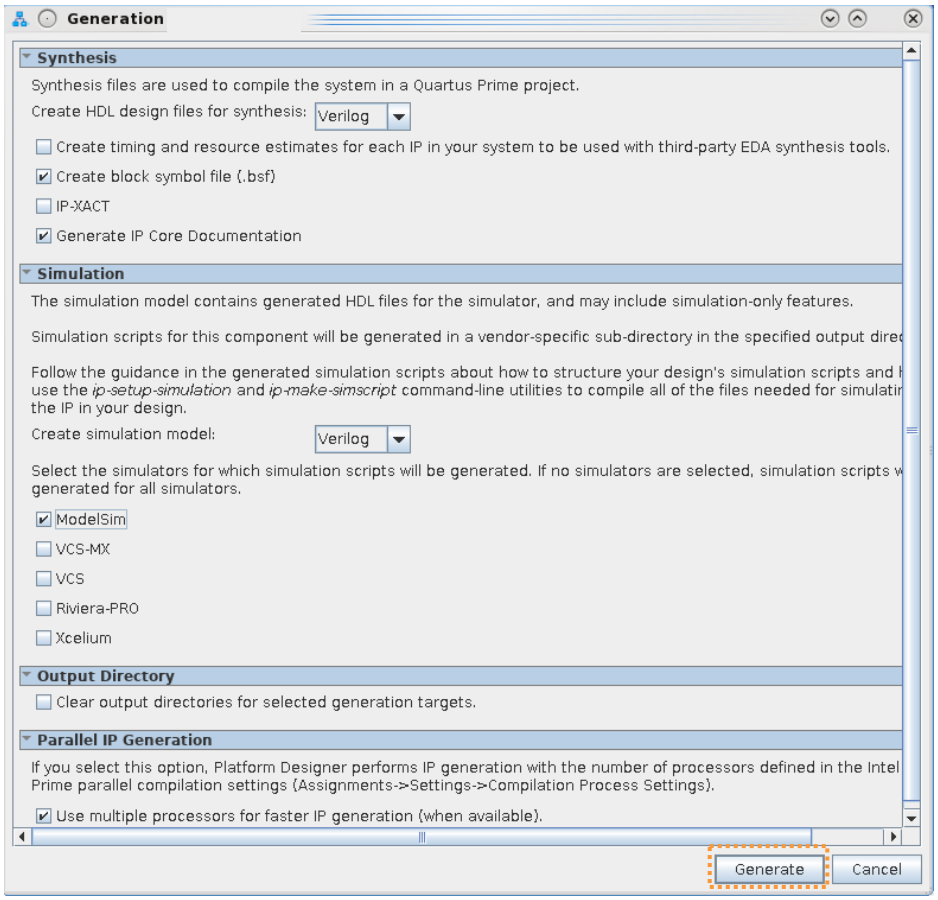
Platform Designer system generation creates the interconnect between IP components, and generates files for Quartus Prime synthesis and simulation in supported third-party tools.

If you make changes to your system, Platform Designer prompts you to generate your system before closing. Generate your system to ensure that the Compiler includes any changes you make to your system.

Follow these steps to generate a Platform Designer system:

1. Open a system in Platform Designer.
2. Consider whether to specify a unique generation ID, as Specifying the Generation ID on page 91 describes.

Figure 81. Platform Designer Generation Dialog Box



3. Click the **Generate HDL** button. The **Generation** dialog box appears.

4. Specify options for generation of **Synthesis**, **Simulation**, and testbench files, as Generation Dialog Box Options on page 90 describes.
5. Consider whether to specify options for **Parallel IP Generation**, as Disabling or Enabling Parallel IP Generation on page 92 describes.
6. To start system generation, click **Generate**.

   Note: Platform Designer may add unique suffixes (hashes) to ip component files during generation to ensure uniqueness of the file. The uniqueness of the files is necessary because the IP component is dynamic. The RTL generates during runtime, according to the input parameters. This methodology ensures no collisions between the multiple variants of the same IP. The hash derives from the parameter values that you specify. A given set of parameter values produces the same hash for each generation.

1.16.1. Generation Dialog Box Options

Platform Designer system generation creates files for Quartus Prime synthesis and supported third-party simulators. The **Generation** dialog box appears when you click **Generate HDL**, or when you attempt to close a system prior to generation.

By default, the synthesis and simulation files generate into the Platform Designer project directory.

You can specify the following system generation options in the **Generation** dialog box:

Table 20. Generation Dialog Box Options

Option	Description
Create HDL design files for synthesis	Allows you to specify **Verilog** or **VHDL** file type generation for the system's top-level definition and child instances. Select **None** to skip generation of synthesis files.
Create timing and resource estimates for each IP in your system to be used with third-party synthesis tools	Generates a non-functional Verilog Design File (`.v`) for use by supported third-party EDA synthesis tools. Estimates timing and resource usage for the IP component. The generated netlist file name is `<ip_component_name>_syn.v`.
Create Block Symbol File (.bsf)	Generates a Block Symbol File (`.bsf`) for use in a larger system schematic Block Diagram File (`.bdf`).
IP-XACT	Generates an IP-XACT file for the system, and adds the file to the IP Catalog. *Note:* Platform Designer supports importing and exporting files in IP-XACT 2009 format and exporting IP-XACT files in 2014 format.
Generate IP Core Documentation	Generates the IP user guide documentation for the components in your system (when available).
Create simulation model	Generates Verilog HDL or VHDL simulation models and simulator setup scripts. Enable the checkbox for one or more simulators to generate setup scripts for those tools in the following location: `<top-level system name>/<system name>/<sim>/<simulator>` If you do not specify a simulator, the setup scripts generate for all supported tools.
Clear output directories for selected generation targets	Clears previous synthesis and simulation file generation data for the current system.
Use multiple processors for faster IP generation (when available)	Disables or enables parallel IP generation for faster IP generation using multiple processors when available in your system.

Note: The Questa* Intel FPGA Edition simulator supports native, mixed-language (VHDL, Verilog HDL, SystemVerilog) simulation. Therefore, Intel simulation libraries may not be compatible with single language simulators. If you have a VHDL-only license, some versions of ModelSim* simulators may not support simulation for IPs written in Verilog. As a workaround, you can use the Questa Intel FPGA Edition simulator, or purchase a mixed-language simulation license from Siemens EDA.

Related Information

List of Supported Simulators

1.16.2. Specifying the Generation ID

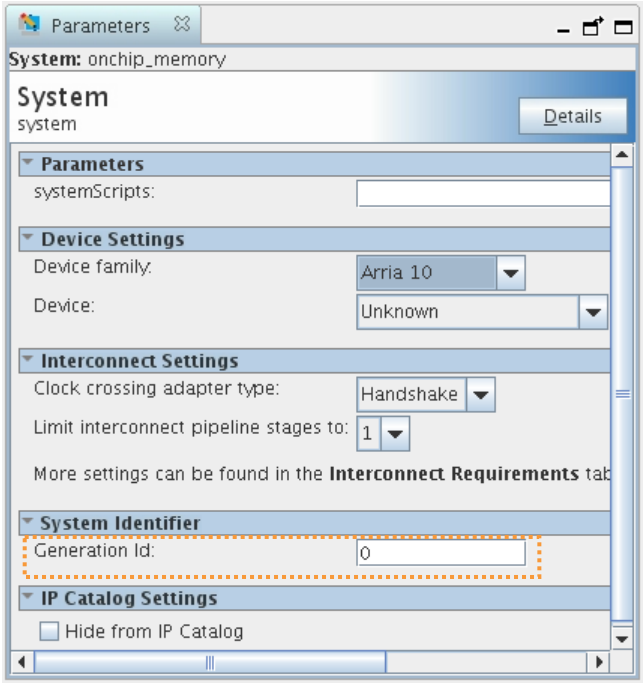
You can specify the **Generation ID** to uniquely identify that specific system generation number. This parameter allows system tools, such as Nios II processor, Nios V processor, or HPS (Hard Processor System) tools, to verify software-build compatibility with a specific Platform Designer system.

The **Generation ID** parameter is a unique integer value that derives from the timestamp during Platform Designer system generation. You can optionally modify this value to a value of your choosing to identify the system.

To specify the **Generation ID** parameter:

1. In the **Hierarchy** tab, select the top-level system.
2. Click **View ➤ Parameters**.
3. Under **System Identifier**, view or edit the value of **Generation ID**.

Figure 82. Generation ID in Parameters Tab



1.16.3. Disabling or Enabling Parallel IP Generation

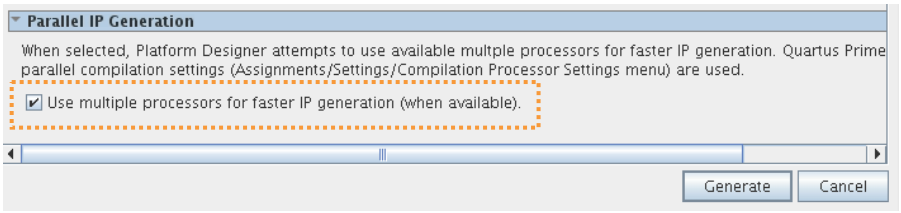
By default, the Quartus Prime software and Platform Designer use multiple processors if available in your PC or workstation for faster IP generation. IP generation for large systems can be time consuming. The use of parallel IP generation can potentially reduce the total IP generation time for designs with large numbers of IP.

The `qsys-generate` command line utility similarly uses parallel IP generation by default when multiple processors are available. You can disable or enable the use of parallel IP generation for the current IP generation, for the current project, or for all projects. You can also specify the maximum number of processors to use for parallel IP generation.

Disabling or Enabling Parallel IP Generation for the Current IP Generation

1. Open a system or IP component in Platform Designer, and click **Generate HDL**.
2. In the **Generation** dialog box, turn on or off **Use multiple processors for faster IP generation (when available)**. Platform Designer retains this setting for subsequent generations.

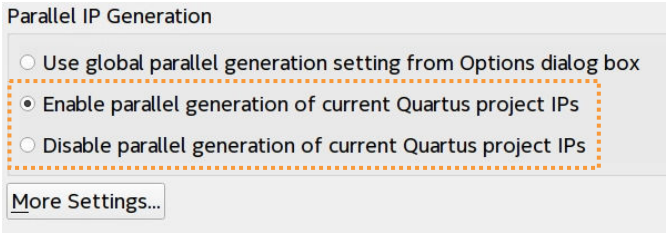
Figure 83. Disables or Enables Parallel IP Generation for the Current IP Generation



Disabling or Enabling Parallel IP Generation for a Single Project

1. In the Quartus Prime software, click **Assignments ➤ Settings ➤ Compilation Process Settings**.
2. Under **Parallel IP Generation**, select **Disable parallel generation of current Quartus project IPs** to disable parallel IP generation for the current project. Select **Enable parallel generation of current Quartus project IPs** to enable parallel IP generation for the current project.

Figure 84. Enables or Disables Parallel IP Generation for a Single Project



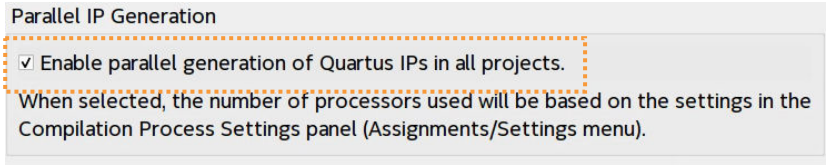
Alternatively, you can disable or enable parallel IP generation for a project with the following line in the project `.qsf` file:

```
set_global_assignment -name PROJECT_IP_GEN_PARALLEL_ENABLED <off|on>
```

Disabling or Enabling Parallel IP Generation for all Projects

1. In the Quartus Prime software, click **Tools ➤ Options ➤ IP Settings**.
2. Under **Parallel IP Generation**, enable or disable the **Enable parallel generation of Quartus IPs in all projects option**. When enabled, the Quartus Prime software uses multiple processors (if available in your system) for faster IP generation.

Figure 85. Enables or Disables Parallel IP Generation for all Projects



Alternatively, you can disable or enable parallel IP generation for all projects by adding the following line to the `quartus2.ini` file:

```
ENABLE_PARALLEL_IP_GEN=<off|on>
```

Specifying the Maximum Number of Processors

Parallel IP generation derives the maximum number of processors to use from the **Maximum processors allowed** Compiler setting. If you specify no value for this setting, the Quartus Prime software selects an appropriate number based on the available processors, and the number of tasks the processors can execute in parallel.

1. In the Quartus Prime software, click **Assignments ➤ Settings ➤ Compilation Process Settings**.
2. Under **Parallel compilation**, specify the **Maximum processors allowed** for processing designs.

Alternatively, you can set the number of processors with the following line in the project `.qsf` file:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS <number>
```

For the `qsys-generate` command line utility, you can use the `--parallel[=<number>]` argument, where <*number*> indicates the target number of processors.

Related Information

qsys-generate Command-Line Options on page 444

1.16.4. Files Generated for Platform Designer Systems

The Quartus Prime Pro Edition software generates the following output file structure for Platform Designer systems. Platform Designer automatically adds the generated `.qsys` system file to the open Quartus Prime project following generation.

Figure 86. Files generated for Platform Designer Systems

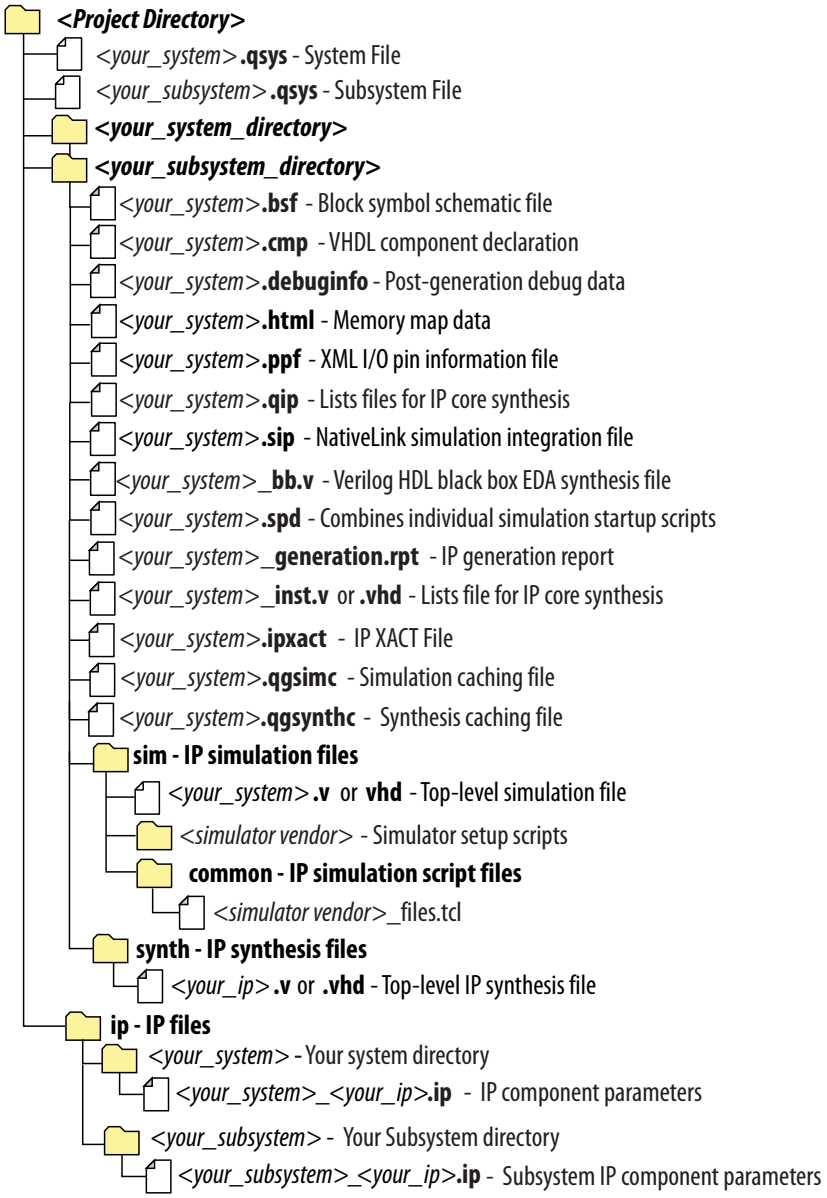


Table 21. Files Generated for Platform Designer Systems

File Name	Description
<your_system>.qsys	The Platform Designer system.
<your_subsystem>.qsys	The Platform Designer subsystem.
ip/	Contains the parameter files for the IP components in the system and subsystems.
<your_ip>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you can use in VHDL design files.

continued...

File Name	Description
<*your_system*>_generation.rpt	IP or Platform Designer generation log file. A summary of the messages during IP generation.
<*your_system*>.qgsimc	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_system*>.qgsynth	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_system*>.qip	Contains all the required information about the IP component to integrate and compile the IP component in the Quartus Prime software.
<*your_system*>.csv	Contains information about the upgrade status of the IP component.
your_system.bsf	A Block Symbol File (.bsf) representation of the IP variation for use in Block Diagram Files (.bdf).
<*your_system*<>.spd	Required input file for ip-make-simscript to generate simulation scripts for supported simulators. The .spd file contains a list of files generated for simulation, along with information about memories that you can initialize.
<*your_system*>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components created for use with the Pin Planner.
<*your_system*>_bb.v	Use the Verilog black box (_bb.v) file as an empty module declaration for use as a black box.
<*your_system*>.sip	Contains information required for NativeLink simulation of IP components. Add the .sip file to your Quartus Prime Standard Edition project to enable NativeLink for supported devices. The Quartus Prime Pro Edition software does not support NativeLink simulation.
<*your_system*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*your_system*>.regmap	If the IP contains register information, the Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of host and agent interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*your_system*>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Platform Designer system. During synthesis, the Quartus Prime software stores the .svd files for agent interface visible to the System Console hosts in the .sof file in the debug session. System Console reads this section, which Platform Designer can query for register map information. For system agents, Platform Designer can access the registers by name.
<*your_system*>.v <*your_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a ModelSim script msim_setup.tcl to set up and run a simulation.
aldec/	Contains a Riviera-PRO* script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS* simulation. Contains a shell script vcsmx_setup.sh and synopsys_ sim.setup file to set up and run a VCS MX simulation.
/cadence	Contains a shell script ncsim_setup.sh and other setup files to set up and run an NCSIM simulation.

continued...

File Name	Description
/xcelium	Contains a shell script xcelium_setup.sh and other setup files to set up and run a Xcelium* simulation.
/common	Contains a set of Tcl files, <simulator>_files.tcl, which provide all design related simulation information required by a corresponding simulation script. The Tcl file contains designs from current system-level hierarchy, and references to sub-systems and IP components.
/submodules	Contains HDL files for the IP core submodule.
<IP submodule>/	For each generated IP submodule directory, Platform Designer generates /synth and /sim sub-directories.

For generated IP components, Platform Designer appends unique suffixes (hashes) to the IP component's RTL file name to ensure uniqueness of the RTL file and IP component file. The uniqueness of the files is necessary because a system can have multiple instances of the same IP, each with different parameterizations, resulting in multiple variances of the IP component. The hash derives from the parameterization that you specify for the IP component. This methodology ensures no collisions between the multiple variants of the same IP.

1.16.5. Generating System Testbench Files

Platform Designer can generate testbench files that instantiate the current Platform Designer system and add Bus Functional Models (BFMs) to drive the top-level interfaces. BFMs interact with the system in the simulator.

You can generate a standard or simple testbench system with BFM or Mentor Verification IP (for AMBA 3 AXI or AMBA 4 AXI) components that drive the external interfaces of the system. Platform Designer generates a Verilog HDL or VHDL simulation model for the testbench system to use in the simulation tool.

First generate a testbench system, and then modify the testbench system in Platform Designer before generating the simulation model. Typically, you select only one of the simulation model options.

Follow these steps to generate system testbench files:

1. Open and configure a system in Platform Designer.
2. Click **Generate ➤ Generate Testbench System**. The **Generation** dialog box appears.
3. Specify options for the test bench system, as Testbench Generation Options describes.
4. Click **Generate**. The testbench files generate according to your specifications.
5. Open the testbench system in Platform Designer. Make changes to the BFMs, as needed, such as changing the instance names and **VHDL ID** value. For example, you can modify the **VHDL ID** value in the **Avalon Interrupt Source Intel FPGA IP** component.
6. If you modify a BFM, regenerate the simulation model for the testbench system.
7. Compile the system and load the Platform Designer system and testbench into your simulator, and then run the simulation.

Figure 87. Platform Designer Simulation Testbench Directory Structure

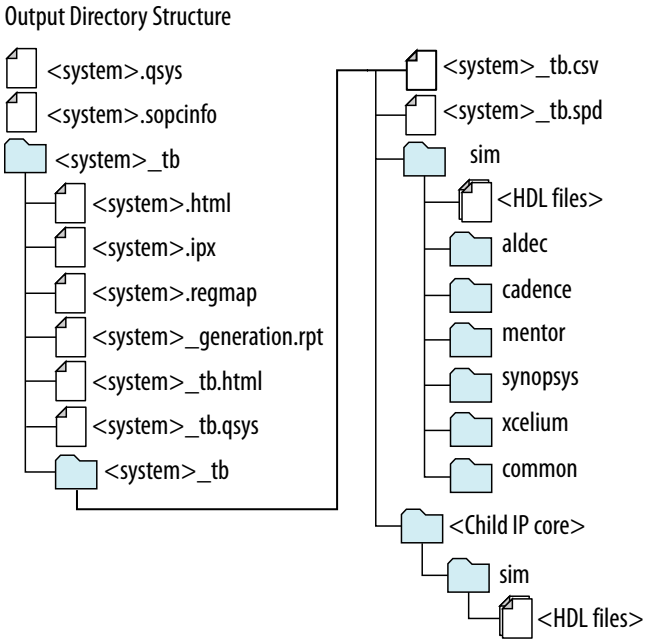


Table 22. Testbench Generation Options

Option	Description
Create testbench Platform Designer system	Specifies a simple or standard testbench system: • **Standard, BFMs for standard Platform Designer Interconnect**—Creates a testbench Platform Designer system with BFM IP components attached to exported Avalon and AMBA 3 AXI or AMBA 3 AXI interfaces. Includes any simulation partner modules specified by IP components in the system. The testbench generator supports AXI interfaces and can connect AMBA 3 AXI or AMBA 3 AXI interfaces to Mentor Graphics AMBA 3 AXI or AMBA 3 AXI manager and subordinate BFMs. However, BFMs support address widths only up to 32-bits. • **Simple, BFMs for clocks and resets**—Creates a testbench Platform Designer system with BFM IP components driving only clock and reset interfaces. Includes any simulation partner modules specified by IP components in the system.
Create testbench simulation model	Specifies Verilog HDL or VHDL simulation model files and simulation scripts for the testbench. Use this option if you do not need to modify the Platform Designer-generated testbench before running the simulation.
Output directory	Specifies the path for output of generated testbench files. Turn on **Clear output** to remove any previously generated content from the location.
Parallel IP Generation	Turn on **Use multiple processors for faster IP generation (when available)** to generate IP using multiple CPUs when available in your system.

1.16.5.1. Platform Designer Testbench Files

Platform Designer generates the following testbench files.

Table 23. Platform Designer Testbench Files

File Name or Directory Name	Description
<*system*>_tb.qsys	The Platform Designer testbench system.
<*system*>_tb.v or <*system*>_tb.vhd	The top-level testbench file that connects BFMs to the top-level interfaces of <*system*>_*tb*.qsys.
<*system*>_tb.spd	Required input file for `ip-make-simscript` to generate simulation scripts for supported simulators. The `.spd` file contains a list of files generated for simulation and information about memory that you can initialize.
<*system*>.html and <*system*>_tb.html	A system report that contains connection information, a memory map showing the address of each agent with respect to each host to which it is connected, and parameter assignments.
<*system*>_generation.rpt	Platform Designer generation log file. A summary of the messages that Platform Designer issues during testbench system generation.
<*system*>.ipx	The IP Index File (`.ipx`) lists the available IP components, or a reference to other directories to search for IP components.
<*system*>.svd	Allows HPS System Debug tools to view the register maps of peripherals connected to HPS within a Platform Designer system. Similarly, during synthesis the `.svd` files for agent interfaces visible to System Console hosts are stored in the `.sof` file in the debug section. System Console reads this section, which Platform Designer can query for register map information. For system agents, Platform Designer can access the registers by name.
mentor/	Contains a ModelSim script `msim_setup.tcl` to set up and run a simulation
aldec/	Contains a Riviera-PRO script `rivierapro_setup.tcl` to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script `vcs_setup.sh` to set up and run a VCS simulation. Contains a shell script `vcsmx_setup.sh` and `synopsys_ sim.setup` file to set up and run a VCS MX simulation.
/cadence	Contains a shell script `ncsim_setup.sh` and other setup files to set up and run an NCSIM simulation.
/xcelium	Contains a shell script `xcelium_setup.sh` and other setup files to set up and run an Xcelium simulation.
/common	Contains a set of Tcl files, `<simulator>_files.tcl`, which provide all design related simulation information required by a corresponding simulation script. The Tcl file contains designs from current system-level hierarchy, and references to sub-systems and IP components.
/submodules	Contains HDL files for the submodule of the Platform Designer testbench system.
<*child IP cores*>/	For each generated child IP core directory, Platform Designer testbench generates `/synth` and `/sim` subdirectories.

1.16.6. Generating Example Designs for IP Components

Some Platform Designer IP components include example designs that you can use or modify to replicate similar functionality in your own system. You must generate the examples to view or use them.

Use any of the following methods to generate example designs for IP components:

- Double-click the IP component in the Platform Designer IP Catalog or **System View** tab. The parameter editor for the component appears. If available, click the **Example Design** button in the parameter editor to generate the example design. The **Example Design** button only appears in the parameter editor if an example is available.
- For some IP components, click **Generate ➤ Generate Example Design** to access an example design. This command only enables when a design example is available.

The following Platform Designer system example designs demonstrate various design features and flows that you can replicate in your Platform Designer system.

Related Information

Intel FPGA Design Example Web Page

1.16.7. Incremental System Generation Example

You can modify the parameters of an IP component and regenerate the RTL for just that particular IP component.

The following example demonstrates incremental generation of a Platform Designer System:

1. Create a new Platform Designer system, as Creating or Opening a Platform Designer System on page 14 describes.
2. Use the IP Catalog to locate and add the **On-Chip Memory (RAM or ROM)** **Reset Bridge**, and **Clock Bridge** components to the system, as Adding IP Components to a System on page 40 describes.
3. Make the necessary system connections between the IP components added to the system, as Connecting System Components on page 63 describes.
4. To save and close the system without generating, click **File ➤ Save** and close Platform Designer.
5. In the Quartus Prime software, click **File ➤ Open Project**.
6. Select the Quartus Prime project associated with your saved Platform Designer system. The Quartus Prime software opens the project and the associated Platform Designer system.
7. To start the compilation of the Quartus Prime project, click **Processing ➤ Start Compilation**.
8. After compilation completes, in Platform Designer, click **File ➤ Open**.
9. Select the `.ip` file for any one of the IP components in your saved system.
10. Modify some parameter in this `.ip` file.

Note: Make sure your modifications do not affect the parent system, requiring a system update by running **Validate System Integrity** from within the Platform Designer system after loading the parent system, or by running `qsys-validate` from the command-line.

11. To save the IP file, click **File ➤ Save**.
12. To restart the compilation of the same Quartus Prime project with modified Platform Designer system, click **Processing ➤ Start Compilation** in the Quartus Prime software. Platform Designer generates the RTL only for the modified IP component, skipping the generation of the other components in the system.

1.16.8. Generating the HPS IP Component System View Description File

Platform Designer systems that contain an HPS IP component generate a System View Description (`.svd`) file that lists peripherals connected to the ARM® processor.

The `.svd` (or CMSIS-SVD) file format is an XML schema specified as part of the Cortex Microcontroller Software Interface Standard (CMSIS) that ARM provides. The `.svd` file allows HPS system debug tools (such as the DS-5 Debugger) to view the register maps of peripherals connected to HPS in a Platform Designer system.

Related Information

- Component Interface Tcl Reference on page 666
- CMSIS - Cortex Microcontroller Software

1.16.9. Generating Header Files for Host Components

You can use the `sopc-create-header-files` command from the Nios II command shell to create header files for any host component in your Platform Designer system. The Nios II tool chain uses this command to create the processor's `system.h` file. You can also use this command to generate system level information for a hard processing system (HPS) in Intel's SoC devices or other external processors. The header file includes address map information for each agent, relative to each host that accesses the agent. Different hosts may have different address maps to access a particular agent component. By default, the header files are in C format and have a `.h` suffix. You can select other formats with appropriate command-line options.

Table 24. sopc-create-header-files Command-Line Options

Option	Description
`<sopc>`	Path to Platform Designer `.sopcinfo` file, or the file directory. If you omit this option, the path defaults to the current directory. If you specify a directory path, you must make sure that there is a `.sopcinfo` file in the directory.
`--separate-hosts`	Does not combine a module's hosts that are in the same address space.
`--output-dir[=<dirname>]`	Allows you to specify multiple header files in *dirname*. The default output directory is '.'
`--single[=<filename>]`	Allows you to create a single header file, *filename*.
`--single-prefix[=<prefix>]`	Prefixes macros from a selected single host.
`--module[=<moduleName>]`	Specifies the module name when creating a single header file.

continued...

Option	Description
`--host[=<hostName>]`	Specifies the host name when creating a single header file.
`--format[=<type>]`	Specifies the header file format. Default file format is `.h`.
`--silent`	Does not display normal messages.
`--help`	Displays help for `sopc-create-header-files`.

By default, the `sopc-create-header-files` command creates multiple header files. There is one header file for the entire system, and one header file for each host group in each module. A host group is a set of hosts in a module in the same address space. In general, a module may have multiple host groups. Addresses and available devices are a function of the host group.

Alternatively, you can use the `--single` option to create one header file for one host group. If there is one CPU module in the Platform Designer system with one host group, the command generates a header file for that CPU's host group. If there are no CPU modules, but there is one module with one host group, the command generates the header file for that module's host group.

You can use the `--module` and `--host` options to override these defaults. If your module has multiple host groups, use the `--host` option to specify the name of a host in the desired host group.

Table 25. Supported Header File Formats

Type	Suffix	Uses	Example
h	`.h`	C/C++ header files	`#define FOO 12`
m4	`.m4`	Macro files for m4	`m4_define("FOO", 12)`
sh	`.sh`	Shell scripts	`FOO=12`
mk	`.mk`	Makefiles	`FOO := 12`
pm	`.pm`	Perl scripts	`$macros{FOO} = 12;`

Note: You can use the `sopc-create-header-files` command when you want to generate C macro files for DMAs that have access to memory that the Nios II processor cannot access.

1.17. Generating Simulation Files for Platform Designer Systems and IP Variants

If your design contains Intel FPGA IP or a Platform Designer system, you must first generate files for RTL simulation of the IP or system with the Quartus Prime Platform Designer before running simulation.

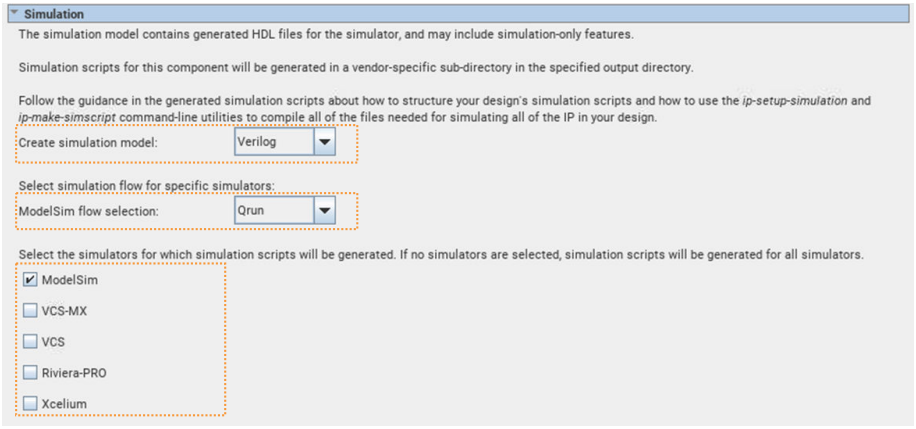
When you generate the system (or IP variant), Platform Designer optionally creates simulation files, including the functional simulation model, any testbench (or example design), and vendor-specific simulator setup scripts for each IP core.

You can use the functional simulation model and any testbench or example design for simulation of the IP or system. The IP generation output may also include scripts to compile and run any testbench. The scripts list all models or libraries you require to simulate your IP core.

To generate the simulation model and simulator setup scripts for your Platform Designer system or component, follow these steps:

1. Click **Tools ➤ Platform Designer**. Platform Designer and open or create a Platform Designer system or IP variant.
2. In Platform Designer, after specifying parameters, click **Generate ➤ Generate HDL**. The **Generation** dialog box appears.
3. Under **Simulation**, specify **Verilog** or **VHDL** for the **Create simulation model** option.

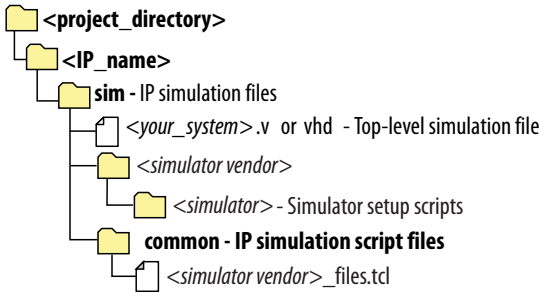
Figure 88. Simulation Options in Generation Dialog Box



4. If you want to specifically use ModelSim, specify **Traditional** or **Qrun** for the ModelSim flow option. Otherwise, **Qrun** flow is the default selection.
5. Turn on or off the **ModelSim**, **VCS-MX**, **VCS**, **Riviera-Pro**, or **Xcelium** option to generate simulator setup scripts for the simulation tool. If you turn on no simulator options, the scripts generate for all simulators.
6. Click the **Generate** button. Platform Designer generates the simulation models and setup scripts for your system or IP component in the following directory:

```
<top-level system name>/<system name>/<sim>/<simulator>
```

Figure 89. Generated Simulation Files Location



By default, Platform Designer generates the simulation scripts for the currently loaded system and all subsystems. Alternatively, you can open a subsystem to generate a simulation script only for that subsystem.

You can use scripts to compile the required device libraries and system design files in the correct order and elaborate or load the top-level system for simulation.

Table 26. Simulation Script Variables

The simulation scripts provide variables that allow flexibility in your simulation environment.

Variable	Description
`TOP_LEVEL_NAME`	If the testbench Platform Designer system is not the top-level instance in your simulation environment because you instantiate the Platform Designer testbench within your own top-level simulation file, set the `TOP_LEVEL_NAME` variable to the top-level hierarchy name.
`QSYS_SIMDIR`	If the simulation files generated by Platform Designer are not in the simulation working directory, use the `QSYS_SIMDIR` variable to specify the directory location of the Platform Designer simulation files.
`QUARTUS_INSTALL_DIR`	Points to the Quartus installation directory that contains the device family library.

Example 3. Top-Level Simulation HDL File for a Testbench System

The example below shows the `pattern_generator_tb` generated for a Platform Designer system called `pattern_generator`. The `top.sv` file defines the top-level module that instantiates the `pattern_generator_tb` simulation model, as well as a custom SystemVerilog test program with BFM transactions, called `test_program`.

```
module top();
 pattern_generator_tb tb();
 test_program pgm();
endmodule
```

Related Information

qsys-generate Command-Line Options on page 444

1.17.1. Using the Qrun Flow

The Quartus Prime Pro Edition software now supports a new Qrun flow for IP generation. The Qrun flow optionally creates simulation files, including the functional simulation model, and any testbench (or example design).

The Qrun flow, for use with only the QuestaSim* and Questa Intel FPGA Edition simulators, is an enhancement over the traditional flow that can automatically combine the compile, optimize, and simulate functions into a single step.

This section describes how to specify the Qrun settings, generate the system or component simulation model and simulator setup scripts, and generate the testbench and example design.

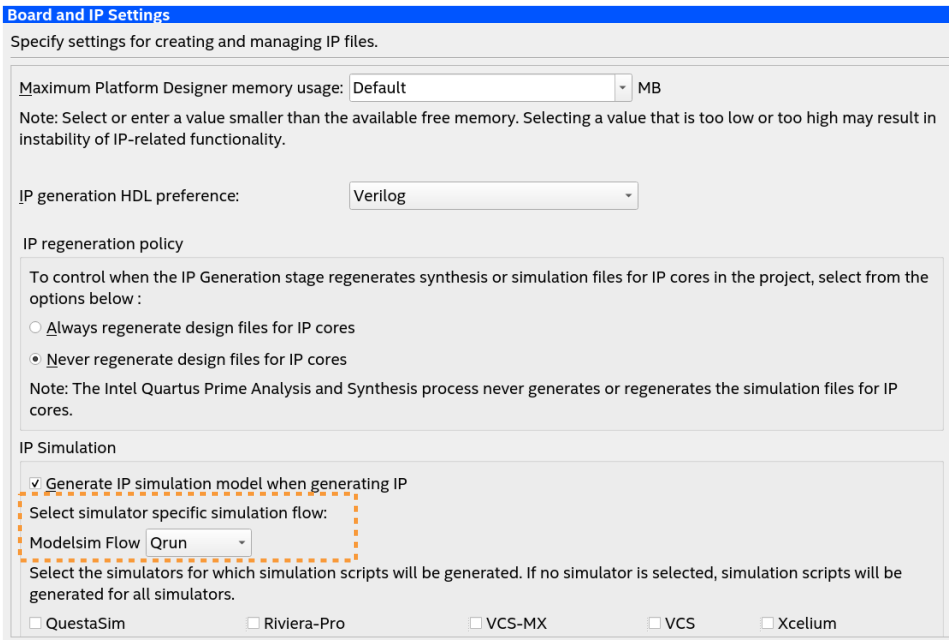
1.17.1.1. Specifying Simulation File Generation Settings

Before generating simulation files using the Qrun flow, you specify your supported simulator and other options for simulation file generation. These setting impact the generation of simulation files when generating HDL for IP in your project.

To specify simulation file generation settings, follow these steps:

1. In the Quartus Prime Pro Edition software, click **Assignments ➤ Settings ➤ Board and IP Settings**. The **Board and IP Settings** dialog box appears.
2. Under **IP Simulation**, turn on **Generate IP simulation model when generating IP**. Turning on this option enables the remaining settings.
3. For **Select simulator specific simulation flow**, make sure **Qrun** is selected to enable the Qrun flow. The alternative setting runs the **Traditional** flow.

Figure 90. Board and IP Settings Page of Settings Dialog Box



4. To specify one or more specific simulators for which to generate simulation files, enable the checkbox for those simulators. To enable generation for all supported simulators, leave all checkboxes disabled (default setting).

To generate the simulation model and simulator setup scripts for your Platform Designer system or IP component in batch mode, use this command:

```
ip-make-simscript [args] --modelsim_flow=QRUN
```

Type `ip-make-simscript -help` for all available arguments (`[args]`).

1.17.1.2. Generating the Simulation Model and Setup Scripts

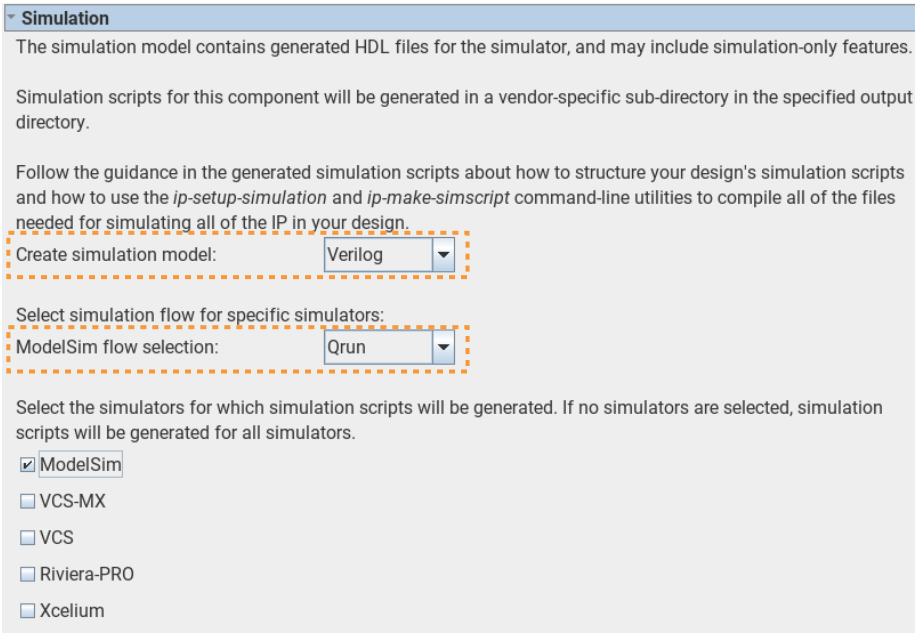
You generate the simulation model and setup scripts for IP components and Platform Designer systems when generating HDL for these IP in your project.

Platform Designer generates the simulation model and setup scripts according to your specifications in Specifying Simulation File Generation Settings.

To generate the simulation model and simulator setup scripts for your Platform Designer system or IP component, follow these steps:

1. In the Quartus Prime Pro Edition software, click **Tools ➤ Platform Designer** and open or create an IP variant or Platform Designer system.
2. After specifying any IP component or system parameters in the parameter editor, click the **Generate HDL** button. The **Generation** dialog box appears.
3. Under **Simulation**, select either **Verilog** or **VHDL** for **Create Simulation Model**. Selecting one of these options makes the **Modelsim flow selection** setting editable.
4. For **Modelsim flow selection**, make sure **Qrun** is selected to enable the Qrun flow. The alternative setting runs the **Traditional** flow.

Figure 91. Generation Dialog Box Settings



5. To specify one or more specific simulators for which to generate simulation files, enable the checkbox for those simulators. To enable generation for all supported simulators, leave all checkboxes disabled (default setting).
6. Click **Generate**. Platform Designer generates the simulation models and setup scripts for your system or IP component in the *<project>*/*<IP name>*/sim/<vendor> directory.

To generate the simulation model and simulator setup scripts for your Platform Designer system or IP component in batch mode, use this command:

```
qsys-generate <file> [args] --modelsim_flow=QRUN
```

1.17.1.3. Generating the Testbench System

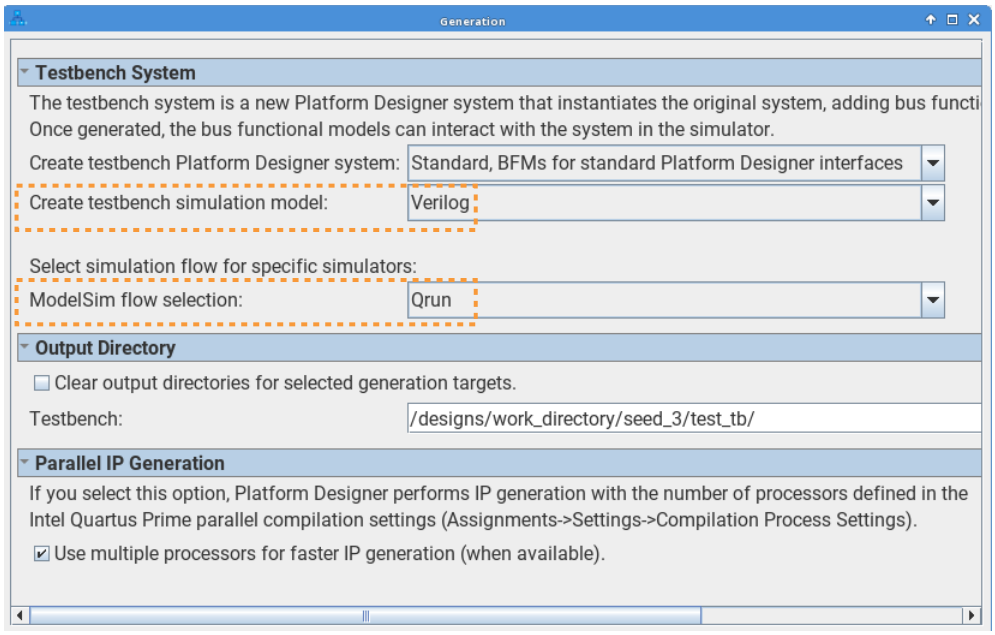
You can optionally generate a testbench system that instantiates the original system, adding bus functional models to drive the top-level interfaces. Once generated, the bus functional models can interact with the system or IP in the simulator.

Platform Designer generates the simulation model and setup scripts according to your specifications in Specifying Simulation File Generation Settings.

To generate the testbench system for a Platform Designer system or IP component, follow these steps:

1. In the Quartus Prime Pro Edition software, click **Tools ➤ Platform Designer** and open or create an IP variant or Platform Designer system.
2. After specifying any IP component or system parameters in the parameter editor, click the **Generate ➤ Generate Testbench System** button. The **Generation** dialog box appears.
3. Under **Testbench System**, select either **Verilog** or **VHDL** for **Create testbench simulation model**. Selecting one of these options makes the **Modelsim flow selection** setting editable.
4. For **Modelsim flow selection**, make sure **Qrun** is selected to enable the Qrun flow. The alternative setting runs the **Traditional** flow.

Figure 92. Generation Dialog Box Settings



5. Click **Generate**. Platform Designer generates the simulation models and setup scripts for your system or IP component under the specified **Output Directory**.

1.17.1.4. Generating Example Design Simulation Files

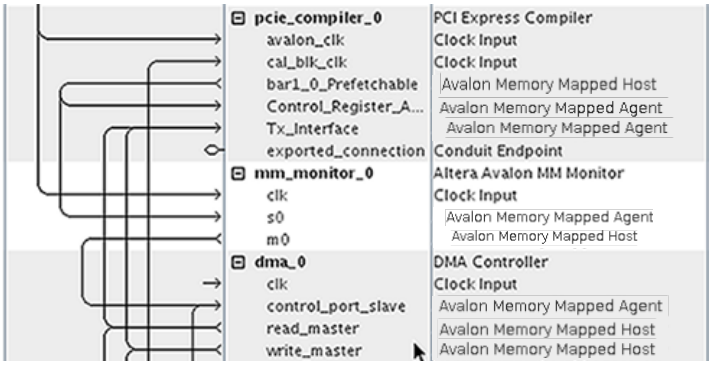
When you run **Generate ➤ Generate Example Design**, Platform Designer automatically generates the simulator setup script `msim_setup.tcl` containing `qrun` commands.

1.17.2. Adding Assertion Monitors for Simulation

You can add monitors to Avalon Memory-Mapped, AXI, and Avalon Streaming interfaces in your system to verify protocol and test coverage with a simulator that supports SystemVerilog assertions.

Figure 93. Inserting an Avalon Memory-Mapped Monitor Between an Avalon Memory-Mapped Host and Agent Interface

This example demonstrates the use of a monitor with an Avalon Memory-Mapped monitor between the `pcie_compiler bar1_0_Prefetchable` Avalon Memory-Mapped host interface, and the `dma_0 control_port_agent` Avalon Memory-Mapped agent interface.



Similarly, you can insert an Avalon Streaming monitor between Avalon Streaming source and sink interfaces.

Note: Refer to the *Nios V Processor Quick Start Guide* for information on simulating Nios V processor designs.

Related Information

Nios V Embedded Processor Design Handbook

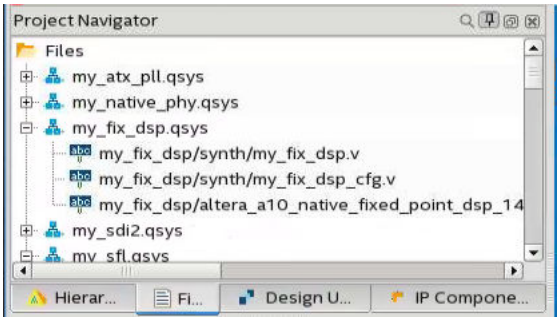
1.18. Adding a System to an Quartus Prime Project

Platform Designer requires that you to specify an associated Quartus Prime project at time of system creation. After you specify the associated project, Platform Designer automatically adds any system or IP component that you generate to that project. You can also manually add a Platform Designer system or component to a project.

To add a Platform Designer system or component to an Quartus Prime project, perform one or more of the following steps:

1. In Platform Designer, specify the associated **Quartus project** when you create a system, or click **File ➤ Select Quartus Project** to change this setting. Platform Designer automatically adds any system or IP component that you generate to the associated Quartus Prime project.
2. To manually add a Platform Designer system or component to your project, generate the system or component, and then click **Project ➤ Add/Remove Files in Project** in the Quartus Prime software.
3. Select and add the `.qsys` files to your project. The Quartus Prime Project Navigator **Files** tab lists all system and component files that you or Platform Designer add to your project.

Figure 94. Platform Designer System Files in Project



1.19. Managing Hierarchical Platform Designer Systems

Platform Designer supports hierarchical systems that include one or more Platform Designer subsystems within another Platform Designer system. Platform Designer allows you to create, explore, and edit systems and subsystems together in the same Platform Designer window. Platform Designer generates the complete system hierarchy during the top-level system's generation.

All hierarchical Platform Designer systems appear in the IP Catalog under **Project ➤ System**. You select the system from the IP Catalog to reuse the system across multiple designs. In a team-based hierarchical design flow, you can divide large designs into subsystems and allow team members develop subsystems simultaneously.

Related Information

Viewing the System Hierarchy on page 30

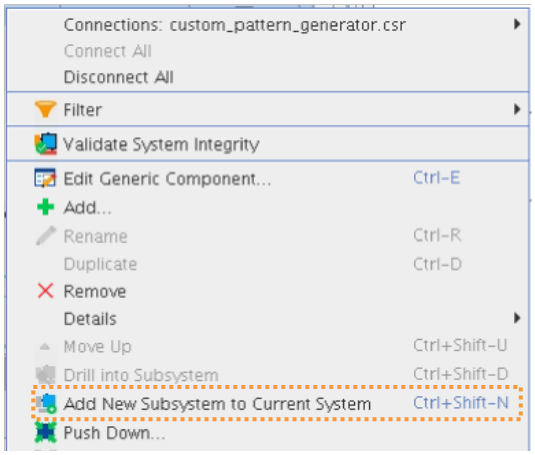
1.19.1. Adding a Subsystem to a Platform Designer System

You can add a Platform Designer system as a subsystem (child) of another Platform Designer system (parent), at any level in the parent system hierarchy.

Follow these steps to add a subsystem to a Platform Designer system:

1. Create a Platform Designer system to use as the subsystem.
2. Open a Platform Designer system to contain the subsystem.
3. On the **System View** tab, use any of the following methods to add the subsystem:
   - Right-click anywhere in the **System View** and click **Add a new subsystem to the current system**.
   - Click the **Add a new subsystem to the current system** button on the toolbar.
   - Press Ctrl+Shift+N.
4. In the **Confirm New System Name** dialog box, confirm or specify the new system file name and click **OK**. The system appears as a new subsystem in the **System View**.

Figure 95. Adding a Subsystem



1.19.2. Viewing and Traversing Subsystem Contents

You can view and traverse the elements and connections within subsystems in a hierarchical Platform Designer system.

Follow these steps to view and traverse subsystem contents:

1. Open a Platform Designer system that contains a subsystem.
2. Use any of the following methods to view the subsystem contents:
   - Use the filtering controls in the **Filter** tab to change the level of detail in the **System View**.
   - Right-click a system in the **System View** or **Schematic** tabs, and then select **Drill into Subsystem**. The subsystem opens in the **System View**.
   - Press Ctrl+Shift+D in the **System View** tab.
3. Use any of the following **System View** or **Schematic** tab toolbar buttons to traverse the system and subsystems:

Figure 96. Traversing Subsystem Contents

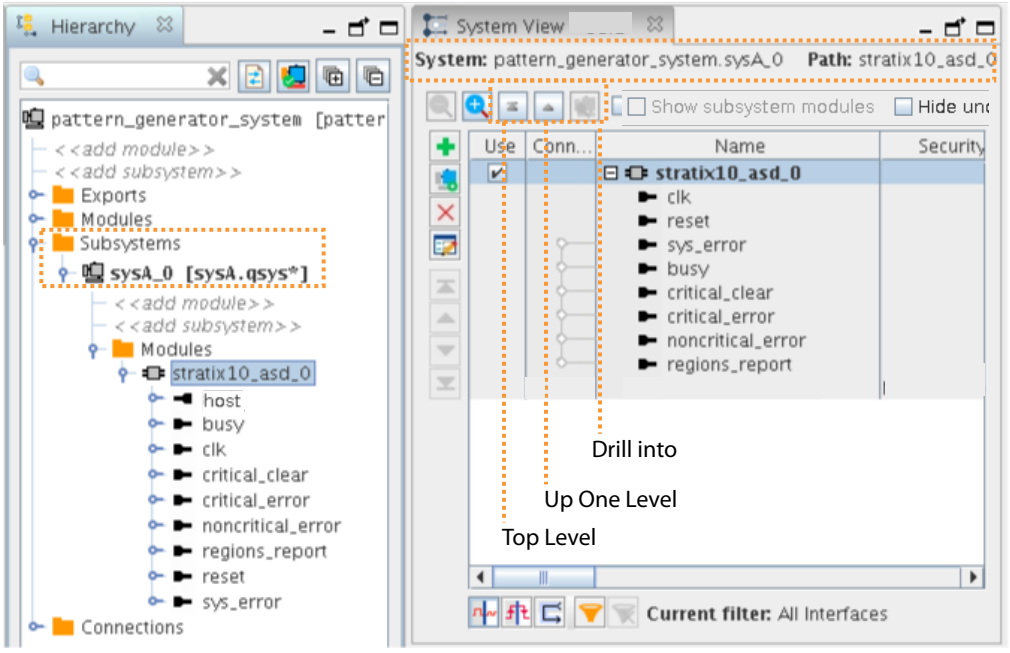


Table 27. System View and Schematic Tab Navigation Buttons

Button	Description
	**Move to the top of the hierarchy**—navigates to the top-level (parent) `.qsys` file for the system.
	**Move up one level of hierarchy**—navigates up one hierarchy level from the current selection.
	**Drill into a subsystem to explore its contents**—opens the subsystem you select in the **System View**.

Note: You can only view and traverse the contents of subsystems that you define in a `.qsys` file, not parameterizable Platform Designer systems or `_hw.tcl` files.

1.19.3. Editing a Subsystem

You can double-click a Platform Designer subsystem in the **Hierarchy** tab to edit its contents in any tab. When you make a change, open tabs refresh their content to reflect your edit. You can change the level of a subsystem, or push the system into another subsystem with commands in the **System View** tab.

Note: You can only edit subsystems that a writable `.qsys` file preserves. You cannot edit systems that you create from composed `_hw.tcl` files, or systems that define instance parameters.

Follow these steps to edit a Platform Designer subsystem:

1. Open a Platform Designer system that contains a subsystem.
2. In the **System View** or **Schematic** tabs, use the **Move Up**, **Move Down**, **Move to Top**, and **Move to Bottom** toolbar buttons to navigate the system level you want to edit. Platform Designer updates to reflect your selection.
3. To edit a system, double-click the system in the **Hierarchy** tab. The system opens and is available for edit in all Platform Designer views.
4. In the **System View** tab, you can rename any element, add, remove, or duplicate connections, and export interfaces, as appropriate.

   Note: Changes to a subsystem affect all instances. Platform Designer identifies unsaved changes to a subsystem with an asterisk next to the subsystem in the **Hierarchy** tab.

1.19.4. Saving a Subsystem

When you save a subsystem as part of a Platform Designer system, Platform Designer confirms the new subsystem name in the **Confirm New System Filenames** dialog box. By default, Platform Designer suggests the same name as the subsystem `.qsys` file and saves in the project's `/ip` directory.

Follow these steps to save a subsystem:

1. Open a Platform Designer system that contains a subsystem.
2. Click **File ➤ Save** to save your Platform Designer design.
3. In the **Confirm New System Filenames** dialog box, click **OK** to accept the subsystem file names.

   Note: If you have not yet saved your top-level system, or multiple subsystems, you can type a new name, and then press **Enter**, to move to the next un-named system.
4. In the **Confirm New System Filenames** dialog box, to edit the name of a subsystem, click the subsystem, and then type the new name.

1.19.5. Changing a Component's Hierarchy Level

You can change the hierarchical level of components in your system.

You can lower the hierarchical level of a component, even into its own subsystem, which can simplify the top-level system view. You can also raise the level of a component or subsystem to share the component or subsystem between two unique subsystems. Management of hierarchy levels facilitates system optimization and can reduce complex connectivity in your subsystems.

Follow these steps to change a component's hierarchy level:

1. Open a Platform Designer system that contains a subsystem.
2. In the **System View** tab, to group and change the hierarchy level of multiple components that share a system-level component, multi-select the components, right-click, and then click **Push down into new subsystem**. Platform Designer pushes the components into their own subsystem and re-establishes the exported signals and connectivity in the new location.
3. In the **System View** tab, to pull a component up out of a subsystem, select the component, and then click **Pull up**. Platform Designer pulls the component up out of the subsystem and re-establishes the exported signals and connectivity in the new location.

1.20. Saving and Archiving Platform Designer Systems

You can save your system in Platform Designer for later modification and reuse. Alternatively, Platform Designer can archive your entire system into a single compressed `.zip` or Tcl script (`.tcl`) file for easy storage and restoration.

If saving a Platform Designer system for external revision control systems, saving as a Tcl script may be preferred because that creates a single, clear text file that you can diff with a previous version to indicate changes.

You can also archive an entire Quartus Prime project, including any Platform Designer system or IP, within in a single, compressed Quartus Prime Archive File (`.qar`). The `.qar` preserves the project and setting files, design files, IP and system files, programming files, and all other files needed to restore the project (and system).

The `qsys-archive` utility provides command-line options for system archive, including system extraction and system dependency reporting. The following describe Platform Designer system saving and archiving in detail:

- Saving Platform Designer Systems on page 112
- Archiving Platform Designer Systems on page 113
- Including Platform Designer Systems in Project Archives on page 115
- Project Files to Include In External Revision Control on page 116
- Archive and Extract Platform Designer Systems with qsys-archive on page 457

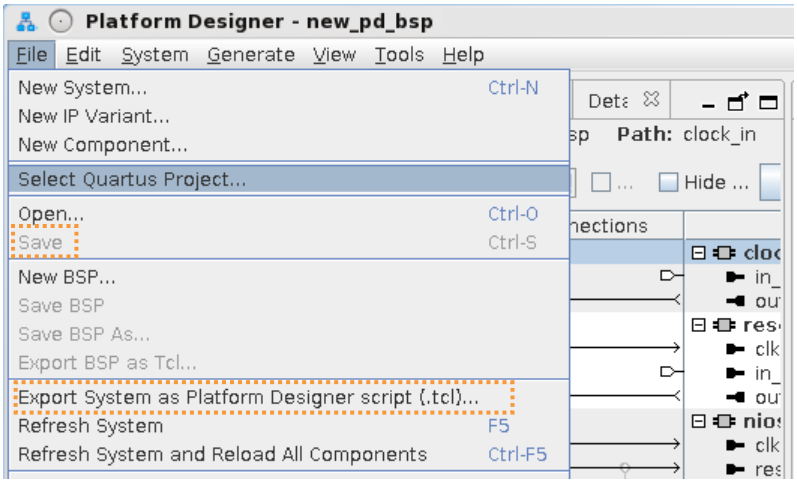
1.20.1. Saving Platform Designer Systems

Platform Designer prompts you to save any changes to your system before generating or closing. You can use any of the following methods to save your Platform Designer system:

- To save a Platform Designer system, click **File ➤ Save** in Platform Designer.
- To save a Platform Designer system as a Tcl script, click **File ➤ Export System as Platform Designer script**.

Note: The generated Tcl scripts also include any pin constraints that you set by applying presets to a Platform Designer system.

Figure 97. Saving A Platform Designer System



Restore this system by executing the `.tcl` file from the **System Scripting** tab, or with the `qsys-script` command.

Note: Restoring with `qsys-script` generates default project (`.qpf`) and settings (`.qsf`) files that contain only basic information. The `.qsf` includes no project-specific assignments from the original project's `.qsf`. You can click **File ➤ Select Quartus Project** in Platform Designer to associate a system with any Quartus Prime project.

Related Information

Generate a Platform Designer System with qsys-script on page 451

1.20.2. Archiving Platform Designer Systems

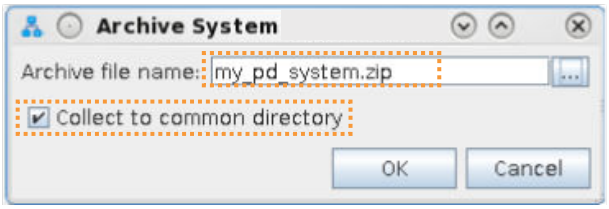
The Platform Designer GUI supports archiving an entire Platform Designer system to a single, compressed `.zip` file, and then restoring the archive in another instance of Platform Designer.

Alternatively, use the `qsys-archive` command to archive and restore Platform Designer systems at the command line.

To archive a Platform Designer system to a single, compressed `.zip` file:

1. Open a Platform Designer system.
2. In Platform Designer, click **File ➤ Archive System**. The **Archive System** dialog box appears.

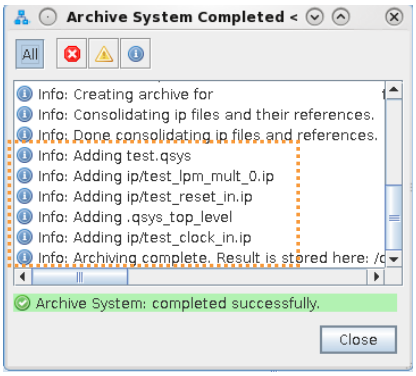
Figure 98. Archive System Dialog Box



3. Specify a name for **Archive file name**.

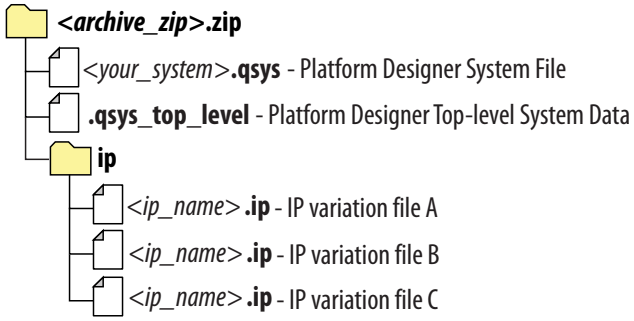
4. Enable or disable **Collect to common directory**. When enabled, Platform Designer collects the system's `.qsys` files in the root directory, and the `.ip` files in a single `ip` directory, while updating all references. Disable this option to maintain the current system's directory structure.
5. Click **OK**. Platform Designer generates the archive.

Figure 99. Archive System Complete



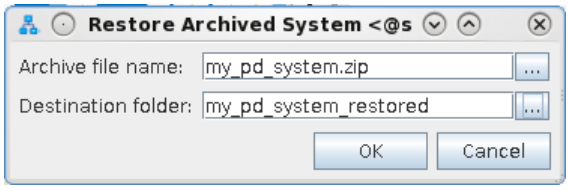
Note: Platform Designer automatically identifies the `.qsys` and `.ip` files needed for system archiving when using **Archive System**. You do not manually specify any files for archiving. **Archive System** does not include custom IP and related files in the archive by default.

Figure 100. System Archive Directory (Collect to common directory)



6. To restore the archived system, click **File ➤ Restore Archived System**. Select the **Archive file name**, and **Destination folder** to extract the restored files.

Figure 101. Restore Archived System



After restoration is complete, Platform Designer automatically launches the **Open System** dialog box, with the extracted project preloaded.

1.20.3. Including Platform Designer Systems in Project Archives

You can optionally save the elements of an Quartus Prime project—including any Platform Designer system or IP—in a single, compressed Quartus Prime Archive File (`.qar`). The `.qar` can preserve the project and settings files, source design files, IP and system files, programming output files, and all other files required to fully restore the project in the Quartus Prime software.

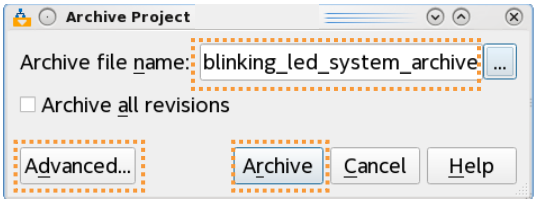
The archive `.qar` allows you to easily share projects between designers, or to transfer your project to a new version of the Quartus Prime software, or to Intel customer support. You can optionally include compilation reports, Platform Designer system files, and EDA tool integration files in the project archive.

You can fully customize the list of files that the archive includes, to ensure that your archive captures all custom IP and other files that you want to preserve. The **Archive Project** dialog box includes preset **File sets** that automatically include the appropriate detected files for **Source control**, **Service requests**, and other archiving scenarios.

By default, the **Archive Project** command includes detected Platform Designer system and IP files that are part of your project. Follow these steps to confirm that your project archive includes the Platform Designer system and IP files that you want:

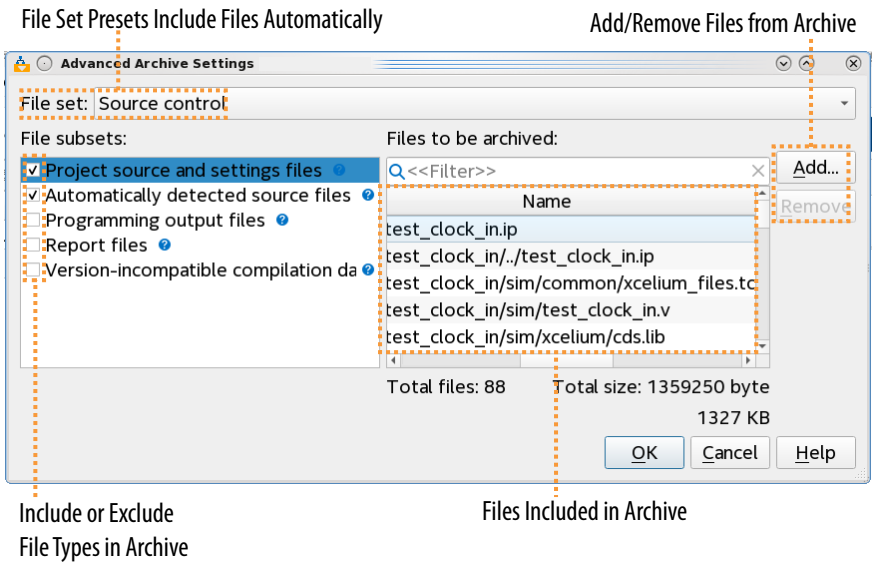
1. Open an Quartus Prime project (`.qpf`) that includes your Platform Designer system and IP.
2. Click **Project ➤ Archive Project** and specify an **Archive file name**.

Figure 102. Archive Project Dialog Box



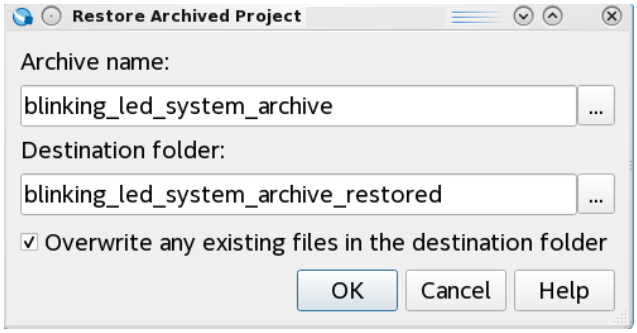
3. Click the **Advanced** button. The **Advanced Archive Settings** dialog box appears.
4. Select **Source control**, **Service requests**, or **Compilation Database and output** for the **File set**. All of these **File sets** include detected Platform Designer system and IP files in the archive automatically. Turn on or off the **File subsets** you want in the archive.
5. To customize the **Files to be archived** list, click the **Add** or **Remove** buttons to include or remove specific files.
6. To save the archive settings, click **OK**.

Figure 103. Advanced Archive Settings Dialog Box



7. To generate the project archive file, click the **Archive** button.
8. To restore the archived project (including the Platform Designer system and IP), click **Project ➤ Restore Archived Project**. Select the **Archive name**, and **Destination folder** to extract the restored files.

Figure 104.



1.20.3.1. Project Files to Include In External Revision Control

When archiving Quartus Prime projects for external source control, The **Source control** setting in **Advanced Archive Settings** dialog box is preset to include all appropriate file types for source control automatically.

Figure 105. Advanced Archive Settings Dialog Box

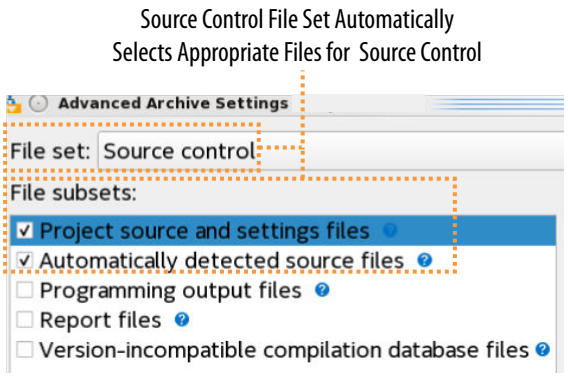


Table 28. Project Files to Include In External Revision Control

File Type	Description
Quartus Prime project setting and assignment files	• Quartus Prime Project Files (`.qpf`) • Quartus Prime Settings Files (`.qsf`) • Quartus Prime Pin Planner File (`.ppf`)
Timing constraint files	Synopsys Design Constraint Files (`.sdc`)
Design files	• Verilog HDL Design Files (`.v`) • SystemVerilog Design Files (`.sv`) • VHDL Design Files (`.vhd`) • Block Symbol Files (`.bsf`) • Verilog Quartus Mapping Files (`.vqm`) • Platform Designer System Files (`.qsys`) • State Machine Editor Files (`.smf`) • Tcl Script Design Files (`.tcl`)
System and IP files	• IP variation file (`.ip`) • Verilog IP design files (`.v`) • SystemVerilog IP design files (`.sv`) • VHDL IP design files (`.sv`) • VHDL Component Declaration Files (`.cmp`) • Quartus Prime IP file (`.qip`) • Quartus Prime Simulation IP File (`.sip`) • Platform Designer System Files (`.qsys`) • Platform Designer connection and parameterization files (`.sopcinfo`) • IP upgrade status files (`.csv`) • IP synthesis parameters files (`.qgsynthc`) • IP simulation parameters files (`.qgsimc`) • Platform Designer system exported as (`.tcl`).
EDA tool integration files	• Verilog HDL Output Files (`.vo`) • VHDL Output Files (`.vho`) • VHDL simulation model files (`.vhd`) • Verilog HDL simulation model files (`.v`) • Simulation library files (`cds.lib`, `hdl.var`) • Simulation setup scripts (`_setup.sh`, `.tcl`, `.spd`, `.txt`)

1.21. Sharing Platform Designer Packaged Subsystems

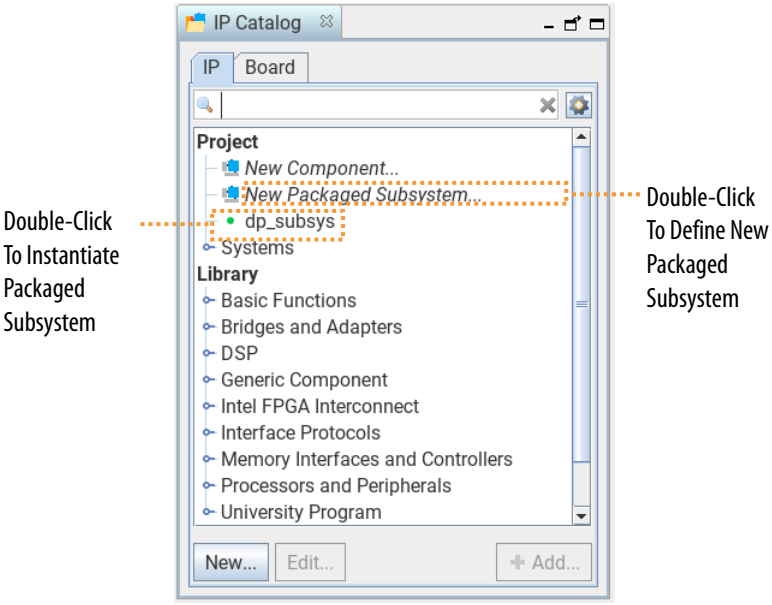
You can readily share or reuse a Platform Designer system by creating a packaged subsystem. The packaged subsystem combines everything about the pre-verified and parameterized system into a single, compressed, and portable `.qcp` file for reuse in other Platform Designer systems.

You or another packaged subsystem user can instantiate the packaged subsystem into another Platform Designer system to comprise the complete or partial system. The packaged subsystem author can disable viewing or modification by other users. This packaged subsystem flow simplifies the reuse and sharing of subsystems across multiple projects or designers.

The packaged subsystem author first defines all properties of the packaged subsystem in Platform Designer, including system components, connections, parameters, and supporting files. The packaged subsystem author next uses the **New Packaged Subsystem** command in Platform Designer's IP Catalog to save the current open system as a packaged subsystem. You can optionally include supporting RTL or other files in the packaged subsystem.

Once created, the packaged subsystem appears under **Project** in IP Catalog and the `.qcp` file appears in the location you specify. You can reuse and share the packaged subsystem by simply sharing the `.qcp` file.

Figure 106. Packaged Subsystem Available in IP Catalog for Instantiation



The following procedures describe how to create, instantiate, and revise a packaged subsystem. For detailed instructions using an example design, refer to *AN 1002: Sharing Platform Designer Packaged Subsystems*.

Related Information

AN 1002: Sharing Platform Designer Packaged Subsystems

1.21.1. User Personas for Packaged Subsystems

The packaged subsystem flow defines the following user personas and flows that this document describes in detail:

- Packaged subsystem author—the packaged subsystem author creates and saves the packaged subsystem `.qcp` file in Platform Designer. The packaged subsystem author defines the components, connections, parameters, and the external interfaces of the packaged subsystem. The packaged subsystem author can specify desired component parameters, and whether the packaged subsystem can be unlocked for modification, or just divable which allows the packaged subsystem user to dive in and view the internals of the packaged subsystem.
- Packaged subsystem user—the packaged subsystem user reuses the packaged subsystem by instantiating and connecting it to other components in another Platform Designer system. The packaged subsystem user may be able to modify the parameters or unlock the packaged subsystem for full editing control, depending on the packaged subsystem author specifications. The packaged subsystem author also determines whether the packaged subsystem is divable, which allows users to view the internals of the packaged subsystem.

1.21.2. Terminology for Packaged Subsystems

Table 29. Packaged Subsystems Terminology

Term	Description
Packaged Subsystem	A packaged subsystem combines everything about a Platform Designer system into a single, compressed `.qcp` file for sharing and reuse in other Platform Designer systems. The packaged subsystem author can optionally lock the packaged subsystem for viewing or modification by the packaged subsystem user.
Packaged Subsystem Author	The creator of a Platform Designer packaged subsystem that saves a system or subsystem as a packaged subsystem (`.qcp`) for sharing and reuse. The packaged subsystem author can set the `UNLOCKABLE` property to prevent or permit packaged subsystem editing for packaged subsystem users.
Packaged Subsystem Sharing	A packaged subsystem author can share a package subsystem with a packaged subsystem user by sharing the packaged subsystem `.qcp` file or by providing the IP search path for the user to specify when launching Platform Designer.
Packaged Subsystem User	Reuses a packaged subsystem by instantiating and connecting it to other components in another Platform Designer system. The packaged subsystem user may be able to modify the parameters or unlock the packaged subsystem for full editing control, depending on the packaged subsystem author specifications.
Package-level parameters	Parameters that appear in the parameter editor GUI when the packaged subsystem user instantiates or selects the packaged subsystem.
run_system_script	An important command for modifying the internal system inside a subsystem, `run_system_script` runs Platform Designer scripting commands such that all commands within curly braces run as separate scripts.
Subsystem	A standard unpackaged subsystem that you create in Platform Designer of the file type `.qsys`.
Subcomponent parameters	Parameters of the components of a packaged subsystem.

continued...

Term	Description
System	A standard top-level system that you create in Platform Designer of the file type `.qsys`.
Divable	The packaged subsystem author can use the `DIVABLE` property to specify whether the packaged subsystem user can dive in and view the internals of the packaged subsystem.
Unlockable	The packaged subsystem author can use the `UNLOCKABLE` property to specify whether the packaged subsystem user can unlock the packaged subsystem for full editing control.

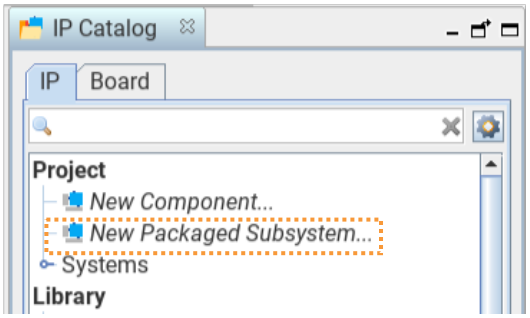
1.21.3. Creating a New Packaged Subsystem

In the packaged subsystem author flow, the packaged subsystem author creates a regular Platform Designer system, and then bundles the system into a packaged subsystem represented as a `.qcp` file. You can then provide the `.qcp` file to one or more packaged subsystem users.

Follow these steps to save an open Platform Designer system as a packaged subsystem `.qcp` file for reuse or sharing in other Platform Designer systems.

1. Define all properties of the system in the Platform Designer **System View** tab, including the system components, connections, parameters, and supporting files that you want to include in the packaged subsystem.
2. In Platform Designer's IP Catalog, click **New Packaged Subsystem**. The **New Packaged Subsystem** dialog box appears for specification of the packaged subsystem properties.

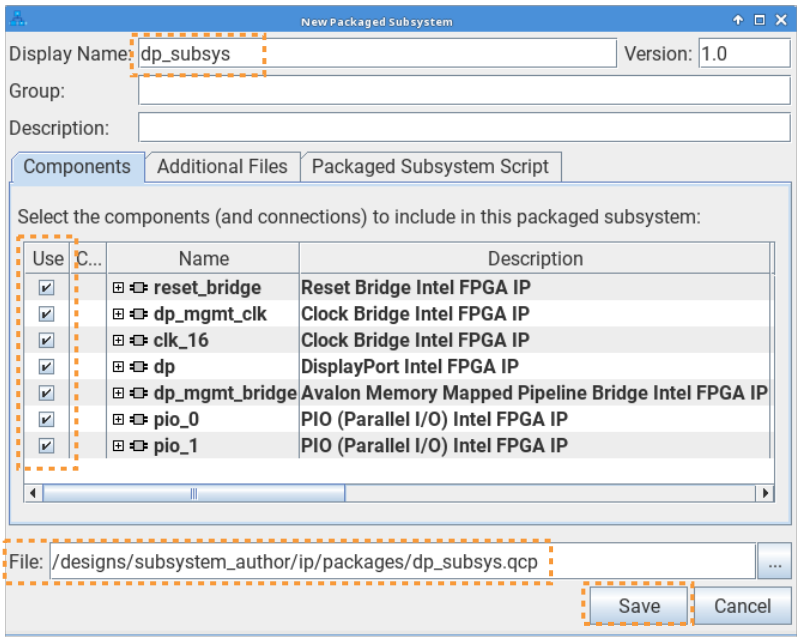
Figure 107. New Packaged Subsystem Command in IP Catalog



3. In the **Components** tab, specify a name for the packaged subsystem in **Display Name**. Refer to the image in New Packaged Subsystem Dialog Box
4. Optionally specify the **Group**, **Version** number, and **Description** to help you identify the packaged subsystem.
5. Under **Select the components (and connections) to include in this package** disable (uncheck) any connection, component, or node that you want to omit from the packaged subsystem in the **Use** column.

6. In the **File** box, specify the location to save the packaged subsystem file. The default path is `<project_directory>/ip/packages/<subsystem_name>.qcp`.
7. Optionally specify subsystem files and package script settings, as Specifying Additional Packaged Subsystem Files on page 121 and Modifying the Packaged Subsystem Script on page 122 describe.
8. When the packaged subsystem is complete, click **Save** in the **New Package** dialog box. The `.qcp` file saves to the location that you specify, and the packaged subsystem appears under **Project** in IP Catalog.

Figure 108. New Packaged Subsystem Dialog Box



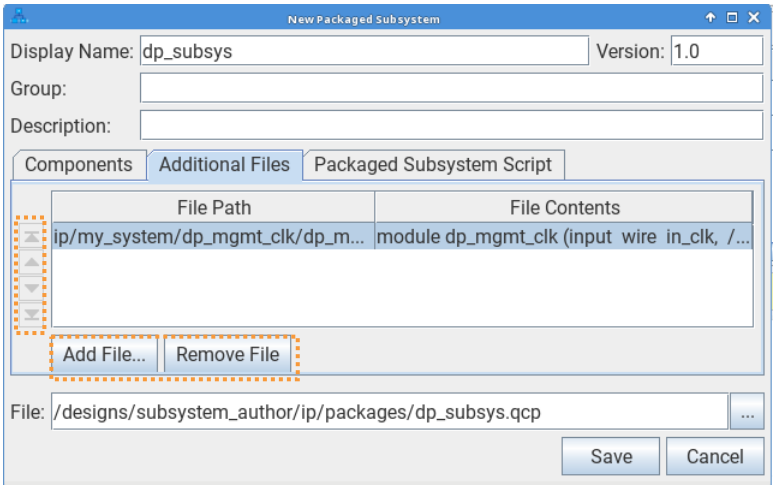
Related Information

AN 1002: Sharing Platform Designer Packaged Subsystems

1.21.4. Specifying Additional Packaged Subsystem Files

Follow these steps if you want to specify additional files to include with a Platform Designer packaged subsystem. You can include supporting files for your packaged subsystem, such as RTL for custom IP components.

Figure 109. Specifying Additional Files for a Packaged Subsystem



1. In the **Components** tab of the **New Packaged Subsystem** dialog box, define the basic properties of the new packaged subsystem, as Creating a New Platform Designer Packaged Subsystem describes.
2. Click the **Additional Files** tab.
3. To add a new file to the packaged subsystem, click the **Add File** button and select the supporting files you want to add.
4. To remove any additional files from the packaged subsystem, select the file and click the **Remove File** button.
5. To reorder the list of additional files, click the arrow buttons to move the files up or down in the list.
6. Optionally modify packaged subsystem script settings, as Modifying the Packaged Subsystem Script describes.
7. When the packaged subsystem is complete, click **Save** in the **New Package** dialog box. The `.qcp` file saves to the location that you specify, and the packaged subsystem appears under **Project** in IP Catalog.

Related Information

AN 1002: Sharing Platform Designer Packaged Subsystems

1.21.5. Modifying the Packaged Subsystem Script

The packaged subsystem infrastructure uses a Tcl script file to specify the packaged subsystem properties. You can modify the packaged subsystem script on the **Packaged Subsystem Script** tab to control the following in the packaged subsystem:

- Add parameters to enable or disable individual IP components in the packaged subsystem.
- Lock or unlock packaged subsystem editing for packaged subsystem users.
- Add packaged subsystem-level parameters that set parameter values of internal components.
- Modify the internal `.qsys` system file.

For example, when you enable or disable components on the **Components** tab, Platform Designer modifies the default script lines correspondingly in the packaged subsystem script when you right-click in the **Packaged Subsystem Script** tab and click **Reset with Default Script**. The packaged subsystem author can optionally modify the packaged subsystem script directly on the **Packaged Subsystem Script** tab.

For example, the following code creates two checkbox controls that are visible in the parameter editor GUI of the packaged subsystem for packaged subsystem users. These controls are linked to the RX and TX packaged subsystem level parameters. The RX and TX parameters are of type boolean and are asserted false by default.

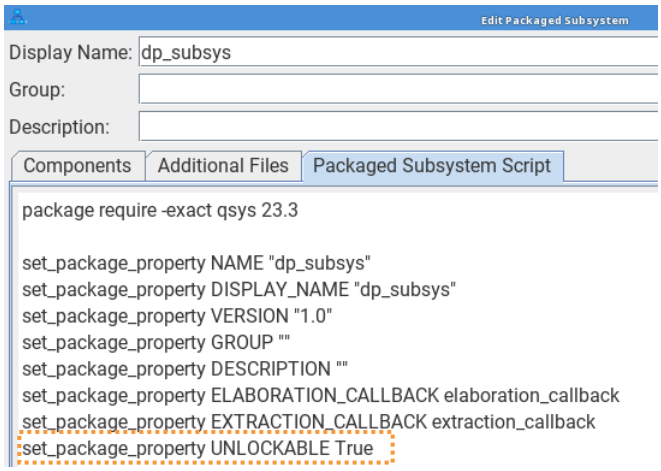
```
add_parameter TX BOOLEAN false "Enable transmitter"
add_parameter RX BOOLEAN false "Enable receiver"
```

Note: For details on these commands, refer to Parameters.

To modify the packaged subsystem script, follow these steps:

1. In the **Components** tab of the **New Packaged Subsystem** dialog box, define the basic properties of a the packaged subsystem, as Creating a New Platform Designer Packaged Subsystem describes.
2. Click the **Packaged Subsystem Script** tab.
3. Modify any of the script in the **Packaged Subsystem Script** tab to match your deployment preferences. For example, the following modification allows the packaged subsystem user to unlock the packaged subsystem for full control.

Figure 110. Packaged Subsystem Script Change (Subsystem Can Be Unlocked)



4. When the packaged subsystem is complete, click **Save** in the **New Package** dialog box. The `.qcp` file saves to the location that you specify, and the packaged subsystem appears under **Project** in IP Catalog.

Related Information

AN 1002: Sharing Platform Designer Packaged Subsystems

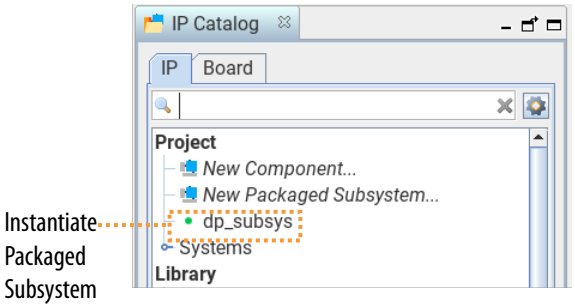
1.21.6. Instantiating a Packaged Subsystem

As a packaged subsystem user, you can instantiate a packaged subsystem in another Platform Designer system. For example, you can share a packaged subsystem `.qcp` file with another designer who can then instantiate and connect the packaged subsystem in their own Platform Designer system. The packaged subsystem author determines which elements of the packaged subsystem are editable in the packaged subsystem.

To instantiate a packaged subsystem within a Platform Designer system, follow these steps:

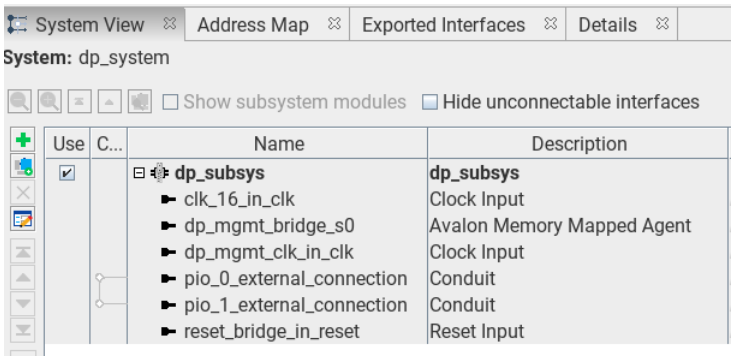
1. The packaged subsystem author creates the packaged subsystem, as Creating a New Platform Designer Packaged Subsystem describes.
2. The packaged subsystem user opens or creates the Platform Designer system that will instantiate the packaged subsystem, as Creating or Opening a Platform Designer System describes.
3. The packaged subsystem user places the subsystem `.qcp` file in the Quartus Prime software's IP search path. The packaged subsystem now appears under **Projects** in Platform Designer's IP Catalog.

Figure 111. Packaged Subsystem Available in IP Catalog for Instantiation



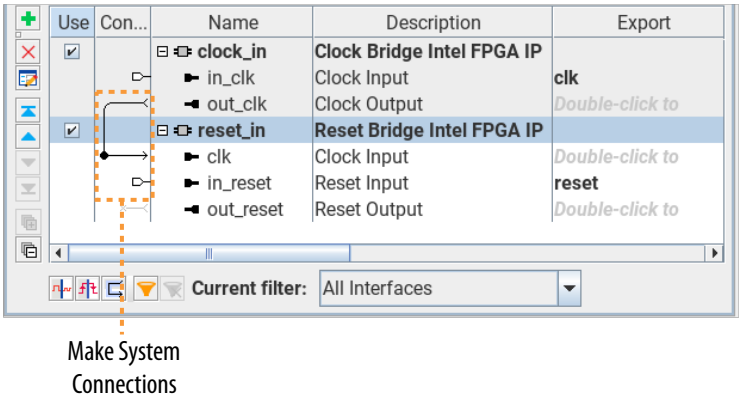
4. To instantiate the packaged subsystem in the open Platform Designer system, the packaged subsystem user double-clicks the packaged subsystem name in IP Catalog. The packaged subsystem now appears as a single IP component in the **System View** tab, according to the deployment options that the packaged subsystem author specifies.

Figure 112. Packaged Subsystem Instantiated in Platform Designer System



5. Connect the packaged subsystem to other system components, including clock and reset signals.

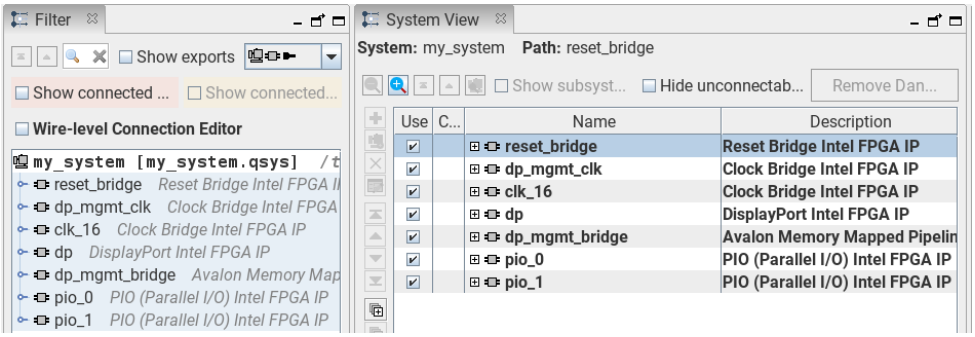
Figure 113. Connecting System Components



6. Depending on options that the packaged subsystem author specifies, you may be able to perform one or more of the following actions on the packaged subsystem:

- **Right-Click ➤ Dive Into Package**—when the `DIVABLE` property is set to `True` (the default setting) in the packaged subsystem script, you can dive into the packaged subsystem to view the details of the packaged subsystem components, connections, and parameters in another instance of Platform Designer.

Figure 114. Dive Into Package Shows Contents of the Packaged Subsystem



- **Right-Click ➤ Unlock**—when the `UNLOCKABLE` property is set to `True` in the packaged subsystem script you can unlock the packaged subsystem and interact with it like any other Platform Designer system.

  Note: This action cannot be undone.

Related Information

AN 1002: Sharing Platform Designer Packaged Subsystems

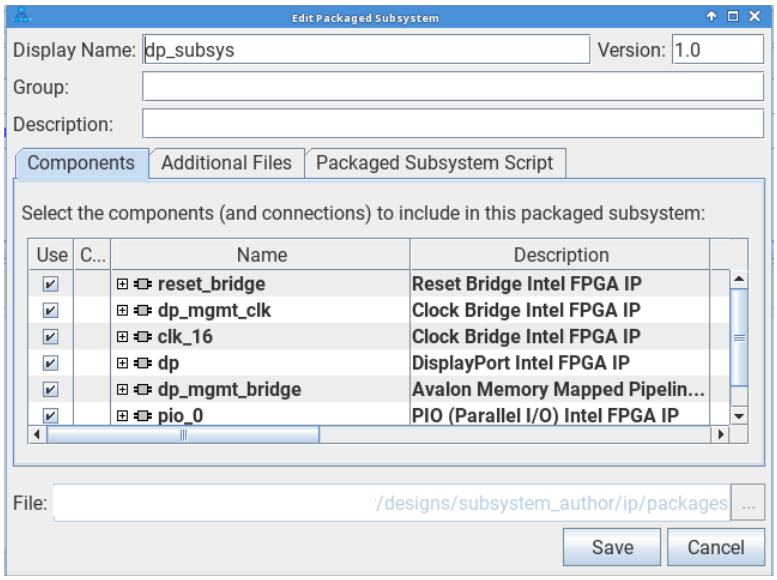
1.21.7. Revising a Packaged Subsystem

As the packaged subsystem author, you can revise a packaged subsystem that you have previously created. You can open the existing packaged subsystem in Platform Designer, allow unlock of the packaged subsystem for other users, and specify other system changes in the **Packaged Subsystem Script** tab. You can then save the updated packaged subsystem (`.qcp`) with the same name.

To revise a packaged subsystem that you previously created in Platform Designer, follow these steps:

1. Open the Platform Designer system that contains the packaged subsystem, as Creating or Opening a Platform Designer System describes.
2. In Platform Designer's IP Catalog, right-click the packaged subsystem under **Project** and click **Edit**. The **Edit Packaged Subsystem** dialog box appears.

Figure 115. Edit Packaged Subsystem Dialog Box



3. Make your desired revisions in the **Components**, **Additional Files**, and **Packaged Subsystem Scripts** tabs, as Creating a New Packaged Subsystem on page 120 describes.
4. When the packaged subsystem is complete, click **Save** in the **Edit Packaged Subsystem** dialog box. The `.qcp` file saves to the location that you specify, and the packaged subsystem appears under **Project** in IP Catalog.

Related Information

AN 1002: Sharing Platform Designer Packaged Subsystems

1.21.8. New Packaged Subsystem Dialog Box Options and Controls

The following settings and controls are available in the **New Packaged Subsystem** dialog box. These settings and options are also available in the **Edit Packaged Subsystem** dialog box unless otherwise noted.

Table 30. Component Tab Options and Controls

Option Name	Description
Display Name	Specifies a name for the new packaged subsystem you are creating. When the packaged subsystem is complete, this named packaged subsystem appears as a single IP in the Platform Designer IP Catalog. This option requires a value.
Version	Specifies the version number of the packaged subsystem. The default value is `1.0`. This setting is optional.
Group	Specifies the IP group of the packaged subsystem. This setting is optional.
Description	Specifies descriptive details about the packaged subsystem to help you identify it later.
Components	Species the components in the packaged subsystem, their connections, and whether they should be visible in the packaged subsystem for other users.
Use	Specifies whether the component should appear in the packaged subsystem or not. Turn on the checkbox to include the component in the packaged subsystem. Turn off the checkbox to exclude the component from the packaged subsystem. By default, **Use** is on for all components in the current system.
File	Specifies the path for saving the packaged subsystem `.qcp` file. The default path is `<project_directory>/ip/packages/<subsystem_name>.qcp`. Using the default path ensures that IP Catalog finds the packaged subsystem without modifying the IP search path. This option is read only in **Edit Packaged Subsystem** dialog box.
Save	Saves the `.qcp` file to the default location or a location that you specify, and the packaged subsystem appears under **Project** in IP Catalog.

Table 31. Additional Files Tab Options and Controls

Option Name	Description
File Path	Lists the path and filename of all files that you add to the packaged subsystem with **Add File**.
File Contents	Provides a description of the file types that you add to the packaged subsystem with **Add File**.
Add File	Allows you to browse for and add supplementary files to include in the packaged subsystem.
Remove File	Removes one or more files from the files list so the file is not added to the packaged subsystem.
Move to the top	Moves the selected file to the top of the files list.
Move up	Moves the selected file up one level in the files list.
Move down	Moves the selected file down one level in the files list.
Move to the bottom	Moves the selected file down to the bottom of the files list.

Table 32. Package Script Tab Options

Option	Description
Package script	Displays the editable packaged subsystem script that defines the packaged subsystem properties, parameters, and callbacks. You can directly edit this script to match your preferences.
Right-Click ➤ Reset with default script	Resets the current packaged subsystem script with the default packaged subsystem script for all subsystems.
Right-Click ➤ Preview the GUI	Displays a preview of the parameterization GUI that appears when users double-click the packaged subsystem in IP catalog.

1.22. Comparing Platform Designer Systems and IP components

Platform Designer includes an integrated System Diff Tool that highlights the differences between two Platform Designer systems or IP components (`.qsys` or `.ip` files). You can use the System Diff Tool to quickly identify the differences between two versions of systems or IP components.

The Platform Designer System Diff Tool supports the following use cases:

- Compare two versions of the same Platform Designer system (`.qsys`) on disk. View the modifications to the system.
- Compare the currently open Platform Designer system (`.qsys`) with a system on disk. View the differences between the systems.
- Compare a Platform Designer system (`.qsys`) file before and after a version upgrade. View the updated settings.
- Compare a Platform Designer system (`.qsys`) file with different device settings. View the setting differences.
- Compare a `.ip` component file before and after parameter changes. View the changes to any component interfaces.

The System Diff Tool uses color coding to highlight differences for any of following items added or removed in the system:

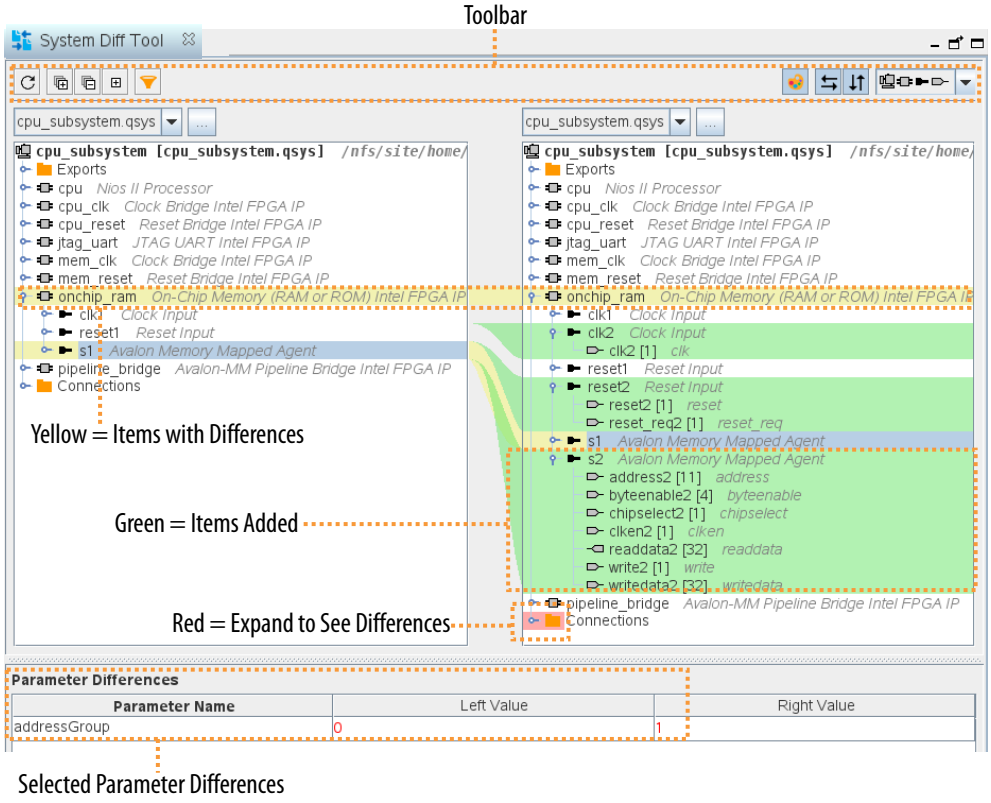
- Exports
- Modules
- Interfaces
- Ports
- Connections
- Parameters
- Assignments
- Properties

1.22.1. Using the System Diff Tool

You can open the System Diff Tool GUI from Platform Designer (**View ➤ System Diff Tool**) to compare Platform Designer systems (`.qsys`) or IP component files (`.ip`).

The System Diff Tool GUI displays the file comparison hierarchically, using color coding to display the differences in systems, modules, interfaces, ports, exports, and connections side-by-side in the top two panes. The bottom pane also displays the selected parameter differences, or you can right-click to **Show All Values** for the selected item.

Figure 116. Platform Designer System Diff Tool

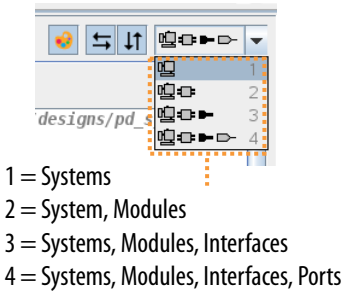


To run the System Diff Tool from Platform Designer:

1. Click **View ➤ System Diff Tool**.
2. In the Current System list at the top of each pane, specify the items for comparison. The System Diff Tool GUI then highlights the differences in each file. Selections and scrolling synchronize between the two panes:
   - Removed items appear in purple highlight.
   - Added items appear in green highlight.
   - Items with differences appear in yellow highlight.(5)
   - Items with collapsed children with differences are highlighted in red.
3. Optionally perform any of the following to refine the comparison view:

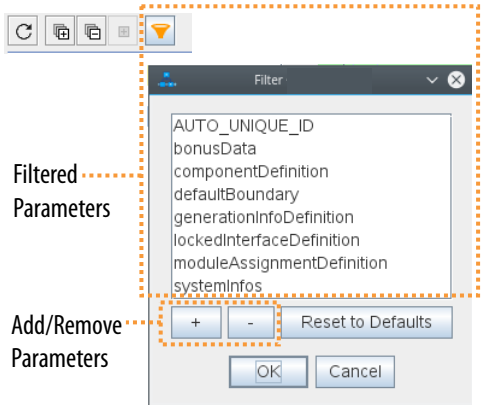
- To filter the comparison view by system, module, instance, or port hierarchy, select the filter icon that corresponds with the level of hierarchy that you want to display.

Figure 117. Filter the Comparison by Hierarchy



- To filter the comparison view by parameter name, click the Filter button to hide or show various parameters in the comparison. By default, this filter is set to exclude internal parameters that do not affect system behavior.

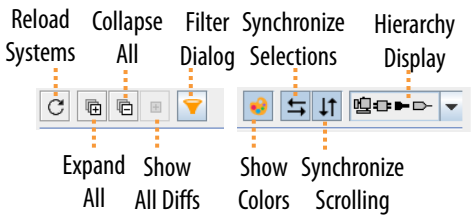
Figure 118. Filter the Comparison by Parameter



- To reload the systems for comparison after making changes to a system, click the **Reload Systems** button on the toolbar.
- To expand or collapse the comparison hierarchy, click the **Expand All** or **Collapse All** button.
- To expand the hierarchy to show all differences, click the **Show All Diffs** button.
- To show the comparison color coding, click the **Show Colors** button.
- To synchronize the scrolling or selections between the comparison panes, click the **Synchronize Scrolling** or **Synchronize Selections** button. You can use **Synchronize Selections** to compare the parameters of any two selected modules. (normally, items are diffed if they have the same name.)

(5) Item comparison is strictly based on naming. If an item on the left and right have the same name, they are compared. Otherwise, the item is marked as added or removed.

Figure 119. System Diff Toolbar



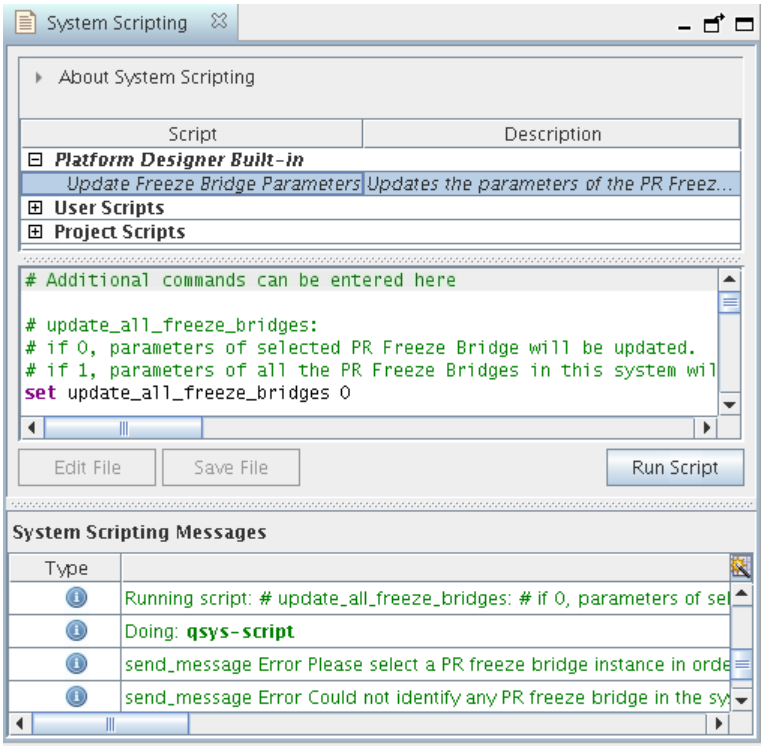
1.23. Running System Scripts

The **System Scripting** tab allows you to enter, save, and execute Tcl scripts on your Platform Designer system. The tab includes a selection of provided scripts, as well as support for storage and retrieval of your own scripts.

Follow these steps to enter, save, and execute Tcl scripts on your Platform Designer system:

1. To open the **System Scripting** tab, click **View ➤ System Scripting**.

Figure 120. System Scripting Tab



2. For **User Scripts** or **Project Scripts**, click **<<add script>>** to add a new script file to this entry. You can drag items between the **Project Scripts** and **User Scripts** fields.
3. To add additional commands to run before the script, right-click the column header and enable **Additional Commands**. Selecting this option displays a third column, in addition to **File** and **Description**. Double-click the entry in this field to add

commands to execute before running your script. Alternatively, you can add the additional commands to your script, directly through the display pane in the middle, in the specified section.

The **System Scripting** tab provides the following fields:

Table 33. System Scripting Tab Options

Name	Description
Platform Designer Built-in Scripts	Lists non-editable scripts that Platform Designer provides.
User Scripts	You can add your own scripts to this entry. Platform Designer saves these scripts to your user preference file, available in your home directory. The scripts that you add to this entry are available every time you open Platform Designer.
Project Scripts	You can add your own scripts to this entry. Platform Designer saves these scripts to your current system. The scripts that you add to this entry are available only when you open this specific Platform Designer system.
Edit File	Selecting the script in the **File** field displays the script in the pane below. Click **Edit File** to edit the script.
Revert File	Discards all your changes to the edited file.
Save File	Saves your changes to the edited file.
Run Script	Executes the selected script.
System Scripting Messages	Displays the warning and error messages when running the script.

Related Information

Platform Designer Command-Line Utilities on page 441

1.24. Creating a System with Platform Designer Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout. • Updated *Changing the Target Board for a System* topic to describe alternative methods. • Updated *Create New Board Dialog Box Options* topic to describe additional preset options. • Updated *Generating Board and Preset Files for Existing Systems* section to describe alternative methods. • Updated *Customizing IP Presets* topic to describe board settings. • Updated *Creating IP Presets* topic to describe board settings. • Added new *Applying Presets After Migrating a Board* section. • Added new *Using the Qrun Flow in Platform Designer* section. • Updated *Specifying IP Component Instantiation Options* topic to remove HLS options. • Updated *Saving Platform Designer Systems* topic to describe generated Tcl scripts also include any preset pin constraints. • Added new *Add IP RTL Core Generated from the Intel oneAPI Base Toolkit* topic.
2023.12.20	23.4	• Updated *Synchronizing System Component Information* topic to explain synchronizing behavior for "Different Items" versus "Missing Items."
2023.10.02	23.3	• Updated *What's New In This Version* topic for packaged subsystems. • Revised *Viewing a Platform Designer System* for the latest GUI layout description. • Added new *Sharing Platform Designer Packaged Subsystems* section to describe new ability to package and share a subsystem.
2023.06.26	23.2	• Updated *Generating Simulation Files for NoC Designs* topic for latest NoC design flows. • Updated *Generating a Simulation Registration Include File for RTL Simulation of NoC Designs* topic for latest NoC design flows.
2023.04.03	23.1	• Added *Generating Presets for Existing Systems with Multiple Instances* topic. • Added more examples to *Generating Board and Presets for Existing Systems Using Command Line* topic. • Added *Preset Files Saved* topic. • Added *Generating Simulation Files for NoC Designs* topic. • Added *Generating a Simulation Registration Include File in the Platform Designer Flow* topic. • Added *Contents of Simulation Registration Include File* topic. • Added *Connecting NoC IP* topic. • The product family name is updated to "Intel Agilex 7" to reflect the different family members.
2022.12.12	22.4	• Updated *What's New in This Version* topic for board-aware flow support. • Revised *Creating or Opening a Platform Designer System* topic to reference board-aware GUI. • Updated *Specifying the Target Intel FPGA Device or Board for a System* topic for board-aware GUI. • Added new *Using the Board-Aware Flow in Platform Designer* section. • Added new *Creating IP Presets Targeting Specific Boards* section. • Removed limitation statement about no support for assertion monitors in Questa Intel FPGA Edition simulator. This simulator does support assertion monitors.

continued...

Document Version	Quartus Prime Version	Changes
2022.09.26	22.3	• Added clarifying note about the default agent selection to *Specifying a Default Avalon Agent or AXI Subordinate* topic. • Updated screenshot in *Specifying Interconnect Parameters* topic for new **Response FIFO Type** option. • Revised *Interconnect Parameters* topic for new **Response FIFO Type** option.
2022.06.20	22.2	• Updated *What's New In This Version* topic for latest changes in Platform Designer.
2022.04.02	22.1	• Added Top FAQs navigation and *What's New In This Version* topic. • Added new *Correcting Platform Designer System Timing Issues* topic. • Updated entire chapter for new AXI "manager" and AXI "subordinate" replacement terms. Refer to the AMBA® AXI and ACE Protocol Specification.
2021.10.04	21.3	• Added new *Preserving a System Module, Interface, or Port for Debugging* topic. • Added new *Changing the Platform Designer Font* topic. • Added new *Comparing Platform Designer Systems and IP Components* section. • Updated *Generation Dialog Box Options* topic note for Questa Intel FPGA Edition simulator. • Added references to the Nios V processor and documentation throughout.
2021.03.29	21.1	• Converted to "host" and "agent" inclusive terminology for Avalon memory mapped interface descriptions and related GUI elements throughout. • Added new *Saving and Archiving Platform Designer Systems* overview. • Revised *Saving Platform Designer Systems* for more detail, screenshot, and links about Export System as Platform Designer script (.tcl). • Revised *Archiving Platform Designer Systems* for more detail, screenshots, and links about using qsys-archive command. • Added new *Including Platform Designer Systems in Project Archives* topic. • Added new *Project Files to Include in External Revision Control* topic. • Updated *Generation Dialog Box Options* topic for new simulation options. • Updated *Simulating Platform Designer Systems* for new simulation options and images.
2020.12.14	20.4	• Updated "Specifying Interconnect Parameters" topic for latest GUI options and methods. • Updated "Interconnect Parameters" table for latest parameters and names. • Updated all System View tab screenshots for latest filter options. • Referenced Linux limitation for HLS generic component types. • Revised "Files generated for IP cores and Platform Designer Systems" diagram variable names.
2020.09.28	20.3	• Removed reference to obsolete Read/Write Waveforms option from "Modifying IP Parameters" topic. • Removed reference to obsolete System Information tab and Implementation Templates tab from "Specifying IP Component Instantiation Options" topic. • Removed reference to obsolete Direction option from "Changing a Conduit to a Reset" topic. • Added details about filter controls to "Filtering the System View" topic.
2020.01.31	19.1	• Removed obsolete "Implementing Performance Monitoring" topic.

continued...

Document Version	Quartus Prime Version	Changes
2019.10.02	19.1	• Updated location of interconnect parameters security setting in "Configuring Platform Designer System Security" topic.
2019.09.30	19.1	• Removed reference to obsolete Bus Analyzer Toolkit from "Implementing Performance Monitoring" topic.
2019.06.24	19.1	• Removed obsolete Interconnect Type parameter from "Interconnect Parameters" topic.
2019.04.30	19.1	• Corrected typographical error in "Interconnect Parameters" topic.
2019.04.01	19.1	• Described new Domains tab for specifying system-wide or domain-specific interconnect parameters. • Described new default use of synchronous reset option for Stratix® 10 designs in "Interconnect Parameters." • Described new Schematic tab in "Previewing the System Interconnect."
2018.12.15	18.1	• Replaced references to System Contents tab with new System View tab. • Described new Filter tab in Filtering the "Filtering the System View." • Updated "Disabling or Enabling Parallel IP Generation" to indicate option is now on by default and describe optional settings. • Moved command-line utility information into new "Platform Designer Command-Line Interface" chapter. • Removed "Creating a Combined Simulation Script" topic that does not apply to Platform Designer. • Revised headings and re-organized content into user task-based sections. • Updated screenshots for latest version.
2018.09.24	18.1	• Removed duplicated topic: *Manually Control Pipelining in the Platform Design Interconnect*. The topic is now in the *Platform Design Interconnect* chapter. • Added statement about supported standards for IP-XACT. • Divided topic: *Specify Implementation Type for IP Components* into *Configure the System Representation of an IP Core* and *Implementation Type*. • Reorganized information about associating Intel Quartus Prime projects to Platform Designer systems. • Grouped information regarding definition and management of IP cores in Platform Designer under topic: *IP Cores in Platform Designer*, and updated contents. • Expanded description of parallel IP generation. • In topic *64-Bit Addressing Support*, added link to information about the auto base assignment feature.
2018.06.15	18.0	• Updated description of *Enable ECC protection* in table: *System-Wide Interconnect Requirements*. • Updated example in topic: *Generate a Platform Design System with qsys-script*.
2018.05.07	18.0	• Added support for hierarchical simscripts, and the Xcelium Parallel Simulator in . • Added support for `--debug` command used with `qsys-edit`. • Added support for wire-level expressions and connectivity. • Added `_hw.tcl` commands to support wire-level expressions.
2017.11.06	17.1	• Changed instances of *Qsys Pro* to Platform Designer
2017.05.06	17.0	• Updated the topic - *Create/Open Project in Qsys Pro* • Updated the topic - *Modify the Target Device* • Updated the topic - *Modify the IP Search Path*

continued...

Document Version	Quartus Prime Version	Changes
		• Added new topic - *Save your System* • Added new topic - *Archive your System* • Added new topic - *Synchronize IP File References* • Updated the topic - *Upgrade Outdated IP Components in Qsys Pro*. • Added new topic - *Run System Scripts* • Added new topic - *View Avalon Memory Mapped Domains in Your Qsys Pro System* • Updated the topic - *Qsys Pro Scripting Command Reference* for new Tcl scripting commands • Updated the topic - *Qsys Pro Scripting Property Reference* for new Tcl scripting property
2016.10.31	16.1	• Implemented Intel rebranding. • Implemented Qsys rebranding. • Integrated Qsys Pro chapter with Qsys. • Added command-line options for qsys-archive. • Added command-line options for quartus_ipgenerate. • Updated the Qsys Pro scripting commands. • Added topic on Qsys Pro design conversion.
2016.05.03	16.0	• Qsys Command-Line Utilities updated with latest supported command-line options. • Added: *Generate Header Files*
2015.11.02	15.1	• Added: *Troubleshooting IP or Qsys Pro System Upgrade*. • Added: *Generating Version-Agnostic IP and Qsys Pro Simulation Scripts*. • Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0	• New figure: *Avalon Memory Mapped Write Host Timing Waveforms in the Parameters Tab*. • Added **Enable ECC protection** option, *Specify Qsys Interconnect Requirements*. • Added External Memory Interface Debug Toolkit note, *Generate a Qsys System*. • Modelsim-Altera now supports native mixed-language (VHDL/Verilog/SystemVerilog) simulation, *Generating Files for Synthesis and Simulation*.
December 2014	14.1	• Create and Manage Hierarchical Qsys Systems. • Schematic tab. • View and Filter Clock and Reset Domains. • **File ➤ Recent Projects** menu item. • Updated example: Hierarchical System Using Instance Parameters
August 2014	14.0a10	• Added distinction between legacy and standard device generation. • Updated: *Upgrading Outdated IP Components*. • Updated: *Generating a Qsys System*. • Updated: *Integrating a Qsys System with the Quartus II Software*. • Added screen shot: *Displaying Your Qsys System*.
June 2014	14.0	• Added tab descriptions: Details, Connections. • Added *Managing IP Settings in the Quartus II Software*. • Added *Upgrading Outdated IP Components*. • Added *Support for Avalon Memory Mapped Non-Power of Two Data Widths*.

continued...

Document Version	Quartus Prime Version	Changes
November 2013	13.1	• Added *Integrating with the .qsys File*. • Added *Using the Hierarchy Tab*. • Added *Managing Interconnect Requirements*. • Added *Viewing Qsys Interconnect*.
May 2013	13.0	• Added AMBA APB support. • Added qsys-generate utility. • Added VHDL BFM ID support. • Added *Creating Secure Systems (TrustZones)* . • Added *CMSIS Support for Qsys Systems With An HPS Component*. • Added VHDL language support options.
November 2012	12.1	• Added AMBA AXI4 support.
June 2012	12.0	• Added AMBA AX3I support. • Added Preset Editor updates. • Added command-line utilities, and scripts.
November 2011	11.1	• Added Synopsys VCS and VCS MX Simulation Shell Script. • Added Cadence Incisive Enterprise (NCSIM) Simulation Shell Script. • Added *Using Instance Parameters and Example Hierarchical System Using Parameters*.
May 2011	11.0	• Added simulation support in Verilog HDL and VHDL. • Added testbench generation support. • Updated simulation and file generation sections.
December 2010	10.1	Initial release.

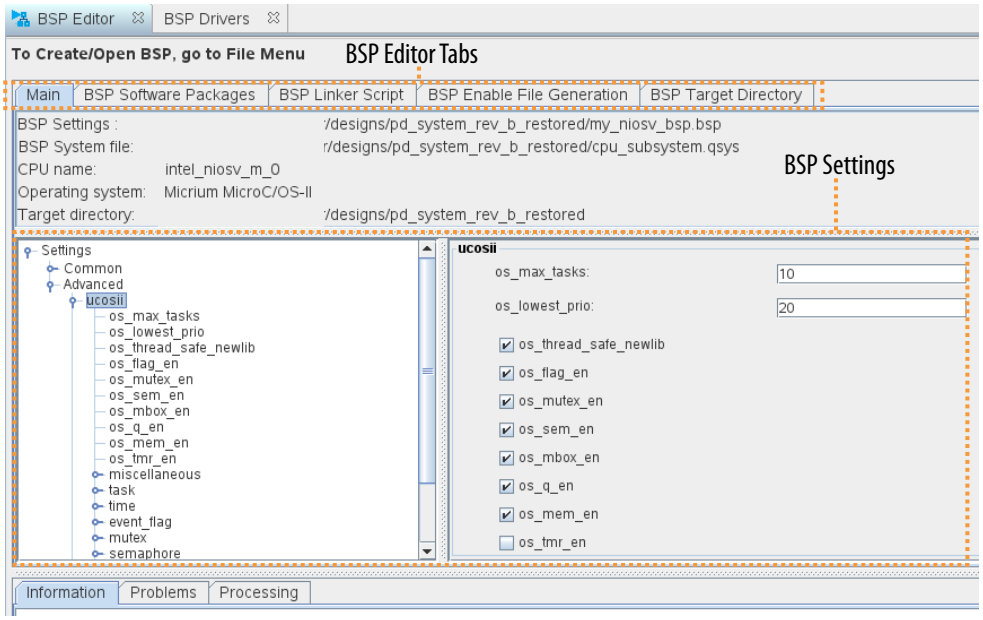
2. Creating a Board Support Package with BSP Editor

Platform Designer includes the BSP Editor board support package editing tool. A board support package (BSP) provides a software runtime environment for embedded systems, such as Nios II processor and Nios V processor systems.

The BSP simplifies your interface with connected hardware by providing the Intel FPGA hardware abstraction layer (HAL), an optional RTOS, and device drivers. BSP Editor is also available from the Nios II Embedded Design Suite (EDS).

The BSP Editor is a GUI tool that you can launch from Platform Designer to configure BSP contents. The BSP Editor allows you to specify the Harvard architecture CPU for your BSP, based on your Platform Designer system's `.qsys` file, or based on a `.sopcinfo` file that stores the target hardware definition. The `.sopcinfo` file generates during Platform Designer system HDL generation.

Figure 121. BSP Editor



The BSP isolates your application from system-specific details such as the memory map, available devices, and processor configuration. You decide the placement of the application's program code and data, depending on the memory devices that you connect to the CPU's instruction bus.

For each peripheral IP that you connect to the selected CPU's data bus, the BSP Editor automatically includes the matching driver for each peripheral IP as part of the BSP. The BSP Editor packages each driver using the `sw.tcl` API. The `sw.tcl` API defines

the collection of driver source files for the matching IP component type accessible by the CPU's data bus. Each driver may define software settings for configuring driver implementation or behavior.

You can use BSP Editor in the following modes. The BSP **System file** that you specify determines the mode.

Table 34. BSP Editor Modes in Platform Designer

	BSP Editor Integrated Mode	BSP Editor Stand-Alone Mode
Description	Generate a BSP directly for a Platform Designer system (`.qsys`). Any changes that you make in the **System View** automatically update in the BSP. Nios V processor designs require Integrated Mode.	BSP Editor is not fully integrated with Platform Designer, requiring manual update of the BSP for any changes that you make to the system using the `.sopcinfo` file. Nios II processor designs require Stand-Alone Mode.
Processor Support	Nios V Processor[6]	Nios II Processor
To Launch (BSP System File)	Specifying a `.qsys` file as **System file** launches BSP Editor in integrated mode.	Specifying a `.sopcinfo` as **System file** launches BSP Editor in stand-alone mode.

Note: For complete information about how to develop a BSP for a Nios II processor system or Nios V processor system, refer to the *Nios II Software Developer Handbook* or the *Nios V Processor Quick Start Guide*, respectively.

Related Information

- Nios II Software Developer Handbook
- Nios V Embedded Processor Design Handbook
- Nios V Processor Reference Guide

2.1. Creating a BSP from Platform Designer

You can launch the BSP Editor from Platform Designer to define a new BSP for a Platform Designer system or other hardware design. You can launch the BSP Editor in either integrated or stand-alone mode, depending on the **System file** that you specify for the BSP in Platform Designer.

When you create a BSP from Platform Designer, the BSP Editor stores your BSP definition in a BSP Settings File (`.bsp`) for later retrieval. You can reopen an existing BSP by opening the `.bsp` file in Platform Designer. The BSP generally includes a `.a` file, header files (for example, `system.h`), and a linker script (`linker.x`). You use these BSP files when creating an application.

Launching BSP Editor in Integrated Mode

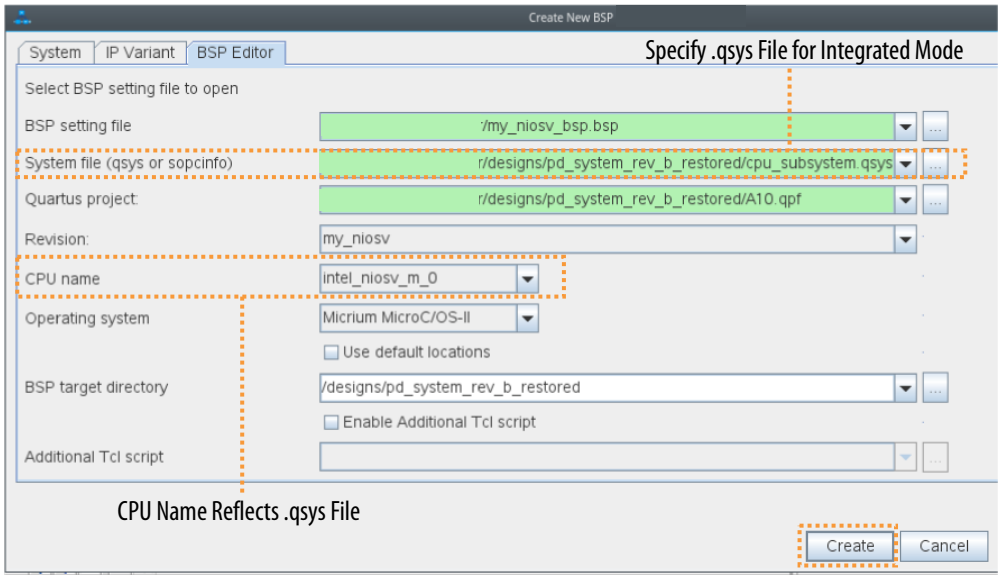
In integrated mode, BSP Editor automatically updates for any changes that you make to your system in Platform Designer. You launch BSP Editor in integrated mode by specifying your Platform Designer system's `.qsys` file as the main input to the BSP generation. Nios V processor designs require integrated mode. Nios II processor designs require stand-alone mode.

[6] The Nios V processor does not support stand-alone mode. The Nios II processor does not support integrated mode.

To launch BSP Editor in integrated mode:

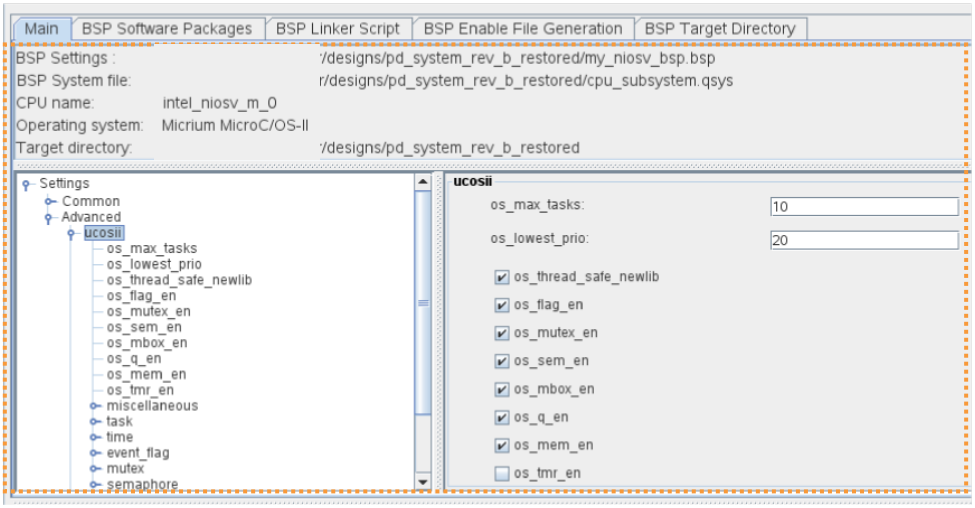
1. Open a Platform Designer system, as Creating or Opening a Platform Designer System on page 14 describes.
2. In Platform Designer, click **File ➤ New BSP**.
3. Specify the name and location for the new BSP settings file, and click the **Create** button. The **Create New BSP** dialog box opens.
4. For **System file**, specify the `.qsys` file that defines the target hardware platform for the current Platform Designer system. The **CPU name** and **BSP target directory** reflect the values in the `.qsys` file.

Figure 122. Create New BSP Dialog Box (Integrated Mode)



5. Specify options for the **Operating System** and **BSP target directory**, as Create New BSP Dialog Box on page 143 describes.
6. To create the BSP, click the **Create** button. The BSP Editor opens and displays the BSP Editor's **Main** tab in Platform Designer.

Figure 123. BSP Editor Main Tab



7. To configure the BSP, specify settings and options on the BSP Editor tabs. Refer to BSP Editor GUI on page 145 for details on these settings.
8. To generate the files for the BSP according to your specifications, click the **Generate BSP** button in BSP Editor. BSP Editor files do not automatically generate when you generate Platform Designer system files.

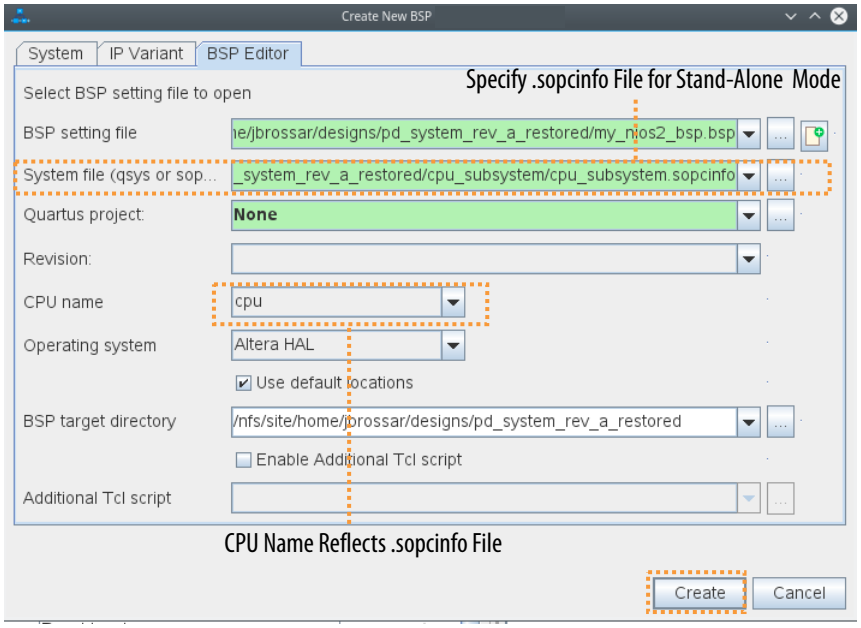
Launching BSP Editor in Stand-Alone Mode

Platform Designer generates a `<system_name>/.sopcinfo` file during HDL generation of the system. You can specify a `.sopcinfo` file and other BSP file saving options when defining a new BSP in stand-alone mode. Stand-alone mode is required for Nios II processor designs, and is not supported for Nios V processor designs.

Note: If you make any changes to your Platform Designer system after creating a BSP in stand-alone mode, you must regenerate the system's HDL and `.sopcinfo` files and create a new BSP to reflect any changes. The BSP does not update automatically with changes that you make to a Platform Designer system in stand-alone mode.

1. Open a Platform Designer system, as Creating or Opening a Platform Designer System on page 14 describes.
2. If generating a BSP for a Platform Designer system, click **Generate HDL** to generate the HDL and `.sopcinfo` files defining the target hardware for the system.
3. In Platform Designer, click **File ➤ New BSP**.
4. Specify the name and location for the new BSP settings file, and click the **Create** button. The **New Platform Designer BSP** dialog box opens.
5. For the **SOPC Information File name** option, specify the `.sopcinfo` file that step 2 generates. The **CPU name** and **BSP target directory** reflect the values in the `.sopcinfo` file.

Figure 124. Create New BSP Dialog Box (Stand-Alone Mode)



6. You can specify options for the **Operating System** and BSP file output, as Create New BSP Dialog Box on page 143 describes.
7. To create the BSP, click the **Create** button. The BSP Editor opens and displays the BSP Editor's **Main** tab in Platform Designer.
8. To configure the BSP, specify settings and options on the BSP Editor tabs. Refer to BSP Editor GUI on page 145.
9. To generate the files for the BSP according to your specifications, click the **Generate** button in BSP Editor.

2.1.1. Create New BSP Dialog Box

The **Create New BSP** dialog box allows you to specify the following options to define a new BSP:

Table 35. Create New BSP Dialog Box Settings

Name	Description
BSP setting file	Specifies the directory that contains the BSP settings file following BSP generation. By default: • `<project>/software/hal_bsp/settings.bsp` (HAL) • `<project>/software/ucOS_bsp/settings.bsp` (Micrium MicroC/OS-II)
System file (qsys or sopcinfo)	Specifies an existing `.qsys` or `.sopcinfo` file that defines the target hardware platform for a Platform Designer system or other hardware design. Platform Designer generates the `.qsys` file at system or component creation, and the `.sopcinfo` file during HDL generation of the system.
CPU name	Specifies the system CPU name if the **System file** defines a CPU, such as the Nios II processor. If the **System file** does not define a CPU, this setting is read-only. If the **System file** defines more than one CPU, you can select the CPU for this BSP.

continued...

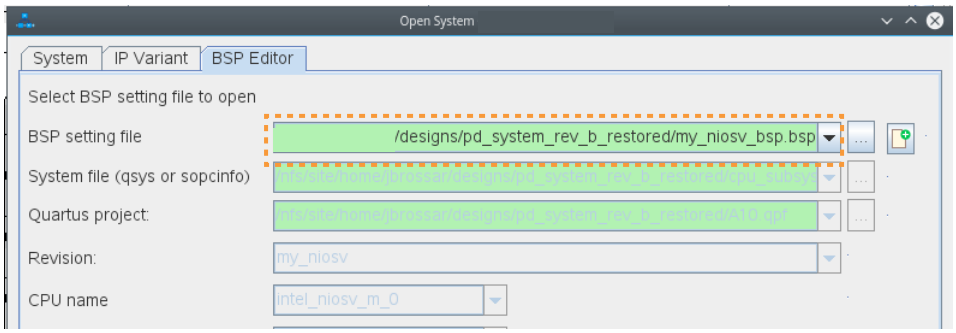
Name	Description
Operating System	Specifies the operating system of the **CPU name**. The operating system can be either **Altera HAL** or **Micrium MicroC/OS-II**. The BSP file output corresponds to your selection. You cannot change the **Operating system** for an existing BSP. Rather, you must create a new BSP that specifies a new **Operating System**.
Use default locations	Specifies whether BSP Editor uses default or custom file locations for the **BSP target directory** and **BSP settings file** storage. Disable this option to specify a custom location.
BSP target directory	Specifies the directory that contains the BSP files following BSP generation. By default: • `<project>/software/hal_bsp/` (HAL) • `<project>/software/ucOS_bsp/` (Micrium MicroC/OS-II)
Enable additional Tcl script file	Specifies an additional Tcl script (`.tcl`) that you can run to specify the BSP settings.

2.2. Opening a BSP from Platform Designer

To open and modify an existing BSP from within Platform Designer:

1. In Platform Designer, click **File ➤ Open**, and then click the **BSP Editor** tab.
2. For **BSP setting file**, click the **...** button to select the `.bsp` file for the BSP you want to open.

Figure 125. Open BSP Settings File



3. Click Open. The BSP loads in BSP Editor within Platform Designer.

2.3. Saving a BSP from Platform Designer

To save an existing BSP from within Platform Designer:

1. In Platform Designer, open an existing BSP (`.bsp`), as Opening a BSP from Platform Designer on page 144 describes.
   - Click **File ➤ Save** to save changes to the `.bsp` file to the current location.
   - Click **File ➤ Save BSP As** to save the `.bsp` file to a different location.

2.4. Exporting a BSP as Tcl from Platform Designer

You can export a BSP as a Tcl file from within Platform Designer. You can then run the Tcl script to load the BSP settings.

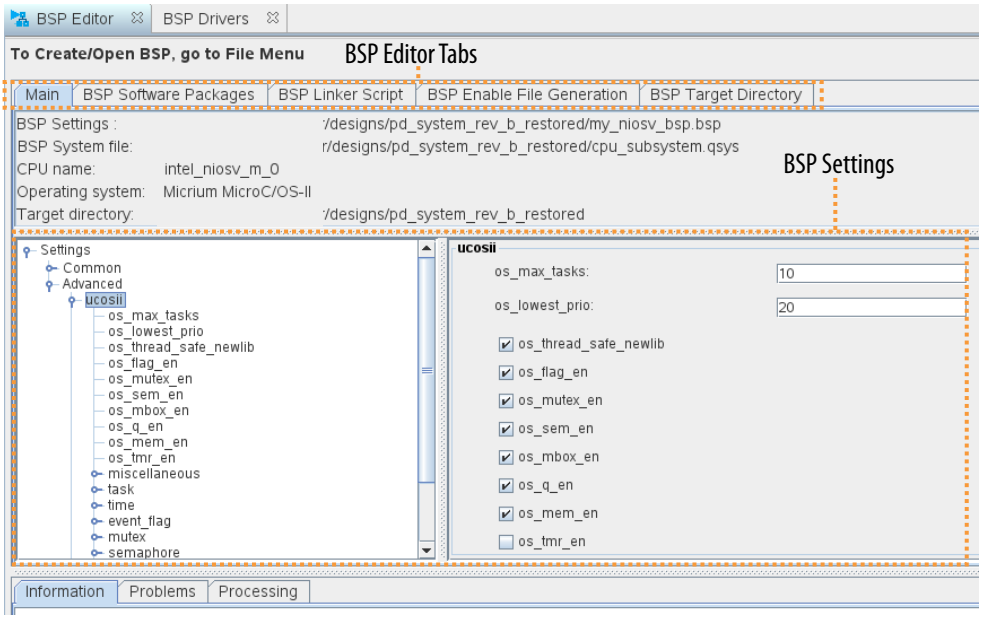
To export a BSP from Platform Designer as Tcl:

1. In Platform Designer, open an existing BSP (`.bsp`), as Opening a BSP from Platform Designer on page 144 describes.
2. Click **File ➤ Export BSP as Tcl**.
3. Specify the filename and location of the BSP Tcl file and click the **Save** button.
4. To reload the BSP, run the Tcl script file.

2.5. BSP Editor GUI

The BSP Editor GUI provides all the tools you need to create even the most complex BSPs. The BSP Editor GUI is divided into two sections. The top section contains the BSP Editor tabs for specifying the settings and other parameters that define the BSP. The **Messages** displays Information, Problems, and Processing messages about your actions in the BSP Editor.

Figure 126. BSP Editor GUI



Each tab of the BSP Editor GUI allows you to view and edit a particular aspect of the BSP, along with relevant command-line parameters and Tcl scripts. The settings that appear on the **Main**, **BSP Software Packages**, and **BSP Drivers** tabs correspond to the related command-line settings.

- Main Tab on page 146
- BSP Software Packages Tab on page 147
- BSP Drivers Tab on page 147
- BSP Linker Script Tab on page 148
- BSP Enable File Generation Tab on page 150
- BSP Target Directory Tab on page 151
- Messages Tabs on page 151

Note: For complete information about how to develop a BSP for a Nios II processor system or Nios V processor system, refer to the *Nios II Software Developer Handbook* or the *Nios V Processor Quick Start Guide*, respectively.

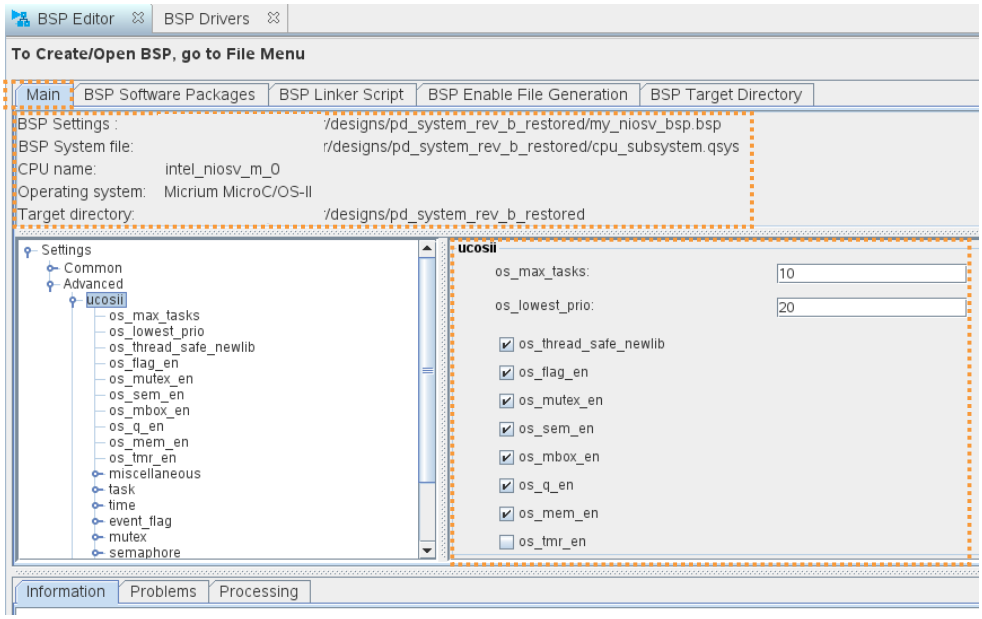
Related Information

- Nios II Software Developer Handbook
- Nios V Embedded Processor Design Handbook
- Nios V Processor Reference Guide

2.5.1. Main Tab

The **Main** tab presents general settings and parameters, and operating system settings, for the BSP. The BSP includes the following settings and parameters:

Figure 127. BSP Editor Main Tab



- The path to the `.qsys` or `.sopcinfo` file specifying the target hardware
- The CPU name
- The operating system and version

Note: You cannot change the operating system in an existing BSP. You must create a new BSP if you want to change the operating system.

- The **BSP target directory**—the destination for files that the BSP Editor copies and creates for your BSP
- BSP settings

The BSP settings appear in a tree structure. The settings are organized into **Common** and **Advanced** categories. Settings are further organized into functional groups. The available settings depend on the operating system.

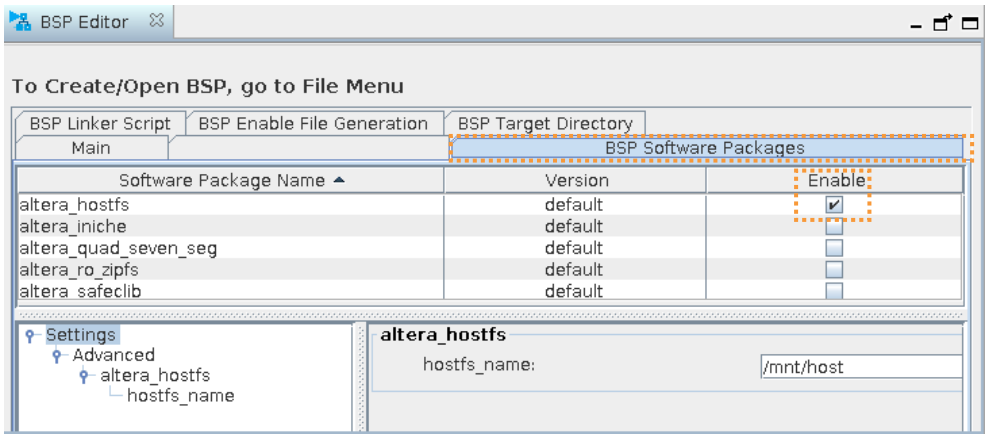
When you select a group of settings, the controls for those settings appear in the pane to the right of the tree. When you select a single setting, the pane shows the setting control, the full setting name, and the setting description.

2.5.2. BSP Software Packages Tab

The **BSP Software Packages** tab allows you to insert and remove software packages in your BSP, and control software package settings.

At the top of the **BSP Software Packages** tab is the software package table, listing each available software package. The table allows you to select the software package version, and enable or disable the software package. The operating system determines which software packages are available.

Figure 128. BSP Software Packages Tab



Many software packages define settings that you can control in your BSP. When you enable a software package, the available settings appear in a tree structure, organized into **Common** and **Advanced** settings.

When you select a group of settings, the controls for those settings appear in the pane to the right of the tree. When you select a single setting, the pane shows the setting control, the full setting name, and the setting description.

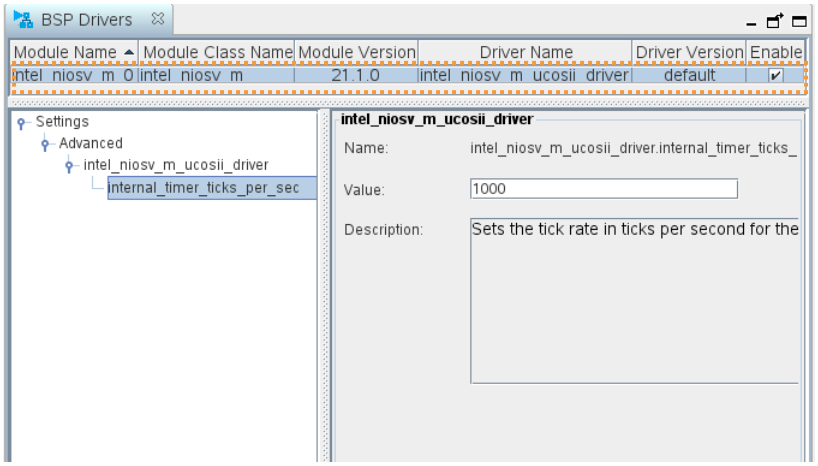
Enabling and disabling software packages and editing software package settings can have a profound impact on BSP behavior. Refer to the documentation for the specific software package for details.

2.5.3. BSP Drivers Tab

The **BSP Drivers** tab allows you to select, enable, and disable drivers for devices in your system, and control driver settings.

At the top of the **BSP Drivers** tab is the driver table, mapping components in the hardware system to drivers. The driver table shows components with driver support. Each component has a module name, module version, module class name, driver name, and driver version, determined by the contents of the hardware system. The table allows you to select the driver by name and version, as well as to enable or disable each driver.

Figure 129. BSP Drivers Tab



When you select a driver version, all instances of that driver in the BSP are set to the version you select. Only one version of a given driver can be used in an individual BSP.

Many drivers define settings that you can control in your BSP. Available driver settings appear in a tree structure below the driver table, organized into **Common** and **Advanced** settings.

When you select a group of settings, the controls for those settings appear in the pane to the right of the tree. When you select a single setting, the pane shows the setting control, the full setting name, and the setting description.

Enabling and disabling device drivers, changing drivers and driver versions, and editing driver settings, can have a profound impact on BSP behavior. Refer to the relevant component documentation and driver information for details.

Related Information

Embedded Peripherals IP User Guide
For more information about Intel FPGA components.

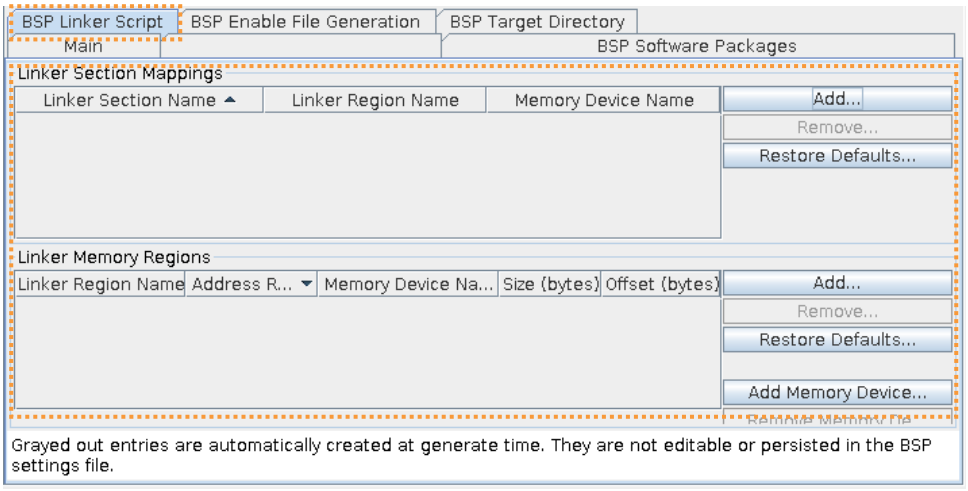
2.5.4. BSP Linker Script Tab

The **BSP Linker Script** tab allows you to view available memory in your hardware system, and examine and modify the arrangement and usage of linker regions in memory.

When you make a change to the memory configuration, the software build tools (SBT) validate your change.

Note: Rearranging linker regions and linker section mappings can have a significant impact on BSP behavior.

Figure 130. BSP Linker Script Tab



2.5.4.1. Linker Section Mappings

At the top of the **Linker Script** tab, the **Linker Section Mappings** table shows the mapping from linker sections to linker regions. You can edit the BSP linker section mappings using the following buttons located next to the linker section table:

- **Add**—Adds a linker section mapping to an existing linker region. The **Add** button opens the **Add Section Mapping** dialog box, where you specify a new section name and an existing linker region.
- **Remove**—Removes a mapping from a linker section to a linker region.
- **Restore Defaults**—Restores the section mappings to the default configuration set up at the time of BSP creation.

2.5.4.2. Linker Regions

At the bottom of the **Linker Script** tab, the **Linker Memory Regions** table shows all defined linker regions. Each row of the table shows one linker region, with its address range, memory device name, size, and offset into the selected memory device.

You reassign a defined linker region to a different memory device by selecting a different device name in the **Memory Device Name** column. The **Size** and **Offset** columns are editable. You can also edit the list of linker regions using the following buttons located next to the linker region table.

Note: Ensure that you specify the correct base address and memory size. If the base address or size of an external memory changes, you must edit the BSP manually to match. The SBT does not automatically detect changes in external memory devices, even if you update the BSP by creating a new settings file.

Table 36. Linker Memory Regions Table Buttons

Name	Description
Add	Adds a linker region in unused space on any existing device. The **Add** button opens the **Add Memory Region** dialog box, where you specify the memory device, the new memory region name, the region size, and the region's offset from the device base address.
Remove	Removes a linker region definition. Removing a region frees the region's memory space to be used for other regions.
Add Memory Device	Creates a linker region representing a memory device that is outside the hardware system. The button opens the **Add Memory Device** dialog box, where you can specify the device name, memory size, and base address. After you add the device, it appears in the linker region table, the **Memory Device Usage Table** dialog box, and the **Memory Map** dialog box. This functionality is equivalent to the `add_memory_device` Tcl command.

- **Restore Defaults**—restores the memory regions to the default configuration set up at the time of BSP creation.
- **Memory Usage**—Opens the **Memory Device Usage Table**. The **Memory Device Usage Table** allows you to view memory device usage by defined memory region. As memory regions are added, removed, and adjusted, each device's free memory, used memory, and percentage of available memory are updated. The rightmost column is a graphical representation of the device's usage, according to the memory regions assigned to it.
- **Memory Map**—Opens the **Memory Map** dialog box. The memory map allows you to view a map of system memory in the processor address space. The **Device** table is a read-only reference showing memories in the hardware system that are hosted by the selected processor. Devices are listed in memory address order.

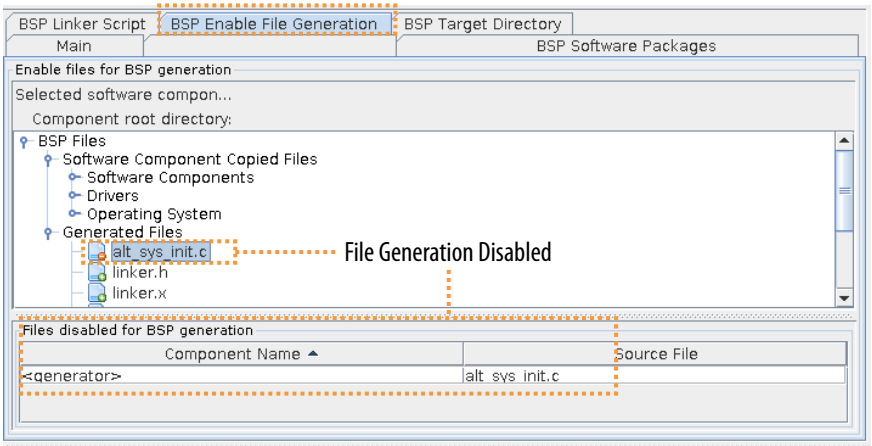
  To the right of the **Device** table is a graphical representation of the processor's memory space, showing the locations of devices in the table. Gaps indicate unmapped address space.

Note: This gap representation is not to scale.

2.5.5. BSP Enable File Generation Tab

The **BSP Enable File Generation** tab allows you to take ownership of specific BSP files that are normally generated by the SBT.

Figure 131. BSP Enable File Generation Tab



When you take ownership of a BSP file, you can modify it, and prevent the SBT from overwriting your modifications. The **BSP Enable File Generation** tab shows a tree view of all target files to be generated or copied when the BSP is generated. To disable generation of a specific file, expand the software component containing the file, expand any internal directory folders, select the file, and right-click. Each disabled file appears in a list at the bottom of the tab. This functionality is equivalent to the `set_ignore_file` Tcl command.

Note: If you take ownership of a BSP file, the SBT can no longer update it to reflect future changes in the underlying hardware. If you change the hardware, be sure to update the file manually.

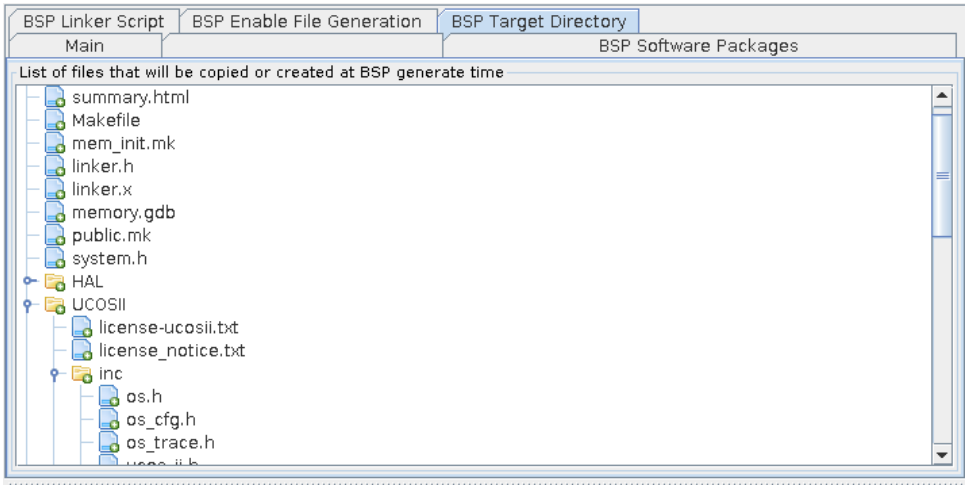
2.5.6. BSP Target Directory Tab

The **BSP Target Directory** tab is a read-only reference showing you what output to expect when the BSP is generated.

It does not depict the actual file system, but rather the files and directories to be created or copied when the BSP is generated. Each software component, including the operating system, drivers, and software packages, specifies source code to be copied into the BSP target directory. The files are generated in the directory specified on the **Main** tab.

When you generate the BSP, existing BSP files are overwritten, unless you disable generation of the file in the **BSP Enable File Generation** tab.

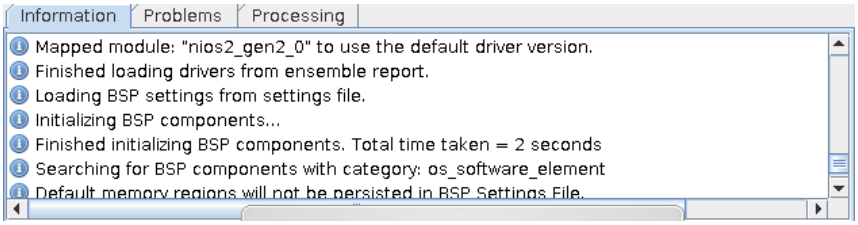
Figure 132. BSP Target Directory Tab



2.5.7. Messages Tabs

The messages area shows messages about the settings and commands that you select in the BSP Editor GUI. The messages area consists of the following tabs:

Figure 133. The Messages Area



- The **Information** tab—displays information messages about settings and commands.
- The **Problems** tab—displays problems to correct before BSP generation.
- The **Processing** tab—displays BSP Editor processing status messages.

Note: For complete information about how to develop a BSP for a Nios II processor system or Nios V processor system, refer to the *Nios II Software Developer Handbook* or the *Nios V Processor Quick Start Guide*, respectively.

Related Information

- Nios II Software Developer Handbook
- Nios V Embedded Processor Design Handbook
- Nios V Processor Reference Guide

2.6. Creating a Board Support Package with BSP Editor Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2023.04.03	23.1	• The product family name is updated to "Intel Agilex 7" to reflect the different family members.
2021.10.04	21.3	• Added *Launching BSP Editor in Integrated Mode* section to *Creating a BSP from Platform Designer* topic. • Updated *Creating a Board Support Package with BSP Editor* topic for Integrated Mode. • Updated *Create New BSP Dialog Box* topic for Integrated Mode. • Updated *Main Tab* topic for Integrated Mode. • Updated command name in *Saving a BSP from Platform Designer* topic. • Updated screenshot in *Opening a BSP from Platform Designer* topic. • Added references to the Nios V processor and documentation throughout.
2021.03.29	21.1	First version of chapter.

3. Creating Platform Designer Components

You can create and package IP components for use in a Platform Designer system. You can use a `_hw.tcl` file to describe IP components, interfaces and HDL files. Platform Designer provides the Component Editor to help you create a simple `_hw.tcl` file.

Related Information

- Avalon Interface Specifications
- Protocol Specifications
- Demo AXI Memory Example

3.1. Platform Designer Components

A Platform Designer component includes the following elements:

- Information about the component type, such as name, version, and author.
- HDL description of the component's hardware, including SystemVerilog, Verilog HDL, or VHDL files, or a blackbox implementation.
- A Synopsys® Design Constraints File `.sdc` that defines the component for synthesis and simulation.
- For _hw.tcl components, an `.ip` file that defines the component's parameters.
- A component's interfaces, including I/O signals.

3.1.1. Platform Designer Interface Support

Platform Designer is most effective when you use standard interfaces available in the IP Catalog to design custom IP. Standard interfaces operate efficiently with Intel FPGA IP components, and you can take advantage of the bus functional models (BFMs), monitors, and other verification IP that the IP Catalog provides.

Platform Designer also supports connections between Avalon and AXI interfaces by generating the interconnect logic. This logic enables you to handle the protocol difference. Platform Designer creates the interconnect logic by converting all the protocols to a proprietary packet format. Then, the tool routes the packet through network switches to the appropriate agents. Here, the packet converts to the agent's protocol.(7)

(7) This document now refers to the Avalon "host" and "agent," and the AXI "manager" and "subordinate," to replace formerly used terms. Refer to the current *AMBA AXI and ACE Protocol Specification* for the latest AMBA AXI and ACE protocol terminology.

Platform Designer supports the following interface specifications:

- Intel FPGA Avalon Memory-Mapped and Streaming
- Arm AMBA 3 AXI (version 1.0)
- Arm AMBA 4 AXI (version 2.0)
- Arm AMBA 4 AXI-Lite (version 2.0)
- Arm AMBA 4 AXI-Stream (version 1.0)
- Arm AMBA 3 APB (version 1.0)

IP components (IP Cores) can have any number of interfaces in any combination. Each interface represents a set of signals that you can connect within a Platform Designer system, or export outside of a Platform Designer system.

Platform Designer IP components can include the following interface types:

Table 37. IP Component Interface Types

Interface Type	Description
Memory-Mapped	Connects memory-referencing host devices with agent memory devices. Host devices can be processors and DMAs, while agent memory devices can be RAMs, ROMs, and control registers. Data transfers between Avalon Memory Mapped host and agent may be uni-directional (read only or write only), or bi-directional (read and write).
Streaming	Connects Avalon Streaming sources and sinks that stream unidirectional data, as well as high-bandwidth, low-latency IP components. Streaming creates datapaths for unidirectional traffic, including multichannel streams, packets, and DSP data. The Avalon interconnect is flexible and can implement on-chip interfaces for industry standard telecommunications and data communications cores, such as Ethernet, Interlaken, and video. You can define bus widths, packets, and error conditions.
Interrupts	Connects interrupt senders to interrupt receivers. Platform Designer supports individual, single-bit interrupt requests (IRQs). In the event that multiple senders assert their IRQs simultaneously, the receiver logic (typically under software control) determines which IRQ has highest priority, then responds appropriately.
Clocks	Connects clock output interfaces with clock input interfaces. Clock outputs can fan-out without the use of a bridge. A bridge is required only when a clock from an external (exported) source connects internally to more than one source.
Resets	Connects reset sources with reset input interfaces. If your system requires a particular positive-edge or negative-edge synchronized reset, Platform Designer inserts a reset controller to create the appropriate reset signal. If you design a system with multiple reset inputs, the reset controller ORs all reset inputs and generates a single reset output.
Conduits	Connects point-to-point conduit interfaces, or represent signals that you export from the Platform Designer system. Platform Designer uses conduits for component I/O signals that are not part of any supported standard interface. You can connect two conduits directly within a Platform Designer system as a point-to-point connection. Alternatively, you can export conduit interfaces and bring the interfaces to the top-level of the system as top-level system I/O. You can use conduits to connect to external devices, for example external DDR SDRAM memory, and to FPGA logic defined outside of the Platform Designer system.

Related Information

Exporting HDL Parameters to a System on page 189

3.1.2. Component Structure

Intel provides components automatically installed with the Quartus Prime software. To obtain a list of all Platform Designer-compliant components available from third-party IP developers, follow these steps:

1. Navigate to the Intel FPGA Find IP web page.
2. In the Find IP search results, click the **Platform Designer (Qsys) Compliant** column filter to show all Platform Designer-compliant components.
3. Further refine the search results by selecting the **End Market**, **Technology**, **Devices**, or **Provider** search filter. The **Provider** filter allows you to select specific third-party IP partners.

Every Platform Designer-compliant component is defined with a *<component_name>*_hw.tcl file. This file is a text file written in the Tcl scripting language that describes the component to Platform Designer. When you design your own custom component, you can create the _hw.tcl file manually, or by using the Platform Designer Component Editor.

The Component Editor simplifies the process of creating _hw.tcl files by creating a file that you can edit outside of the Component Editor to add advanced procedures. When you edit a previously saved _hw.tcl file, Platform Designer automatically backs up the earlier version as _hw.tcl~.

You can move component files into a new directory, such as a network location, so that other users can use the component in their systems. The _hw.tcl file contains relative paths to the other files, so if you move an _hw.tcl file, you should also move all the HDL and other files associated with it.

There are four component types:

- **Static**— static components always generate the same output, regardless of their parameterization. Components that instantiate static components must have only static children.
- **Generated**—generated component's fileset callback allows an instance of the component to create unique HDL design files based on the instance's parameter values.
- **Composed**—composed components are subsystems constructed from instances of other components. You can use a composition callback to manage the subsystem in a composed component.
- **Generic**—generic components allow instantiation of IP components without an HDL implementation. Generic components enable hierarchical isolation between system interconnect and IP components.

Related Information

- Create a Composed Component or Subsystem on page 196
- Add Component Instances to a Static or Generated Component on page 198

3.1.3. Component File Organization

A typical component uses the following directory structure where the names of the directories are not significant:

<component_directory>/

- *<hdl>/*—Contains the component HDL design files, for example `.v`, `.sv`, or `.vhd` files that contain the top-level module, along with any required constraint files.
- *<component_name>*`_hw.tcl`—The component description file.
- *<component_name>*`_sw.tcl`—The software driver configuration file. This file specifies the paths for the `.c` and `.h` files associated with the component, when required.
- *<software>/*—Contains software drivers or libraries related to the component.

Note: Refer to the *Nios V Processor Quick Start* for information about writing a device driver or software package suitable for use with the Nios V processor.

Related Information

- Nios V Embedded Processor Design Handbook
- Nios II Software Developer's Handbook
  Refer to the "Nios II Software Build Tools" and "Overview of the Hardware Abstraction Layer" chapters.
- Nios V Embedded Processor Design Handbook

3.1.4. Component Versions

Platform Designer systems support multiple versions of the same component within the same system; you can create and maintain multiple versions of the same component.

If you have multiple `_hw.tcl` files for components with the same NAME module properties and different VERSION module properties, both versions of the component are available.

If multiple versions of the component are available in the IP Catalog, you can add a specific version of a component by right-clicking the component, and then selecting **Add version** *<version_number>*.

3.1.4.1. Upgrade IP Components to the Latest Version

When you open a Platform Designer design, if Platform Designer detects IP components that require regeneration, the **Upgrade IP Cores** dialog box appears and allows you to upgrade outdated components.

Components that you must upgrade in order to successfully compile your design appear in red. Status icons indicate whether a component is currently being regenerated, the component is encrypted, or that there is not enough information to determine the status of component. To upgrade a component, in the **Upgrade IP Cores** dialog box, select the component that you want to upgrade, and then click **Upgrade**. The Quartus Prime software maintains a list of all IP components associated with your design on the **Components** tab in the Project Navigator.

3.2. Design Phases of an IP Component

When you define a component with the Platform Designer Component Editor, or a custom `_hw.tcl` file, you specify the information that Platform Designer requires to instantiate the component in a Platform Designer system and to generate the appropriate output files for synthesis and simulation.

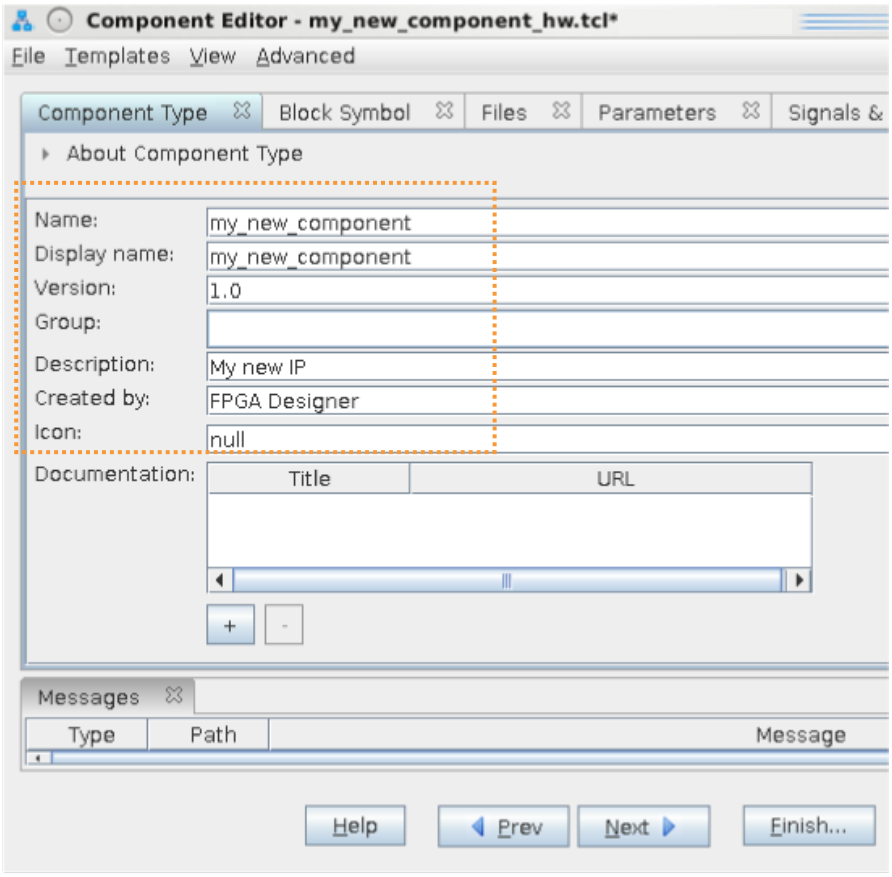
The following phases describe the process when working with components in Platform Designer:

- **Discovery**—During the discovery phase, Platform Designer reads the `_hw.tcl` file to identify information that appears in the IP Catalog, such as the component's name, version, and documentation URLs. Each time you open Platform Designer, the tool searches for the following file types using the default search locations and entries in the **IP Search Path**:
  - `_hw.tcl` files—Each `_hw.tcl` file defines a single component.
  - IP Index (`.ipx`) files—Each `.ipx` file indexes a collection of available components, or a reference to other directories to search.
- **Static Component Definition**—During the static component definition phase, Platform Designer reads the `_hw.tcl` file to identify static parameter declarations, interface properties, interface signals, and HDL files that define the component. At this stage of the life cycle, the component interfaces may be only partially defined.
- **Parameterization**—During the parameterization phase, after an instance of the component is added to a Platform Designer system, the user of the component specifies parameters with the component's parameter editor.
- **Validation**—During the validation phase, Platform Designer validates the values of each instance's parameters against the allowed ranges specified for each parameter. You can use callback procedures that run during the validation phase to provide validation messages. For example, if there are dependencies between parameters where only certain combinations of values are supported, you can report errors for the unsupported values.
- **Elaboration**—During the elaboration phase, Platform Designer queries the component for its interface information. Elaboration is triggered when an instance of a component is added to a system, when its parameters are changed, or when a system property changes. You can use callback procedures that run during the elaboration phase to dynamically control interfaces, signals, and HDL files based on the values of parameters. For example, interfaces defined with static declarations can be enabled or disabled during elaboration. When elaboration is complete, the component's interfaces and design logic must be completely defined.
- **Composition**—During the composition phase, a component can manipulate the instances in the component's subsystem. The **_hw.tcl** file uses a callback procedure to provide parameterization and connectivity of sub-components.
- **Generation**—During the generation phase, Platform Designer generates synthesis or simulation files for each component in the system into the appropriate output directories, as well as any additional files that support associated tools.

3.3. Creating IP Components in the Component Editor

The Platform Designer Component Editor allows you to create and package an IP component and parameterization GUI. When you use the Component Editor to define a component, Platform Designer writes the information to an `_hw.tcl` file.

Figure 134. Component Editor



The Platform Designer Component Editor allows you to perform the following tasks:

- Specify component's identifying information, such as name, version, author, etc.
- Specify the SystemVerilog, Verilog HDL, VHDL files, and constraint files that define the component for synthesis and simulation.
- Create an HDL template to define a component's interfaces, signals, and parameters.
- Set parameters on interfaces and signals that can alter the component's structure or functionality.

If you do not have a top-level HDL component file, you can use the Platform Designer Component Editor to add interfaces, signals, and parameters. In the Component Editor, the order in which the tabs appear reflects the recommended design flow for component development. You can use the **Prev** and **Next** buttons to guide you through the tabs.

In a Platform Designer system, the interfaces of a component are connected in the system, or exported as top-level signals from the system.

If the component is not based on an existing HDL file, enter the parameters, signals, and interfaces first, and then return to the **Files** tab to create the top-level HDL file template. When you click **Finish**, Platform Designer creates the component `_hw.tcl` file with the details that you enter in the Component Editor.

When you save the component, it appears in the IP Catalog.

If you require custom features that the Platform Designer Component Editor does not support, for example, an elaboration callback, use the Component Editor to create the `_hw.tcl` file, and then manually edit the file to complete the component definition.

Note: If you add custom coding to a component, do not open the component file in the Platform Designer Component Editor. The Platform Designer Component Editor overwrites your custom edits.

Example 4. Platform Designer Creates an _hw.tcl File from Entries in the Component Editor

```
#
# connection point clock
#
add_interface clock clock end
set_interface_property clock clockRate 0
set_interface_property clock ENABLED true

add_interface_port clock clk clk Input 1

#
# connection point reset
#
add_interface reset reset end
set_interface_property reset associatedClock clock
set_interface_property reset synchronousEdges DEASSERT
set_interface_property reset ENABLED true

add_interface_port reset reset_n reset_n Input 1

#
# connection point streaming
#
add_interface streaming avalon_streaming start
set_interface_property streaming associatedClock clock
set_interface_property streaming associatedReset reset
set_interface_property streaming dataBitsPerSymbol 8
set_interface_property streaming errorDescriptor ""
set_interface_property streaming firstSymbolInHighOrderBits true
set_interface_property streaming maxChannel 0
set_interface_property streaming readyLatency 0
set_interface_property streaming ENABLED true

add_interface_port streaming aso_data data Output 8
add_interface_port streaming aso_valid valid Output 1
add_interface_port streaming aso_ready ready Input 1

#
```

Related Information

Component Interface Tcl Reference on page 666

3.3.1. Save an IP Component and Create the _hw.tcl File

You save a component by clicking **Finish** in the Platform Designer Component Editor. The Component Editor saves the component as <*component_name*> `_hw.tcl` file.

You can use IP components with other applications, such as the C compiler and a board support package (BSP) generator. Intel recommends that you move `_hw.tcl` files and their associated files to an `ip/` directory within your Quartus Prime project directory.

When changing file locations, ensure that the file paths in the `_hw.tcl` file always point to the HDL code correctly. The Component Editor places the `_hw.tcl` file in the location of the `.qsys` file, rather than in the location of the HDL files. There can be a disconnect if you move the files without updating references. Check the `ADD_FILESET_FILE` property setting in the `_hw.tcl`.

Refer to *Creating a System with Platform Designer* for information on how to search for and add components to the IP Catalog for use in your designs.

Related Information

Creating a System with Platform Designer on page 11

3.3.2. Edit an IP Component with the Platform Designer Component Editor

In Platform Designer, you make changes to a component by right-clicking the component in the **System View** tab, and then clicking **Edit**. After making changes, click **Finish** to save the changes to the `_hw.tcl` file.

You can open an `_hw.tcl` file in a text editor to view the hardware Tcl for the component. If you edit the `_hw.tcl` file to customize the component with advanced features, you cannot use the Component Editor to make further changes without over-writing your customized file.

You cannot use the Component Editor to edit components installed with the Quartus Prime software, such as Intel-provided components. If you edit the HDL for a component and change the interface to the top-level module, you must edit the component to reflect the changes you make to the HDL.

3.3.3. Specify IP Component Type Information

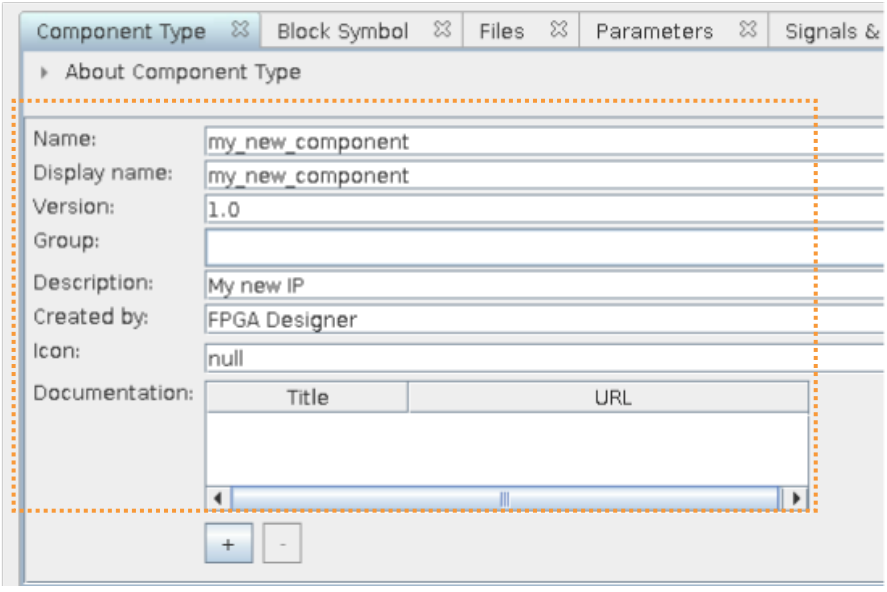
The **Component Type** tab in the Platform Designer Component Editor allows you to specify the following information about the component:

- **Name**—specifies the `_hw.tcl` filename, as well as in the top-level module name when you create a synthesis wrapper file for a non HDL-based component.
- **Display name**—identifies the component in the parameter editor, which you use to configure an instance of the component, and also appears in the IP Catalog under **Project** and on the **System View** tab.
- **Version**—specifies the component version number.

- **Group**—represents the category of the component in the list of available components in the IP Catalog. You can select an existing group from the list, or define a new group by typing a name in the **Group** box. Separating entries in the **Group** box with a slash defines a subcategory. For example, if you type **Memories and Memory Controllers/On-Chip**, the component appears in the IP Catalog under the **On-Chip** group, which is a subcategory of the **Memories and Memory Controllers** group. If you save the component in the project directory, the component appears in the IP Catalog in the group you specified under **Project**. Alternatively, if you save the component in the Quartus Prime installation directory, the component appears in the specified group under **IP Catalog**.
- **Description**—allows you to describe the component. This description appears when the user views the component details.
- **Created By**—specifies the component author name.
- **Icon**—specifies the relative path to an icon file (`.gif`, `.jpg`, or `.png` format) that represents the component and appears as the header in the parameter editor for the component. The default image is the Intel FPGA IP function icon.
- **Documentation**—specifies links to documentation for the component, and appears when you right-click the component in the IP Catalog, and then select **Details**.
  - To specify an Internet file, begin your path with `http://`, for example: `http://mydomain.com/datasheets/my_memory_controller.html`.
  - To specify a file in the file system, begin your path with `file:///` for Linux, and `file:////` for Windows; for example (Windows): `file:////company_server/datasheets my_memory_controller.pdf`.

Figure 135. Component Type Tab in the Component Editor

The **Display name**, **Group**, **Description**, **Created by**, **Icon**, and **Documentation** entries are optional.



When you use the Component Editor to create a component, it writes this basic component information in the _hw.tcl file. The `package require` command specifies the Quartus Prime software version that Platform Designer uses to create the _hw.tcl file, and ensures compatibility with this version of the Platform Designer API in future releases.

Example 5. _hw.tcl Created from Entries in the Component Type Tab

The component defines its basic information with various module properties using the `set_module_property` command. For example, `set_module_property NAME` specifies the name of the component, while `set_module_property VERSION` allows you to specify the version of the component. When you apply a version to the _hw.tcl file, it allows the file to behave exactly the same way in future releases of the Quartus Prime software.

```
# request TCL package from ACDS 14.0

package require -exact qsys 14.0

# demo_axi_memory

set_module_property DESCRIPTION \
"Demo AXI-3 memory with optional Avalon streaming port"

set_module_property NAME demo_axi_memory
set_module_property VERSION 1.0
set_module_property GROUP "My Components"
set_module_property AUTHOR Altera
set_module_property DISPLAY_NAME "Demo AXI Memory"
```

Related Information

Component Interface Tcl Reference on page 666

3.3.4. Create an HDL File in the Platform Designer Component Editor

If you do not have an HDL file for your component, you can use the Platform Designer Component Editor to define the component signals, interfaces, and parameters of your component, and then create a simple top-level HDL file.

You can then edit the HDL file to add the logic that describes the component's behavior.

1. In the Platform Designer Component Editor, specify the information about the component in the **Signals & Interfaces**, and **Interfaces**, and **Parameters** tabs.
2. Click the **Files** tab.
3. Click **Create Synthesis File from Signals**.
   The Component Editor creates an HDL file from the specified signals, interfaces, and parameters, and the .v file appears in the **Synthesis File** table.

Related Information

Specify Synthesis and Simulation Files in the Platform Designer Component Editor on page 167

3.3.5. Defining HDL Parameters in _hw.tcl

Platform Designer supports the ability to reconfigure features of parameterized modules, such as data bus width or FIFO depth. Platform Designer creates an HDL wrapper when you perform **Generate HDL**. By modifying your `_hw.tcl` files to specify parameter attributes and port properties, you can use Platform Designer to generate reusable RTL.

1. To define an alterable HDL parameter, you must declare the following two attributes for the parameter:
   - `set_parameter_property <parameter_name> HDL_PARAMETER true`
   - `set_parameter_property <parameter_name> AFFECTS_GENERATION false`
2. To have parameterized ports created in the instantiation wrapper, you can either set the width expression when adding a port to an interface, or set the width expression in the port property in `_hw.tcl`:
   - To set the width expression when adding a port:

     ```
     add_interface_port <interface> <port> <signal_type> <direction>
     <width_expression>
     ```

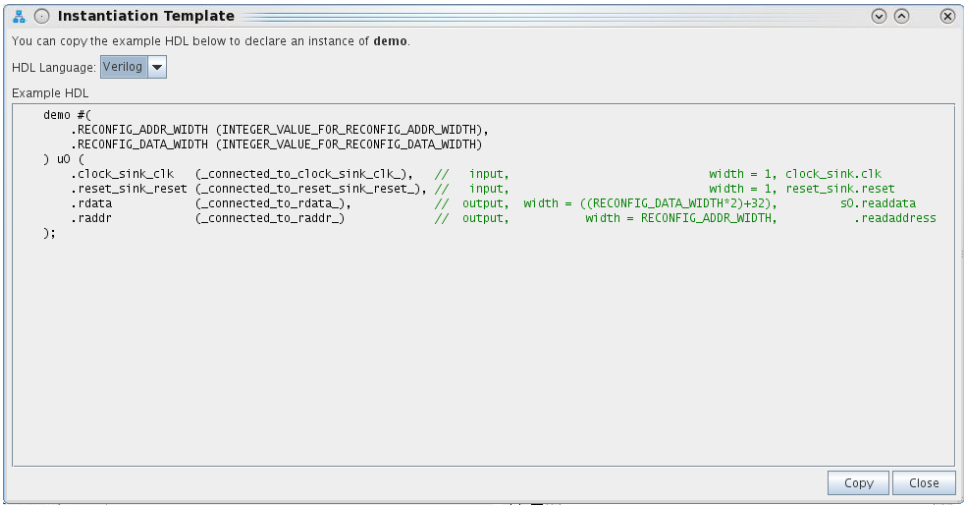
   - To set the width expression in the port property:

     ```
     set_port_property <port> WIDTH_EXPR <width_expression>
     ```

3. To create and generate the IP component in Platform Designer editor, click the **Open System ➤ IP Variant** tab, specify the new IP variant name in the **IP Variant** field and choose the `_hw.tcl` file that defines user alterable HDL parameters in the **Component type** field.
4. Click **Generate HDL** to generate the IP core. Platform Designer generates a parameterized HDL module for you directly.

To instantiate the IP component in your HDL file, click **Generate ➤ Show Instantiation Template** in the Platform Designer editor to display an instantiation template in Verilog or VHDL. Now you can instantiate the IP core in your top-level design HDL file with the template code.

Figure 136. Instantiation Template Dialog Box



The following sample contains _hw.tcl to set exportable width values:

Example 6. Sample _hw.tcl Component with User Alterable Expressions

```
package require -exact qsys 17.1

set_module_property NAME demo
set_module_property DISPLAY_NAME "Demo"
set_module_property ELABORATION_CALLBACK elaborate

# add exportable hdl parameter RECONFIG_DATA_WIDTH
add_parameter RECONFIG_DATA_WIDTH INTEGER 48
set_parameter_property RECONFIG_DATA_WIDTH AFFECTS_GENERATION false
set_parameter_property RECONFIG_DATA_WIDTH HDL_PARAMETER true

# add exportable hdl parameter RECONFIG_ADDR_WIDTH
add_parameter RECONFIG_ADDR_WIDTH INTEGER 32
set_parameter_property RECONFIG_ADDR_WIDTH AFFECTS_GENERATION false
set_parameter_property RECONFIG_ADDR_WIDTH HDL_PARAMETER true

# add non-exportable hdl parameter
add_parameter l_addr INTEGER 32
set_parameter l_addr HDL_PARAMETER false

# add interface
add_interface s0 conduit end

proc elaborate {} {
   add_interface_port s0 rdata readdata output "reconfig_data_width*2 + l_addr"
   add_interface_port s0 raddr readaddress output [get_parameter_value
RECONFIG_ADDR_WIDTH]
   set_port_property raddr WIDTH_EXPR "RECONFIG_ADDR_WIDTH"
}
```

Related Information

Exporting HDL Parameters to a System on page 189

3.3.6. Declaring SystemVerilog Interfaces in _hw.tcl

Platform Designer supports interfaces written in SystemVerilog.

The following example is `_hw.tcl` for a module with a SystemVerilog interface. The sample code is divided into parts 1 and 2.

Part 1 defines the normal array of parameters, Platform Designer interface, and ports

Example 7. Example Part 1: Parameters, Platform Designer Interface, and Ports in _hw.tcl

```
# request TCL package from ACDS 23.1
#
package require -exact qsys 23.1
#
# module ram_ip_sv_ifc_hw
#
set_module_property DESCRIPTION ""
set_module_property NAME ram_ip_sv_ifc_hw
set_module_property VERSION 1.0
set_module_property INTERNAL false
set_module_property OPAQUE_ADDRESS_MAP true
set_module_property AUTHOR ""
set_module_property DISPLAY_NAME ram_ip_hw_with_SV_d0
set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
set_module_property EDITABLE true
set_module_property REPORT_TO_TALKBACK false
set_module_property ALLOW_GREYBOX_GENERATION false
set_module_property REPORT_HIERARCHY false

# Part 1 – Add parameter, platform designer interface and ports
# Adding parameter
add_parameter my_interface_parameter STRING "" "I am an interface parameter"

# Adding platform designer interface clk
add_interface clk clock end
set_interface_property clk clockRate 0
# Adding ports to clk interface
add_interface_port clk clk clk Input 1

# Adding platform designer interface reset
add_interface reset reset end
set_interface_property reset associatedClock clk
#Adding ports to reset interface
add_interface_port reset reset reset Input 1

# Adding platform designer interface avalon_slave
add_interface avalon_slave avalon end
set_interface_property avalon_slave addressUnits WORDS
set_interface_property avalon_slave associatedClock clk
set_interface_property avalon_slave associatedReset reset
# Adding ports to avalon_slave interface
add_interface_port avalon_slave address address Input 8
add_interface_port avalon_slave write write Input 1
add_interface_port avalon_slave readdata readdata Output 32
add_interface_port avalon_slave writedata writedata Input 32
```

Part 2 defines the interface name, ports, and parameters of the SystemVerilog interface.

Example 8. Example Part 2: SystemVerilog Interface Parameters in _hw.tcl

```
# Part 2 – Adding SV interface and its properties.
# Adding SV interface
add_sv_interface bus mem_ifc

set_sv_interface_property bus USE_ALL_PORTS True
# Setting the parameter property to add SV interface parameters
set_parameter_property my_interface_parameter SV_INTERFACE_PARAMETER bus
# Setting the port properties to add them to SV interface port
set_port_property clk SV_INTERFACE_PORT bus
```

```
set_port_property reset SV_INTERFACE_PORT bus

# Setting the port properties to add them as signals inside SV interface
set_port_property address SV_INTERFACE_SIGNAL bus
set_port_property write SV_INTERFACE_SIGNAL bus
set_port_property writedata SV_INTERFACE_SIGNAL bus
set_port_property readdata SV_INTERFACE_SIGNAL bus

#Adding the SV Interface File
#Adding ram_ip files
add_fileset synthesis_fileset QUARTUS_SYNTH
set_fileset_property synthesis_fileset TOP_LEVEL ram_ip
add_fileset_file ram_ip.sv SYSTEM_VERILOG PATH ram_ip.sv
add_fileset_file mem_ifc.sv SYSTEM_VERILOG PATH \
     mem_ifc.sv SYSTEMVERILOG_INTERFACE
```

Related Information

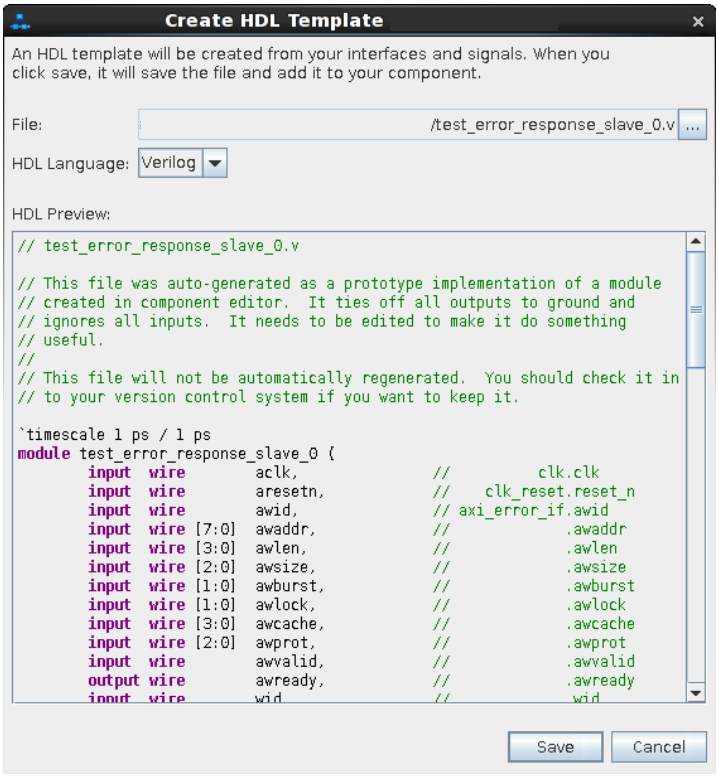
SystemVerilog Interface Commands on page 753

3.3.7. Create an HDL File Using a Template in the Platform Designer Component Editor

You can use a template to create interfaces and signals for your Platform Designer component

1. In Platform Designer, click **New Component** in the IP Catalog.
2. On the **Component Type** tab, define your component information in the **Name**, **Display Name**, **Version**, **Group**, **Description**, **Created by**, **Icon**, and **Documentation** boxes.
3. Click **Finish**.
   Your new component appears in the IP Catalog under the category that you define for "Group".
4. In Platform Designer, right-click your new component in the IP Catalog, and then click **Edit**.
5. In the Platform Designer Component Editor, click any interface from the Templates drop-down menu.
   The Component Editor fills the **Signals** and **Interfaces** tabs with the component interface template details.
6. On the **Files** tab, click **Create Synthesis File from Signals**.
7. Do the following in the **Create HDL Template** dialog box as shown below:
   a. Verify that the correct files appears in **File** path, or browse to the location where you want to save your file.
   b. Select the HDL language.
   c. Click **Save** to save your new interface, or **Cancel** to discard the new interface definition.

Create HDL Template Dialog Box



8. Verify the ***<component_name>*.v** file appears in the **Synthesis Files** table on the **Files** tab.

Related Information

Specify Synthesis and Simulation Files in the Platform Designer Component Editor on page 167

3.3.8. Specify Synthesis and Simulation Files in the Platform Designer Component Editor

The **Files** tab in the Platform Designer Component Editor allows you to specify synthesis and simulation files for your custom component.

If you already have an HDL file that describes the behavior and structure of your component, you can specify those files on the **Files** tab.

If you do not yet have an HDL file, you can specify the signals, interfaces, and parameters of the component in the Component Editor, and then use the **Create Synthesis File from Signals** option on the **Files** tab to create the top-level HDL file. The Component Editor generates the `_hw.tcl` commands to specify the files.

Note: After you analyze the component's top-level HDL file (on the **Files** tab), you cannot add or remove signals or change the signal names on the **Signals & Interfaces** tab. If you need to edit signals, edit your HDL source, and then click **Create Synthesis File from Signals** on the **Files** tab to integrate your changes.

A component uses filesets to specify the different sets of files that you can generate for an instance of the component. The supported fileset types are: `QUARTUS_SYNTH`, for synthesis and compilation in the Quartus Prime software, `SIM_VERILOG`, for Verilog HDL simulation, and `SIM_VHDL`, for VHDL simulation.

In an `_hw.tcl` file, you can add a fileset with the `add_fileset` command. You can then list specific files with the `add_fileset_file` command. The `add_fileset_property` command allows you to add properties such as `TOP_LEVEL`.

You can populate a fileset with a fixed list of files, add different files based on a parameter value, or even generate an HDL file with a custom HDL generator function outside of the `_hw.tcl` file.

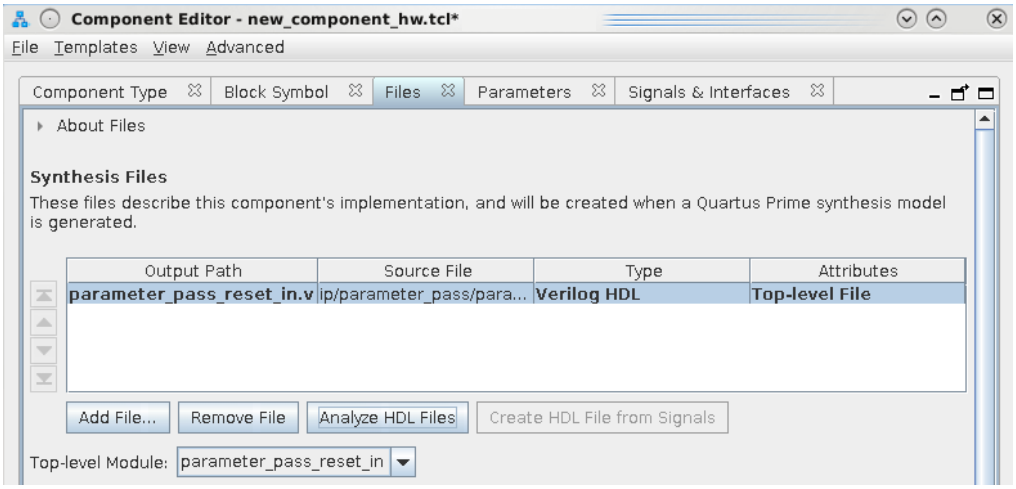
Related Information

- Create an HDL File in the Platform Designer Component Editor on page 162
- Create an HDL File Using a Template in the Platform Designer Component Editor on page 166

3.3.8.1. Specify HDL Files for Synthesis in the Platform Designer Component Editor

In the Platform Designer Component Editor, you can add HDL files and other support files with options on the `Files` tab.

Figure 137. Using HDL Files to Define a Component



A component must specify an HDL file as the top-level file. The top-level HDL file contains the top-level module. The **Synthesis Files** list may also include supporting HDL files, such as timing constraints, or other files required to successfully synthesize and compile in the Quartus Prime software. The synthesis files for a component are copied to the generation output directory during Platform Designer system generation.

3.3.8.2. Analyze Synthesis Files in the Platform Designer Component Editor

After you specify the top-level HDL file in the Platform Designer Component Editor, click **Analyze Synthesis Files** to analyze the parameters and signals in the top-level, and then select the top-level module from the **Top Level Module** list. If there is a single module or entity in the HDL file, Platform Designer automatically populates the **Top-level Module** list.

Once analysis is complete and the top-level module is selected, you can view the parameters and signals on the **Parameters** and **Signals & Interfaces** tabs. The Component Editor may report errors or warnings at this stage, because the signals and interfaces are not yet fully defined.

Note: At this stage in the Component Editor flow, you cannot add or remove parameters or signals created from a specified HDL file without editing the HDL file itself.

The synthesis files are added to a fileset with the name `QUARTUS_SYNTH` and type `QUARTUS_SYNTH` in the `_hw.tcl` file created by the Component Editor. The top-level module is used to specify the `TOP_LEVEL` fileset property. Each synthesis file is individually added to the fileset. If the source files are saved in a different directory from the working directory where the `_hw.tcl` is located, you can use standard fixed or relative path notation to identify the file location for the `PATH` variable.

Example 9. _hw.tcl Created from Entries in the Files tab in the Synthesis Files Section

```
# file sets

add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
set_fileset_property QUARTUS_SYNTH TOP_LEVEL demo_axi_memory

add_fileset_file demo_axi_memory.sv
SYSTEM_VERILOG PATH demo_axi_memory.sv

add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v
```

Related Information

- Specify HDL Files for Synthesis in the Platform Designer Component Editor on page 168
- Component Interface Tcl Reference on page 666

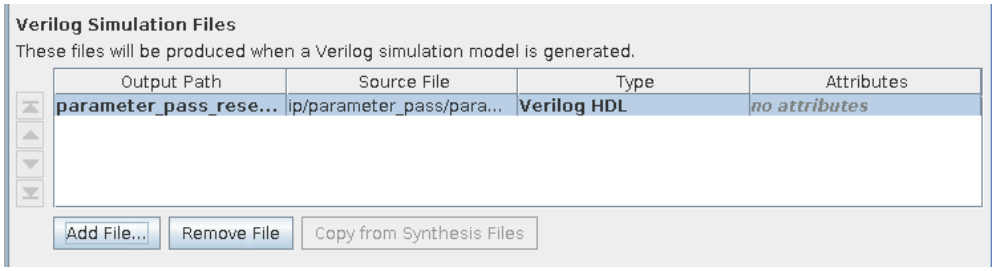
3.3.8.3. Specify Files for Simulation in the Component Editor

To support Platform Designer system generation for your custom component, you must specify VHDL or Verilog simulation files.

You can choose to generate Verilog or VHDL simulation files. In most cases, these files are the same as the synthesis files. If there are simulation-specific HDL files or simulation models, you can use them in addition to, or in place of the synthesis files. To use your synthesis files as your simulation files, click **Copy From Synthesis Files** on the **Files** tab in the Platform Designer Component Editor.

Note: The order that you add files to the fileset determines the order of compilation. For VHDL filesets with VHDL files, you must add the files bottom-up, adding the top-level file last.

Figure 138. Specifying the Simulation Output Files on the Files Tab



You specify the simulation files in a similar way as the synthesis files with the fileset commands in a `_hw.tcl` file. The code example below shows `SIM_VERILOG` and `SIM_VHDL` filesets for Verilog and VHDL simulation output files. In this example, the same Verilog files are used for both Verilog and VHDL outputs, and there is one additional SystemVerilog file added. This method works for designers of Verilog IP to support users who want to generate a VHDL top-level simulation file when they have a mixed-language simulation tool and license that can read the Verilog output for the component.

Example 10. _hw.tcl Created from Entries in the Files tab in the Simulation Files Section

```
add_fileset SIM_VERILOG SIM_VERILOG "" ""
set_fileset_property SIM_VERILOG TOP_LEVEL demo_axi_memory
add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv

add_fileset_file demo_axi_memory.sv SYSTEM_VERILOG PATH \
demo_axi_memory.sv

add_fileset SIM_VHDL SIM_VHDL "" ""
set_fileset_property SIM_VHDL TOP_LEVEL demo_axi_memory
set_fileset_property SIM_VHDL ENABLE_RELATIVE_INCLUDE_PATHS false

add_fileset_file demo_axi_memory.sv SYSTEM_VERILOG PATH \
demo_axi_memory.sv

add_fileset_file single_clk_ram.v VERILOG PATH single_clk_ram.v

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv
```

Related Information

Component Interface Tcl Reference on page 666

3.3.8.4. Include an Internal Register Map Description in the .svd for Agent Interfaces Connected to an HPS Component

Platform Designer supports the ability for IP component designers to specify register map information on their agent interfaces. This allows components with agent interfaces that are connected to an HPS component to include their internal register description in the generated `.svd` file.

To specify their internal register map, the IP component designer must write and generate their own `.svd` file and attach it to the agent interface using the following command:

`set_interface_property` <*agent interface*> `CMSIS_SVD_FILE` <*file path*>

The `CMSIS_SVD_VARIABLES` interface property allows for variable substitution inside the `.svd` file. You can dynamically modify the character data of the `.svd` file by using the `CMSIS_SVD_VARIABLES` property.

Example 11. Setting the CMSIS_SVD_VARIBLES Interface Property

For example, if you set the `CMSIS_SVD_VARIABLES` in the `_hw tcl` file, then in the `.svd` file if there is a variable `{width}` that describes the element `<size>${width}</size>`, it is replaced by `<size>23</size>` during generation of the `.svd` file. Note that substitution works only within character data (the data enclosed by <element>...</element>) and not on element attributes.

```
set_interface_property <interface name> \
CMSIS_SVD_VARIABLES "{width} {23}"
```

Related Information

- Component Interface Tcl Reference on page 666
- CMSIS - Cortex Microcontroller Software

3.3.9. Add Signals and Interfaces in the Platform Designer Component Editor

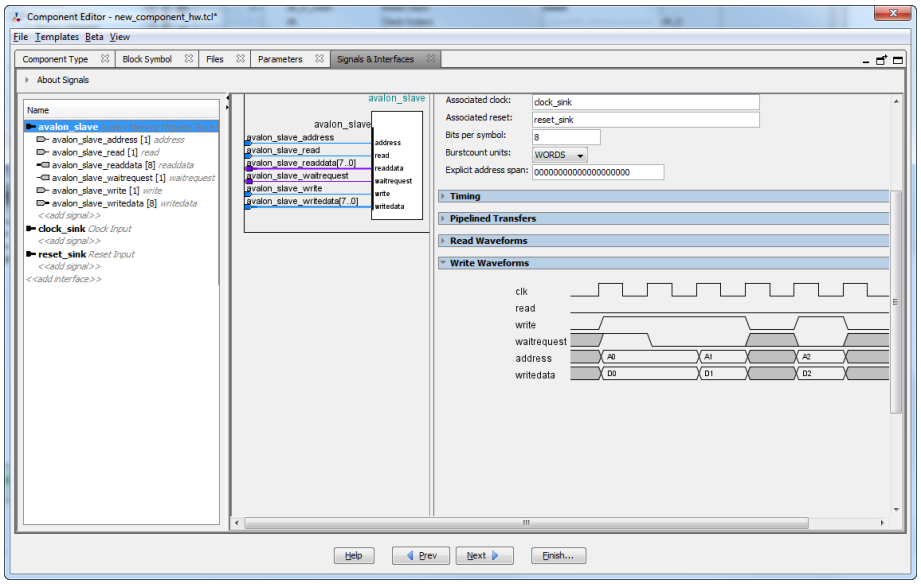
In the Platform Designer Component Editor, the **Signals & Interfaces** tab allows you to add signals and interfaces for your custom IP component.

As you select interfaces and associated signals, you can customize the parameters. Messages appear as you add interfaces and signals to guide you when customizing the component. In the parameter editor, a block diagram displays for each interface. Some interfaces display waveforms to show the timing of the interface. If you update timing parameters, the waveforms update automatically.

1. In Platform Designer, click ***New Component*** in the IP Catalog.
2. In the Platform Designer Component Editor, click the **Signals & Interfaces** tab.
3. To add an interface, click <<*add interface*>> in the left pane.
   A drop-down list appears where you select the interface type.
4. Select an interface from the drop-down list.
   The selected interface appears in the parameter editor where you can specify its parameters.
5. To add signals for the selected interface click <<*add signal*>> below the selected interface.
6. To move signals between interfaces, select the signal, and then drag it to another interface.
7. To rename a signal or interface, select the element, and then press **F2**.
8. To remove a signal or interface, right-click the element, and then click **Remove**.

Alternatively, to remove an signal or interface, you can select the element, and then press **Delete**. When you remove an interface, Platform Designer also removes all of its associated signals.

Figure 139. Platform Designer Signals & Interfaces tab



3.3.10. Specify Parameters in the Platform Designer Component Editor

Components can include parameterized HDL, which allow users of the component flexibility in meeting their system requirements. For example, a component with a configurable memory size or data width, allows using one HDL implementation in different systems, each with unique parameters values.

The **Parameters** tab allows you specify the parameters that are used to configure instances of the component in a Platform Designer system. You can specify various properties for each parameter that describe how to display and use the parameter. You can also specify a range of allowed values that are checked during the validation phase. The **Parameters** table displays the HDL parameters that are declared in the top-level HDL module. If you have not yet created the top-level HDL file, the top-level synthesis file template created from the **Files** tab include the parameters that you create on the **Parameters** tab.

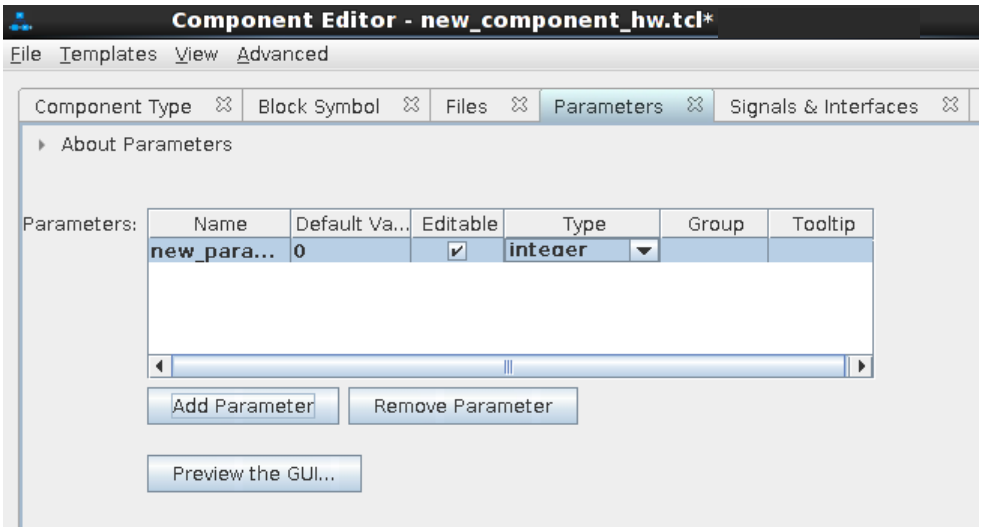
When the component includes HDL files, the parameters match those defined in the top-level module, and you cannot add or remove them on the **Parameters** tab. To add or remove the parameters, edit your HDL source, and then re-analyze the file.

If you create a top-level template HDL file for synthesis with the Component Editor, you can remove the newly-created file from the **Synthesis Files** list on the **Files** tab, make your parameter changes, and then re-analyze the top-level synthesis file.

You can use the **Parameters** table to specify the following information about each parameter:

- **Name**—specifies the parameter name.
- **Default Value**—sets the default value for new instances of the component.
- **Editable**—specifies if the user can edit the parameter value.
- **Type**—defines the parameter type as string, integer, boolean, std_logic, logic vector, natural, or positive.
- **Group**—groups parameters in the parameter editor.
- **Tooltip**—adds a description of the parameter that appears when the user of the component points to the parameter in the editor.

Figure 140. Parameters Tab in the Platform Designer Components Editor



On the **Parameters** tab, you can click **Preview the GUI** at any time to see how the declared parameters appear in the parameter editor. Parameters with their default values appear with checks in the **Editable** column. Editable parameters cannot contain computed expressions. You can group parameters under a common heading or section in the editor with the **Group** column, and a tooltip helps users of the component understand the function of the parameter. Various parameter properties allow you to customize the component's parameter editor, such as specifying parameter option controls, or displaying an image.

Example 12. _hw.tcl Created from Entries in the Parameters Tab

In this example, the first `add_parameter` command includes commonly-specified properties. The `set_parameter_property` command specifies each property individually. The **Tooltip** column on the **Parameters** tab maps to the `DESCRIPTION` property, and there is an additional unused `UNITS` property created in the code. The `HDL_PARAMETER` property specifies that the value of the parameter is specified in the HDL instance wrapper when creating instances of the component. The **Group** column in the **Parameters** tab maps to the display items section with the `add_display_item` commands. If you want to expose the HDL parameter so that you can overwrite the value when you instantiate the module, set `AFFECTS_GENERATION` to false.

Note: If a parameter <*n*> defines the width of a signal, the signal width must follow the format <*n*-1> : 0.

```
#
# parameters
#
add_parameter AXI_ID_W INTEGER 4 "Width of ID fields"
set_parameter_property AXI_ID_W DEFAULT_VALUE 4
set_parameter_property AXI_ID_W DISPLAY_NAME AXI_ID_W
set_parameter_property AXI_ID_W TYPE INTEGER
set_parameter_property AXI_ID_W UNITS None
set_parameter_property AXI_ID_W DESCRIPTION "Width of ID fields"
set_parameter_property AXI_ID_W HDL_PARAMETER true
add_parameter AXI_ADDRESS_W INTEGER 12
set_parameter_property AXI_ADDRESS_W DEFAULT_VALUE 12

add_parameter AXI_DATA_W INTEGER 32
...
#
# display items
#
add_display_item "AXI Port Widths" AXI_ID_W PARAMETER ""
```

Note: If an AXI subordinate's ID bit width is smaller than required for your system, the AXI subordinate response may not reach all AXI managers. The formula of an AXI subordinate ID bit width is calculated as follows:

```
maximum_manager_id_width_in_the_interconnect + log2 = <value>
```

For example, if an AXI subordinate connects to three AXI managers and the maximum AXI manager ID length of the three managers is 5 bits, then the AXI subordinate ID is 7 bits, and is calculated as follows:

$$5 \text{ bits} + 2 \text{ bits } (\log_2(3 \text{ managers})) = 7$$

Platform Designer refers to AXI interface parameters to build AXI interconnect. If these parameter settings are incompatible with the component's HDL behavior, Platform Designer interconnect and transactions may not work correctly. To prevent unexpected interconnect behavior, you must set the AXI component parameters.

Table 38. AXI Manager Parameters

AXI Manager Parameters	Description
`readIssuingCapability`	The maximum number of outstanding read transactions for a manager.
`writeIssuingCapability`	The maximum number of outstanding write transactions for a manager.
`combinedIssuingCapability`	The maximum number of outstanding transactions for a manager.

Table 39. AXI Subordinate Parameters

AXI Subordinate Parameters	Description
`readAcceptanceCapability`	The maximum number of outstanding read commands that a subordinate can accept.
`writeAcceptanceCapability`	The maximum number of outstanding write transactions that a subordinate can accept.
`combinedAcceptanceCapability`	The maximum number of outstanding transactions that a subordinate can accept.
`readDataReorderingDepth`	The number of outstanding read transactions for which a subordinate interface can transmit data. If `readDataReorderingDepth = 1`, the subordinate processes all transactions in order.

Related Information

Component Interface Tcl Reference on page 666

3.3.10.1. Valid Ranges for Parameters in the _hw.tcl File

In the `_hw.tcl` file, you can specify valid ranges for parameters.

Platform Designer validation checks each parameter value against the `ALLOWED_RANGES` property. If the values specified are outside of the allowed ranges, Platform Designer displays an error message. Specifying choices for the allowed values enables users of the component to choose the parameter value from controls in the parameter editor GUI, instead of entering a value.

The `ALLOWED_RANGES` property is a list of valid ranges, where each range is a single value, or a range of values defined by a start and end value.

Table 40. ALLOWED_RANGES Property

ALLOWED_RANGES Property	Values
`{a b c}`	a, b, or c
`{"No Control" "Single Control" "Dual Controls"}`	Unique string values. Quotation marks are required if the strings include spaces .
`{1 2 4 8 16}`	1, 2, 4, 8, or 16
`{1:3}`	1 through 3, inclusive.
`{1 2 3 7:10}`	1, 2, 3, or 7 through 10 inclusive.

Related Information

Declare Parameters with Custom _hw.tcl Commands on page 177

3.3.10.2. Types of Platform Designer Parameters

Platform Designer uses the following parameter types: user parameters, system information parameters, and derived parameters.

Platform Designer User Parameters on page 176
Platform Designer System Information Parameters on page 176
Parameterized Parameter Widths on page 176

Platform Designer Derived Parameters on page 176

Related Information

Declare Parameters with Custom _hw.tcl Commands on page 177

3.3.10.2.1. Platform Designer User Parameters

User parameters are parameters that users of a component can control, and appear in the parameter editor for instances of the component. User parameters map directly to parameters in the component HDL. For user parameter code examples, such as AXI_DATA_W and ENABLE_STREAM_OUTPUT, refer to *Declaring Parameters with Custom hw.tcl Commands*.

3.3.10.2.2. Platform Designer System Information Parameters

A SYSTEM_INFO parameter is a parameter whose value is set automatically by the Platform Designer system. When you define a SYSTEM_INFO parameter, you provide an `information type`, and additional arguments.

For example, you can configure a parameter to store the clock frequency driving a clock input for your component. To do this, define the parameter as SYSTEM_INFO of type CLOCK_RATE:

```
set_parameter_property <param> SYSTEM_INFO CLOCK_RATE
```

You then set the name of the clock interface as the SYSTEM_INFO argument:

```
set_parameter_property <param> SYSTEM_INFO_ARG <clkname>
```

3.3.10.2.3. Parameterized Parameter Widths

Platform Designer allows a `std_logic_vector` parameter to have a width that is defined by another parameter, similar to derived parameters. The width can be a constant or the name of another parameter.

3.3.10.2.4. Platform Designer Derived Parameters

Derived parameter values are calculated from other parameters during the Elaboration phase, and are specified in the `_hw.tcl` file with the DERIVED property. Derived parameter values are calculated from other parameters during the Elaboration phase, and are specified in the `_hw.tcl` file with the DERIVED property. For example, you can derive a clock period parameter from a data rate parameter. Derived parameters are sometimes used to perform operations that are difficult to perform in HDL, such as using logarithmic functions to determine the number of address bits that a component requires.

Related Information

Declare Parameters with Custom _hw.tcl Commands on page 177

3.3.10.3. Obtaining Device Trait Information Using PART_TRAIT System Information Parameter

Within Platform Designer, an IP core can obtain information on the particular traits of a device using the PART_TRAIT system info parameter. This system info parameter takes an argument corresponding to the desired part trait. The requested trait must match the trait name as specified in the device database.

Note: Using this API declares your IP core as dependent on the requested trait.

To get the part number setting of Platform Designer system, use the value DEVICE, with the SYSTEM_INFO_ARG parameter property:

```
add_parameter part_trait_device string ""
set_parameter_property part_trait_device SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_device SYSTEM_INFO_ARG DEVICE
```

To get the base device of the part number setting of Platform Designer system, use the value BASE_DEVICE, with the SYSTEM_INFO_ARG parameter property:

```
add_parameter part_trait_bd string ""
set_parameter_property part_trait_bd SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_bd SYSTEM_INFO_ARG BASE_DEVICE
```

To get the device speed-grade of the part number setting of Platform Designer system, use the value DEVICE_SPEEDGRADE, with the SYSTEM_INFO_ARG parameter property:

```
add_parameter part_trait_sg string ""
set_parameter_property part_trait_sg SYSTEM_INFO_TYPE PART_TRAIT
set_parameter_property part_trait_sg SYSTEM_INFO_ARG DEVICE_SPEEDGRADE
```

3.3.10.4. Declare Parameters with Custom _hw.tcl Commands

The example below illustrates a custom _hw.tcl file, with more advanced parameter commands than those generated when you specify parameters in the Component Editor. Commands include the ALLOWED_RANGES property to provide a range of values for the AXI_ADDRESS_W (**Address Width**) parameter, and a list of parameter values for the AXI_DATA_W (**Data Width**) parameter. This example also shows the parameter AXI_NUMBYTES (**Data width in bytes**) parameter; that uses the DERIVED property. In addition, these commands illustrate the use of the GROUP property, which groups some parameters under a heading in the parameter editor GUI. You use the ENABLE_STREAM_OUTPUT_GROUP (**Include Avalon streaming source port**) parameter to enable or disable the optional Avalon Streaming interface in this design, and is displayed as a check box in the parameter editor GUI because the parameter is of type BOOLEAN. Refer to figure below to see the parameter editor GUI resulting from these _hw.tcl commands.

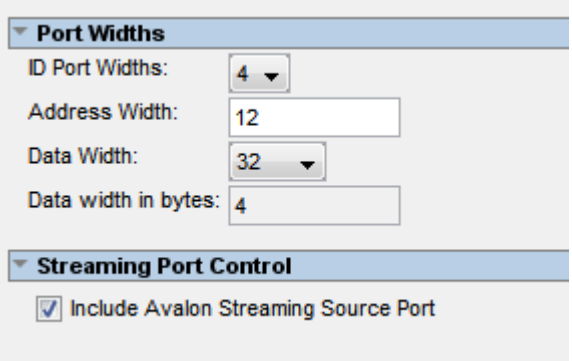
Example 13. Parameter Declaration

In this example, the AXI_NUMBYTES parameter is derived during the Elaboration phase based on another parameter, instead of being assigned to a specific value. AXI_NUMBYTES describes the number of bytes in a word of data. Platform Designer calculates the AXI_NUMBYTES parameter from the DATA_WIDTH parameter by dividing by 8. The _hw.tcl code defines the AXI_NUMBYTES parameter as a derived

parameter, since its value is calculated in an elaboration callback procedure. The AXI_NUMBYTES parameter value is not editable, because its value is based on another parameter value.

```
add_parameter AXI_ADDRESS_W INTEGER 12

set_parameter_property AXI_ADDRESS_W DISPLAY_NAME \
"AXI Subordinate Address Width"

set_parameter_property AXI_ADDRESS_W DESCRIPTION \
"Address width."

set_parameter_property AXI_ADDRESS_W UNITS bits
set_parameter_property AXI_ADDRESS_W ALLOWED_RANGES 4:16
set_parameter_property AXI_ADDRESS_W HDL_PARAMETER true

set_parameter_property AXI_ADDRESS_W GROUP \
"AXI Port Widths"

add_parameter AXI_DATA_W INTEGER 32
set_parameter_property AXI_DATA_W DISPLAY_NAME "Data Width"

set_parameter_property AXI_DATA_W DESCRIPTION \
"Width of data buses."

set_parameter_property AXI_DATA_W UNITS bits

set_parameter_property AXI_DATA_W ALLOWED_RANGES \
{8 16 32 64 128 256 512 1024}

set_parameter_property AXI_DATA_W HDL_PARAMETER true
set_parameter_property AXI_DATA_W GROUP "AXI Port Widths"

add_parameter AXI_NUMBYTES INTEGER 4
set_parameter_property AXI_NUMBYTES DERIVED true

set_parameter_property AXI_NUMBYTES DISPLAY_NAME \
"Data Width in bytes; Data Width/8"

set_parameter_property AXI_NUMBYTES DESCRIPTION \
"Number of bytes in one word"

set_parameter_property AXI_NUMBYTES UNITS bytes
set_parameter_property AXI_NUMBYTES HDL_PARAMETER true
set_parameter_property AXI_NUMBYTES GROUP "AXI Port Widths"

add_parameter ENABLE_STREAM_OUTPUT BOOLEAN true

set_parameter_property ENABLE_STREAM_OUTPUT DISPLAY_NAME \
"Include Avalon Streaming Source Port"

set_parameter_property ENABLE_STREAM_OUTPUT DESCRIPTION \
"Include optional Avalon streaming source (default),\
or hide the interface"

set_parameter_property ENABLE_STREAM_OUTPUT GROUP \
"Streaming Port Control"

...
```

Figure 141. Resulting Parameter Editor GUI from Parameter Declarations



Related Information

- Control Interfaces Dynamically with an Elaboration Callback on page 194
- Component Interface Tcl Reference on page 666

3.3.10.5. Validate Parameter Values with a Validation Callback

You can use a validation callback procedure to validate parameter values with more complex validation operations than the ALLOWED_RANGES property allows. You define a validation callback by setting the VALIDATION_CALLBACK module property to the name of the Tcl callback procedure that runs during the validation phase. In the validation callback procedure, the current parameter values is queried, and warnings or errors are reported about the component's configuration.

Example 14. Demo AXI Memory Example

If the optional Avalon streaming interface is enabled, then the control registers must be wide enough to hold an AXI RAM address, so the designer can add an error message to ensure that the user enters allowable parameter values.

```
set_module_property VALIDATION_CALLBACK validate
proc validate {} {
if {
  [get_parameter_value ENABLE_STREAM_OUTPUT ] &&
  ([get_parameter_value AXI_ADDRESS_W] >
  [get_parameter_value AV_DATA_W])
}
send_message error "If the optional Avalon streaming port\
is enabled, the AXI Data Width must be equal to or greater\
than the Avalon control port Address Width"
}
}
```

Related Information

Component Interface Tcl Reference on page 666

3.4. Creating Generic Components in a System

Platform Designer allows you to add generic components with the implementation defined in `_hw.tcl` (**IP** type), in an HDL file (**HDL** type), or with only a partially defined implementation (**Blackbox** type). The generic component enables hierarchical isolation of the IP components by separating the component instantiation from the component implementation. This generic component is available as **Generic Component** in the Platform Designer IP Catalog.

When you generate a system containing a generic component, the system's RTL instantiates the component, but does not provide the implementation for the component. Rather, you must provide the implementation for the component in a downstream compiler such as Quartus Prime software or in RTL code.

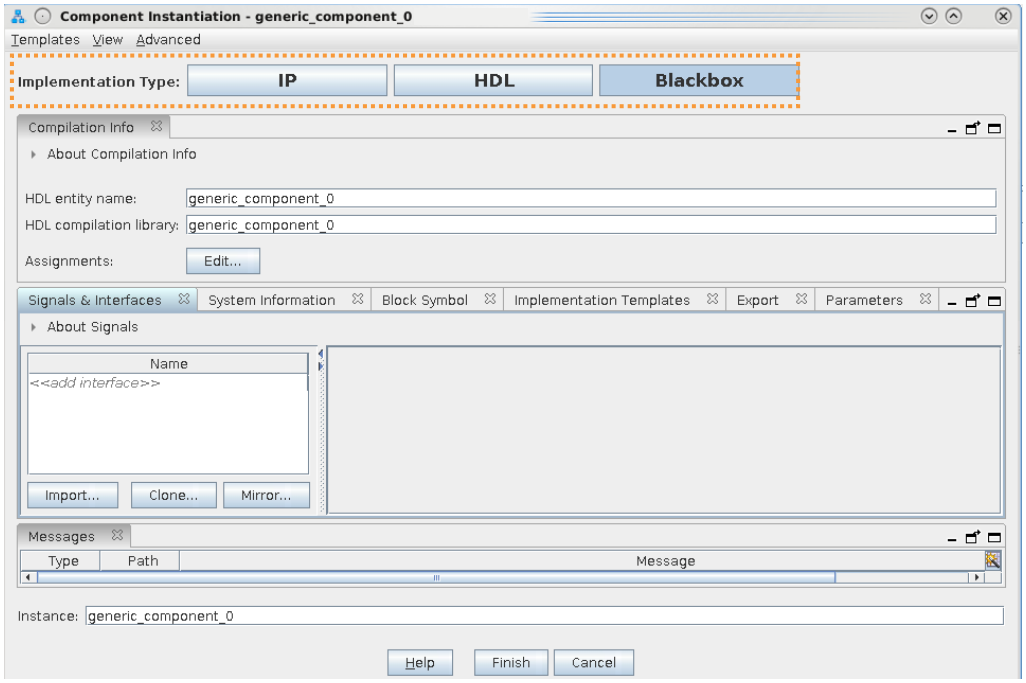
The following generic component **Implementation Type**s are available in the **Component Instantiation** editor, depending on your use case:

Table 41. Component Implementation Type Options

Implementation Type	Description
IP	The default implementation type that defines the component in `_hw.tcl` and preserves the component as a `.ip` file. Platform Designer automatically manages components with the **Implementation Type** of **IP** in the following ways: • Runs background checks against the port widths between the IP component and the `.ip` file to ensure continuity. • Scans the `.ip` file for the error flag to determine if any component has parameterization errors. • Checks for system-info mismatches between the IP file and the IP component in the system, and prompts resolution with IP instantiation warnings in the **Instantiation Messages** tab.
HDL	Defines a generic component from existing RTL. You can load the signals, interfaces, and parameters of the generic component from the HDL file containing the RTL. The HDL parameters are represented as constants local to a module, which you can redefine when instantiating the module. Generic HDL components have no `.ip` file.
Blackbox	Defines a generic component that represents only the signal and interface boundary of an entity, without providing the component's implementation. You then provide the implementation of the component for processing with the Quartus Prime software or an RTL simulator. Generic blackbox components have no `.ip` file.

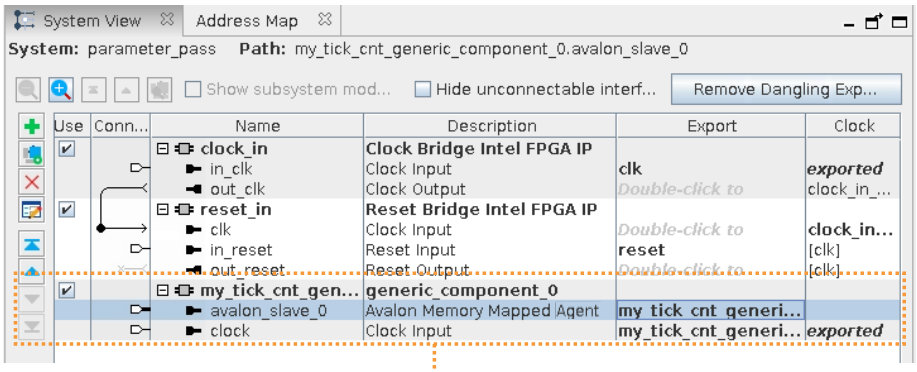
1. From the Platform Designer IP Catalog, double-click **Generic Component**. The **Component Instantiation** editor appears.
2. For **Implementation Type**, click **IP**, **HDL**, **Blackbox**, or **HLS** for your generic component. Refer to Component Implementation Type Options.
3. Add parameters to the generic component, as applicable:
   - To add parameters to a generic **HDL** component, click the **Add File** button under **Implementation Files** to specify the RTL that defines your generic component, as Adding Generic HDL Component Parameters describes.
   - To add parameters to a generic **Blackbox** component, click the **Parameters** tab, and then **Add Parameter** button to define the parameters for your generic component, as Adding Generic Blackbox Component Parameters describes.

Figure 142. Component Instantiation Editor



4. Add interfaces and signals to the generic component, as Adding Generic Component Interfaces and Signals describes.
5. Click **Finish**, the generic component appears in the **System View**.
6. In the **System View**, select the generic component that you create in step 5.

Figure 143. New Generic Component in System View



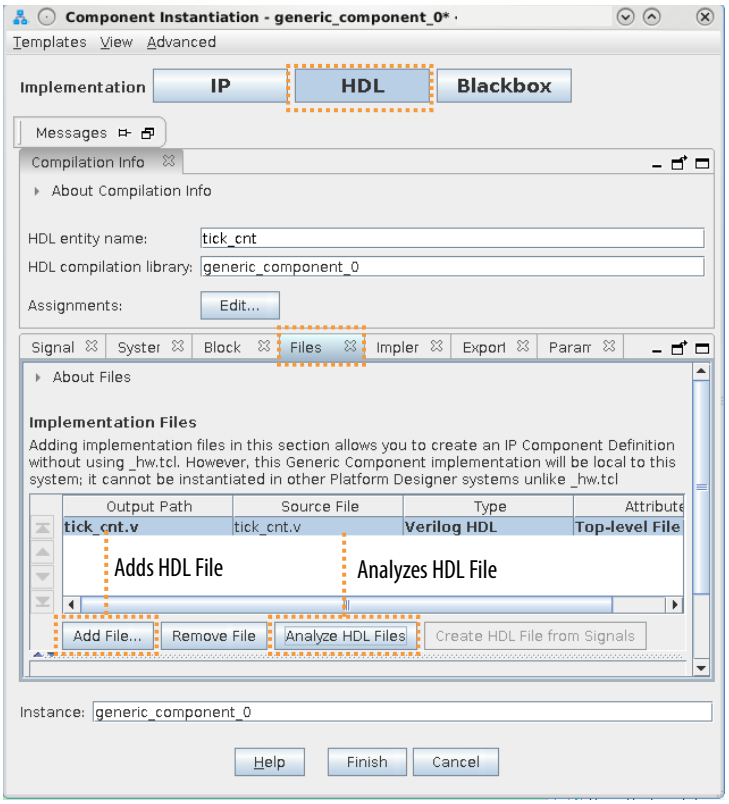
New Generic Component in System View

7. In the **System View** tab, double-click the **Export** column to export the signal(s) when you generate the system HDL.
8. In the **Parameters** tab for the component, modify the default parameter values as needed.
9. In Platform Designer, click the **Generate HDL** button. Platform Designer generates the HDL for the system and generic component according to your specifications.

3.4.1. Adding Generic HDL Component Parameters

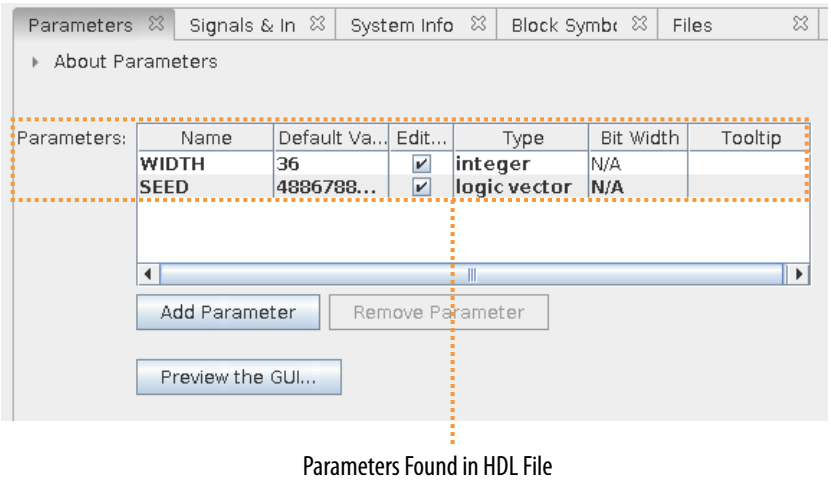
You can specify and modify HDL parameters for generic components that have an **HDL** implementation. This technique allows you to reuse the HDL component in another context with different parameter values. Platform Designer analyzes the HDL implementation, identifies the HDL parameters present, and loads the HDL parameters into the **Component Instantiation** editor where you can modify them. To add parameters in **HDL** mode, follow these steps:

Figure 144. HDL Implementation Type in Component Instantiation Editor



1. Add a generic HDL component, as Creating Generic Components in a System on page 180 describes.
2. In the **Component Instantiation** editor **Files** tab, click the **Add File** button to select the HDL file that contains the component definition.
3. Under **Implementation Files**, select the HDL file, and then click **Analyze HDL files**. The signals and interfaces found in the HDL component appear in the **Signals & Interfaces** tab. The parameters found in the HDL component appear in the **Parameters** tab.
4. On the **Parameters** tab, modify the parameter **Name**, **Value**, **Type**, **Bit Width**, and whether **Editable**.

Figure 145. Parameters Tab for HDL Instantiation



5. View the interfaces and signals, as Adding Generic Component Interfaces and Signals on page 184 describes.
6. When you are done adding parameters and interfaces, click **Finish** in the **Component Instantiation** editor. The component appears in the **System View** tab.

Note: You must treat a generic component with an **Implementation Type** of **HDL** as custom RTL, specific to your current system. When you set a generic component's **Implementation Type** to **HDL**, the output of any RTL that you add to the component is within the system's output directory.

Related Information

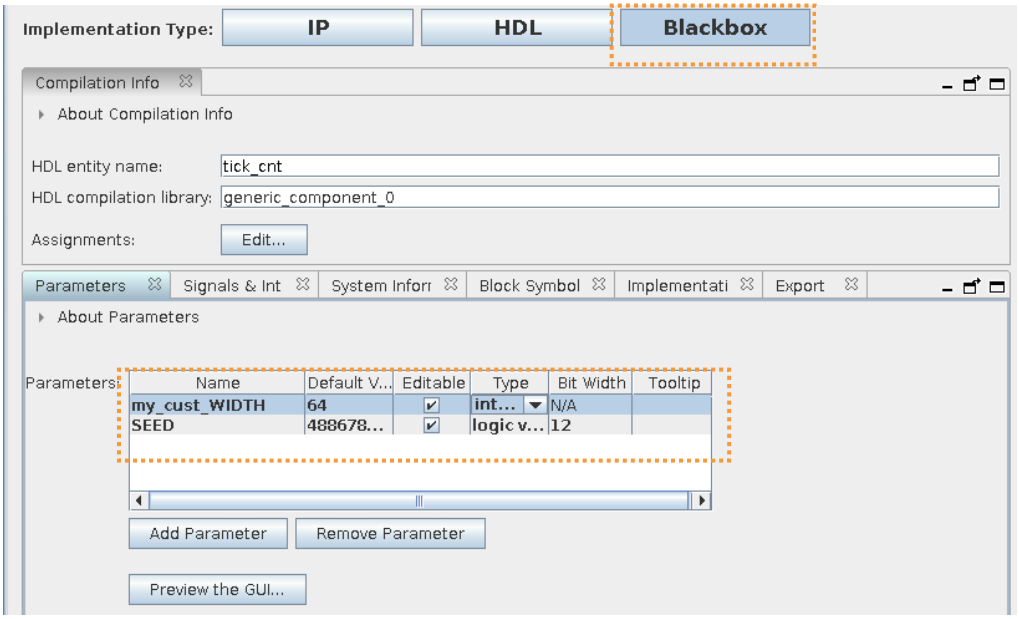
Exporting HDL Parameters to a System on page 189

3.4.2. Adding Generic Blackbox Component Parameters

You can specify and modify parameters for generic components that have a **Blackbox** implementation. This technique allows you to instantiate a component that defines only the signal and interface boundary of an entity, without providing the component's implementation at this stage. You can then add parameters and define their default properties in the **Component Instantiation** editor.

1. Define a generic blackbox component, as Creating Generic Components in a System on page 180 describes.
2. In the **Component Instantiation** editor, click the **Parameters** tab.

Figure 146. Parameters Tab in Component Instantiation Editor (Blackbox Mode)



3. Click the **Add Parameter** button to add and specify default values for the parameter **Name**, **Value**, **Type**, **Bit Width**, and whether **Editable**.
4. Add signals and interfaces on the **Signals and Interfaces** tab, as Adding Generic Component Interfaces and Signals on page 184 describes.
5. When you are done adding parameters and interfaces, click **Finish** in the **Component Instantiation** editor. The component appears in the **System View** tab.

Related Information

Exporting HDL Parameters to a System on page 189

3.4.3. Adding Generic Component Interfaces and Signals

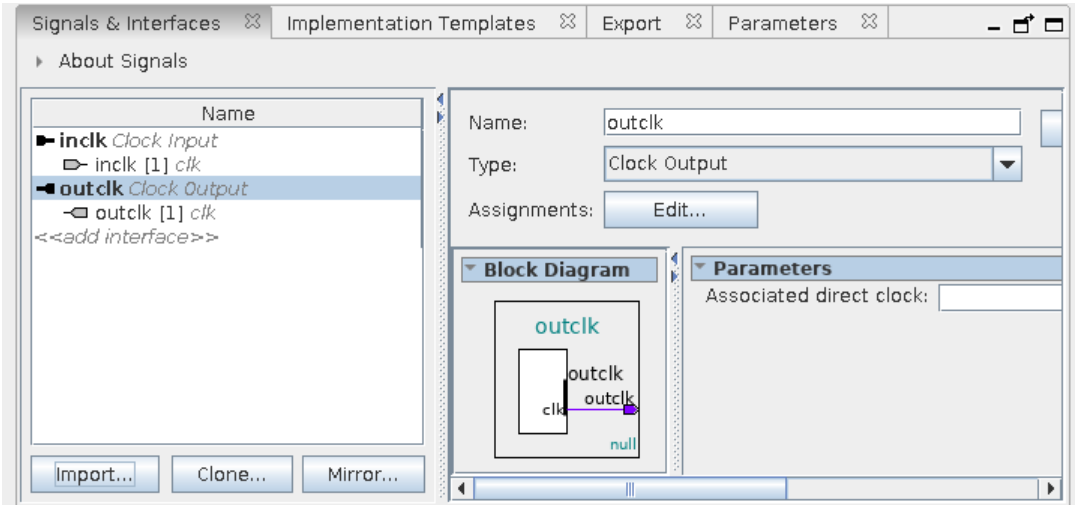
The **Signals & Interfaces** tab of the **Component Instantiation** editor allows you to customize signals and interfaces for your generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab, as Adding Generic Component Interfaces and Signals on page 184 describes.
3. To add an interface, click <<*add interface*>> in the left pane and select the interface. The selected interface appears in the parameter editor to the right, where you specify its parameters.
4. To add signals to the selected interface, click <<*add signal*>> below the selected interface.

5. To move signals between interfaces, select the signal and drag it to another interface.
6. To rename a signal or interface, select the element, and then press F2.
7. To remove a signal or interface, right-click the element, and then click **Remove**.

   Note: Alternatively, to remove a signal or interface, select the element and press **Delete**. When you remove an interface, Platform Designer also removes all of its associated signals.

Figure 147. Creating Custom Interfaces



Note: To add existing template interfaces to your generic component, select the interface from the **Templates** menu in the **Component Instantiation** editor.

3.4.3.1. Mirroring Interfaces in a Generic Component

To mirror existing signals and interfaces from an IP component to your generic component:

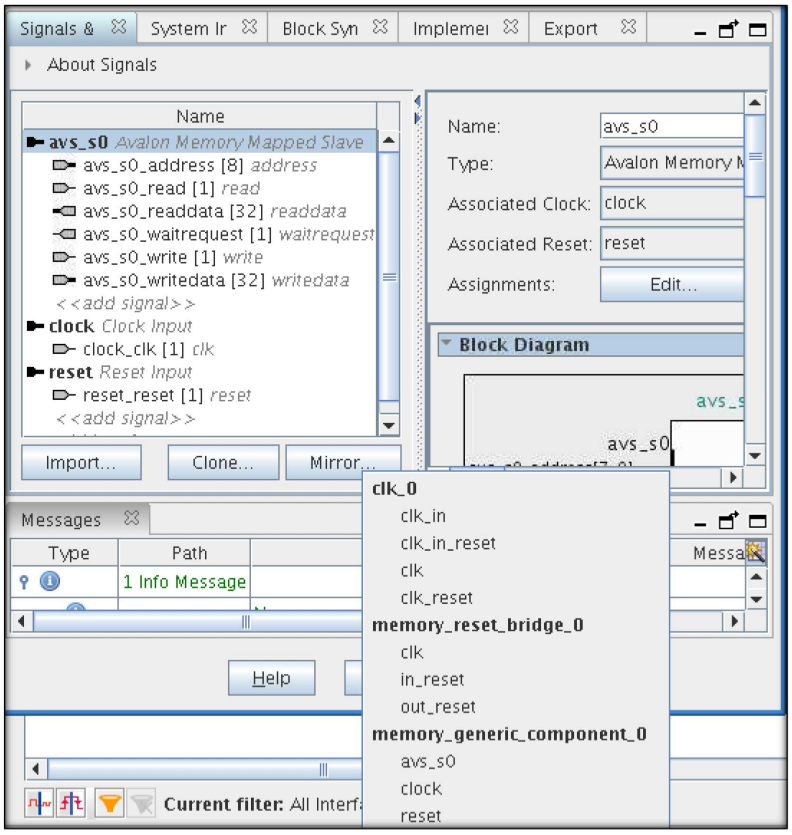
1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Mirror** button. A list appears which lists all the available components in the system and their associated interfaces.
4. Select the desired interface. Platform Designer mirrors the interface and its associated signals and adds the mirrored interfaces and signals to the **Signals & Interfaces** tab of the generic component.

Example 15. Mirroring Interfaces in a Generic Component Example

Selected Interface	Mirrored Interface
Avalon Memory-Mapped Host (`avalon_host`)	Avalon Memory-Mapped Agent (`avalon_agent`)

Signals of the Selected Interface	Signals of the Mirrored Interface
waitrequest(Input 1)	waitrequest(Output 1)
readdata(Input 32)	readdata(Output 32)
readdatavalid(Input 1)	readdatavalid(Output 1)
burstcount(Output 32)	burstcount(Input 32)

Figure 148. Mirroring Interfaces

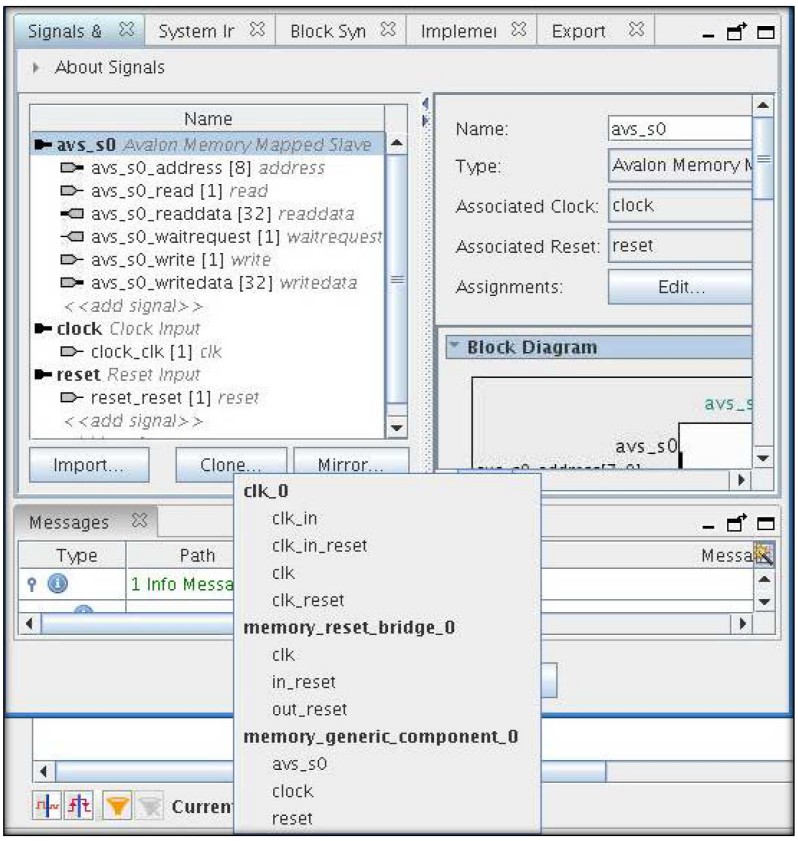


3.4.3.2. Cloning Interfaces in a Generic Component

To clone existing signals and interfaces from an IP component to your generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Clone** button. A list appears which lists all the available components in the system and their associated interfaces.
4. Select the desired interface. Platform Designer clones the interface and adds an exact replica of the interface and its associated signals to the **Signals & Interfaces** tab of the generic component.

Figure 149. Cloning Interfaces



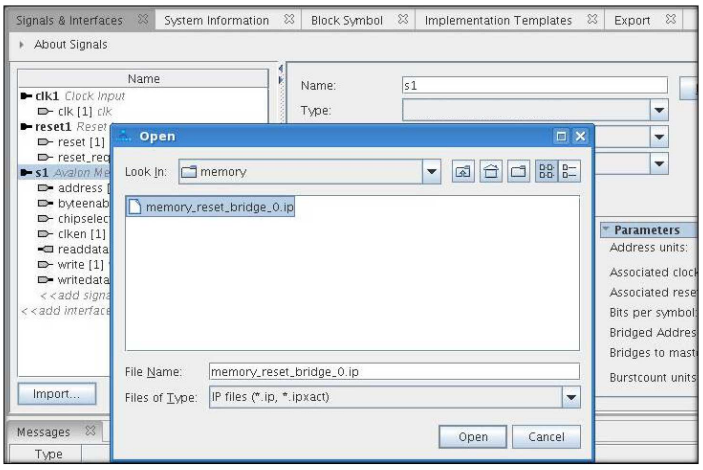
3.4.3.3. Importing Interfaces to a Generic Component

To import interfaces from an existing IP or IP-XACT[8] file to a generic component:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, click the **Signals & Interfaces** tab.
3. Click the **Import** button..
   A dialog box appears from where you choose the IP/IP-XACT file to import to the generic component
4. Select the interface.
   Platform Designer populates the **Signals & Interfaces** tab with the signals and interfaces defined in the selected file.

(8) Platform Designer supports importing and exporting files in IP-XACT 2009 format and exporting IP-XACT files in 2014 format.

Figure 150. Importing Interfaces

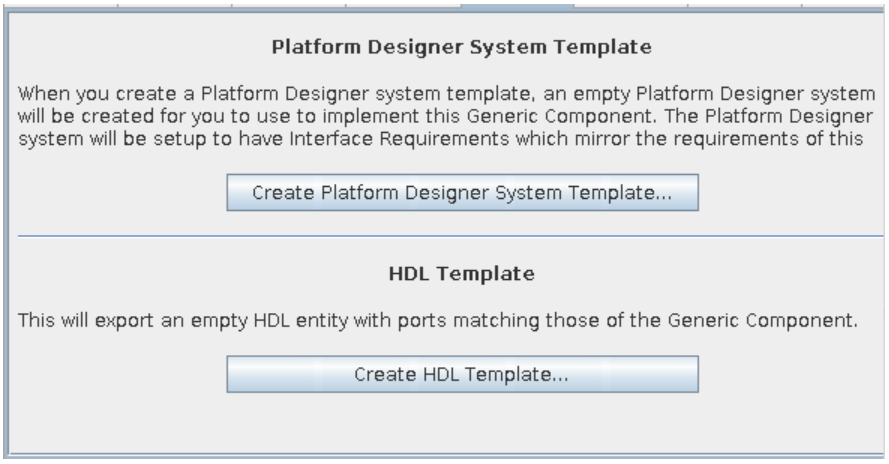


3.4.4. Creating a System Template for a Generic Component

To create a Platform Designer system template:

1. Double-click **Generic Component** in the IP Catalog.
2. In the **Component Instantiation** editor, add the interfaces and signals for the new component in the **Signals & Interfaces** tab, as .
3. Select the **Implementation Templates** tab.
4. Click **Create Platform Designer System Template** button. This option creates an empty Platform Designer system and saves the template as a `.qsys` file to implement this generic component.

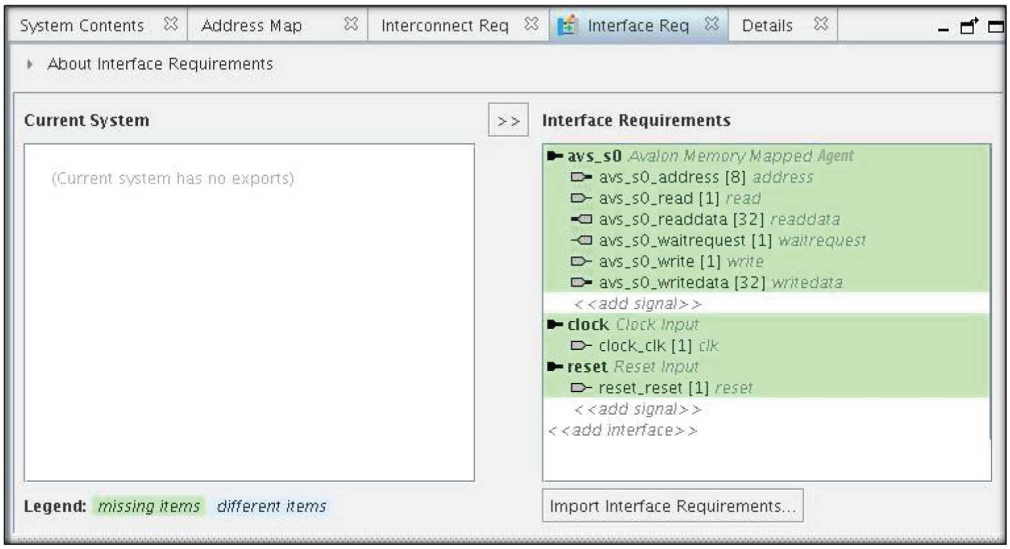
Figure 151. Creating System templates



To implement this component:

1. To open the template Platform Designer system, click **File ➤ Open** and choose the specific `.qsys` file.
2. Add either or both IP components and generic components then export their interfaces to satisfy the specified interface requirements.
3. To view the exported interfaces in the **Interface Requirements** tab, select **View ➤ Interface Requirements**.

Figure 152. Viewing the Interface Requirements from the System Template



3.4.5. Exporting a Generic Component

You can export a generic component as a `.ipxact` file as well as `_hw.tcl` file:

1. Double-click **Generic Component** in the IP Catalog.
2. Select the **Export** tab.
3. To export generic component as an IP-XACT file, click **Export IP-XACT File** and select the location to save your IP-XACT file.
4. To export generic component as a `_hw.tcl` file, click **Export _hw.tcl File** and select the location to save your `_hw.tcl` file.

3.5. Exporting HDL Parameters to a System

You can define and export Platform Designer component HDL parameters to make the parameters accessible to a higher-level system. After adding system HDL parameters, you can map the HDL parameters to components instantiated within the system.

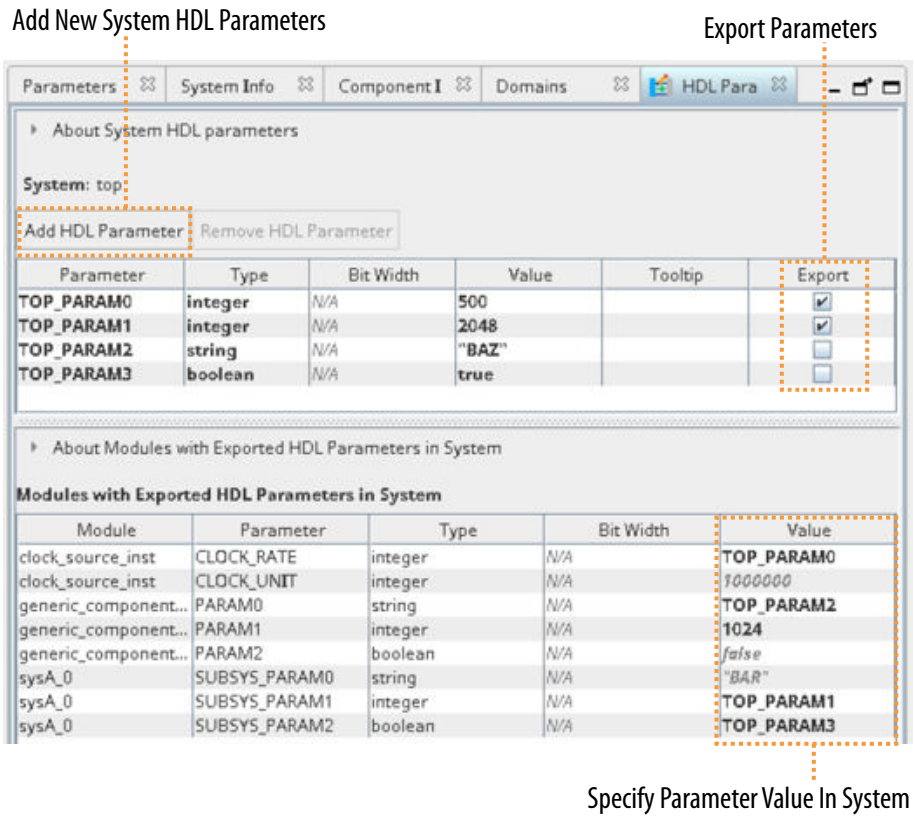
This technique allows you to set a component's parameter values outside of the immediate subsystem that contains the component, and propagate the HDL parameters upward. When you generate the system HDL, the HDL includes the parameters and mapping that you specify.

Note: Parameterization of ports is not supported.

To export an instantiated component's HDL parameters to a system, follow these steps:

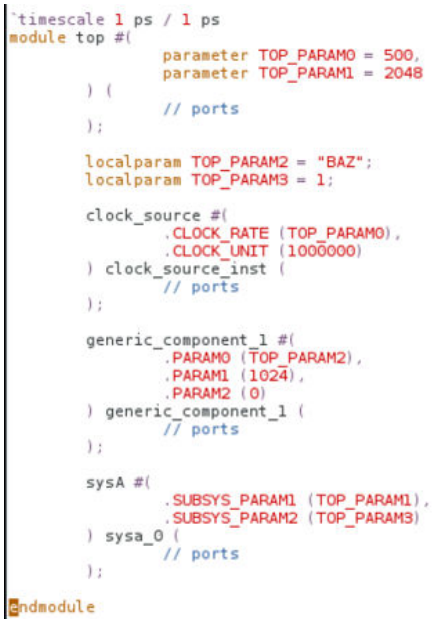
1. Instantiate a component in a Platform Designer system, as any of the following describe:
   - Creating IP Components in the Component Editor on page 158
   - Adding Generic HDL Component Parameters on page 182
   - Adding Generic Blackbox Component Parameters on page 183
2. In Platform Designer, click **View ➤ HDL Parameters**. The **HDL Parameters** tab allows you to define system HDL parameters, and overwrite instantiated components' HDL parameter values in the current system.
3. To add a new HDL parameter to the current system, click the **Add HDL Parameter** button and specify values for the parameter name, **Type**, **Width**, **Value**, and **Tooltip** options that apply to the HDL parameter. Refer to HDL Parameters Tab Settings and Controls on page 192.
4. To export the parameter during system HDL generation, thereby exposing the HDL parameter to a higher-level system, enable the **Export** option.

Figure 153. Specifying HDL Parameters



5. To change the parameter value for **Modules with Exported HDL Parameters**, select the **Value** that you want to implement during HDL generation. You can specify any of the added parameters as the value, as Figure 153 on page 191 shows with any of the TOP_PARAMx parameters.
6. To generate the HDL that includes exported parameters, click the **Generate HDL** button in Platform Designer.

Figure 154. Generated HDL for Parameters Specified in Figure 81



```
`timescale 1 ps / 1 ps
module top #(
		parameter TOP_PARAM0 = 500,
		parameter TOP_PARAM1 = 2048
	) (
		// ports
	);

	localparam TOP_PARAM2 = "BAZ";
	localparam TOP_PARAM3 = 1;

	clock_source #(
		.CLOCK_RATE (TOP_PARAM0),
		.CLOCK_UNIT (1000000)
	) clock_source_inst (
		// ports
	);

	generic_component_1 #(
		.PARAM0 (TOP_PARAM2),
		.PARAM1 (1024),
		.PARAM2 (0)
	) generic_component_1 (
		// ports
	);

	sysA #(
		.SUBSYS_PARAM1 (TOP_PARAM1),
		.SUBSYS_PARAM2 (TOP_PARAM3)
	) sysa_0 (
		// ports
	);

endmodule
```

When you generate the HDL, Platform Designer instantiates the IP or generic components with the inline parameters set to the system HDL parameter values, or set to constant values if applicable:

- **For IP components with parameters defined in _hw.tcl and subsystems**—the generated system HDL only writes out inline parameters for those HDL parameters that you overwrite for the system in the **HDL Parameters** tab (or with the `set_exposed_hdl_parameter_value` scripting command).
- **For generic HDL or Blackbox components**—the generated system HDL writes out all HDL parameters in the instantiation. The generated system HDL prioritizes setting HDL parameter values you specify in the **HDL Parameters** tab (or with the `set_exposed_hdl_parameter_value` scripting command). If the component's HDL parameter value is not overwritten in the **HDL Parameters** tab or with `set_exposed_hdl_parameter_value`, the HDL parameter's default value applies.

Note: If instantiating your component in `_hw.tcl`, rather than using the GUI, you must set `HDL_PARAMETER` to `true` and `AFFECTS_GENERATION` to `false`.

Related Information

- Defining HDL Parameters in _hw.tcl on page 163
- Adding Generic HDL Component Parameters on page 182
- Adding Generic Blackbox Component Parameters on page 183

3.5.1. HDL Parameters Tab Settings and Controls

The Platform Designer **HDL Parameters** tab allows you to define system HDL parameters, and overwrite instantiated components' HDL parameter values in the current system.

The following settings and controls are available in the **HDL Parameters** tab:

Table 42. Component Implementation Type Options

Setting/Control	Description
System HDL Parameters	
Add HDL Parameter	Inserts a new HDL parameter with editable name and default settings. Specify the parameter name and default value for applicable parameter properties.
Remove HDL Parameter	Removes the selected system HDL parameter from the **HDL Parameters** tab.
Parameter	Specifies the name of a system HDL parameter. By default, Platform Designer names new HDL parameters sequentially with `new_parameter_0`, `new_parameter_1`, and so on. Double-click any parameter name to rename.
Type	Specifies the type of HDL parameter, which determines availability of other properties. The following **Types** are available: **boolean**, **integer**, **logic vector**, **natural**, **positive**, **std logic**, and **string**.
Width	Specifies the width of signals for the **logic vector** HDL parameter **Type**. The value of this property is N/A for other **Type**s.
Value	The default value of the HDL parameter.
Tooltip	Alphanumeric text that provides brief guidance about the HDL parameter to the end user in the form of a tooltip.
Export	Specifies that the HDL parameter should be exported, allowing it to be overwritten in a higher level system.
Modules with Exported HDL Parameters in System	
Value	Specifies the value of exported HDL parameters in the current Platform Designer system. The value that you specify overwrites the current parameter value when you **Generate HDL** for the system. Select an added HDL parameter with the **Export** option enabled to allow an upper-level system to overwrite the value.

3.6. Scripting Wire-Level Expressions

Platform Designer supports system scripting commands to apply wire-level expressions to input ports in IP components.

The following commands function with the `qsys-script` utility or in a `_hw.tcl` file to set, retrieve, or remove an expression on a port:

```
set_wirelevel_expression <instance_or_port_bit> <expression>
get_wirelevel_expressions <instance_or_port_bit>
remove_wirelevel_expressions <instance_or_port_bit
```

These commands require a string that you compose from the left-handed and right-handed components of the expression. Platform Designer reports errors in syntax, existence, or system hierarchy.

Related Information

- Wire-Level Connection Commands on page 633
- set_wirelevel_expression on page 633
- get_wirelevel_expressions on page 634
- remove_wirelevel_expressions on page 635

3.7. Control Interfaces Dynamically with an Elaboration Callback

You can allow user parameters to dynamically control your component's behavior with an elaboration callback procedure during the elaboration phase. Using an elaboration callback allows you to change interface properties, remove interfaces, or add new interfaces as a function of a parameter value. You define an elaboration callback by setting the module property ELABORATION_CALLBACK to the name of the Tcl callback procedure that runs during the elaboration phase. In the callback procedure, you can query the parameter values of the component instance, and then change the interfaces accordingly.

Example 16. Avalon Streaming Source Interface Optionally Included in a Component Specified with an Elaboration Callback

```
set_module_property ELABORATION_CALLBACK elaborate

proc elaborate {} {

   # Optionally disable the Avalon streaming data output

    if{[ get_parameter_value ENABLE_STREAM_OUTPUT] == "false" }{
       set_port_property aso_data    termination true
       set_port_property aso_valid   termination true
       set_port_property aso_ready   termination true
       set_port_property aso_ready   termination_value 0
    }
  # Calculate the Data Bus Width in bytes

    set bytewidth_var [expr [get_parameter_value AXI_DATA_W]/8]
    set_parameter_value AXI_NUMBYTES $bytewidth_var
}
```

Related Information

- Declare Parameters with Custom _hw.tcl Commands on page 177
- Validate Parameter Values with a Validation Callback on page 179
- Component Interface Tcl Reference on page 666

3.8. Control File Generation Dynamically with Parameters and a Fileset Callback

You can use a fileset callback to control which files are created in the output directories during the generation phase based on parameter values, instead of providing a fixed list of files. In a callback procedure, you can query the values of the parameters and use them to generate the appropriate files. To define a fileset callback, you specify a callback procedure name as an argument in the `add_fileset` command. You can use the same fileset callback procedure for all of the filesets, or create separate procedures for synthesis and simulation, or Verilog and VHDL.

Example 17. Fileset Callback Using Parameters to Control Filesets in Two Different Ways

The RAM_VERSION parameter chooses between two different source files to control the implementation of a RAM block. For the top-level source file, a custom Tcl routine generates HDL that optionally includes control and status registers, depending on the value of the CSR_ENABLED parameter.

During the generation phase, Platform Designer creates a top-level Platform Designer system HDL wrapper module to instantiate the component top-level module, and applies the component's parameters, for any parameter whose parameter property HDL_PARAMETER is set to true.

```
#Create synthesis fileset with fileset_callback and set top level

add_fileset my_synthesis_fileset QUARTUS_SYNTH fileset_callback

set_fileset_property my_synthesis_fileset TOP_LEVEL \
demo_axi_memory

# Create Verilog simulation fileset with same fileset_callback
# and set top level

add_fileset my_verilog_sim_fileset SIM_VERILOG fileset_callback

set_fileset_property my_verilog_sim_fileset TOP_LEVEL \
demo_axi_memory

# Add extra file needed for simulation only

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH \
verification_lib/verbosity_pkg.sv

# Create VHDL simulation fileset (with Verilog files
# for mixed-language VHDL simulation)

add_fileset my_vhdl_sim_fileset SIM_VHDL fileset_callback
set_fileset_property my_vhdl_sim_fileset TOP_LEVEL demo_axi_memory

add_fileset_file verbosity_pkg.sv SYSTEM_VERILOG PATH
verification_lib/verbosity_pkg.sv

# Define parameters required for fileset_callback

add_parameter RAM_VERSION INTEGER 1
set_parameter_property RAM_VERSION ALLOWED_RANGES {1 2}
set_parameter_property RAM_VERSION HDL_PARAMETER false
add_parameter CSR_ENABLED BOOLEAN enable
set_parameter_property CSR_ENABLED HDL_PARAMETER false

# Create Tcl callback procedure to add appropriate files to
# filesets based on parameters

proc fileset_callback { entityName } {
  send_message INFO "Generating top-level entity $entityName"
  set ram [get_parameter_value RAM_VERSION]
  set csr_enabled [get_parameter_value CSR_ENABLED]

  send_message INFO "Generating memory
  implementation based on RAM_VERSION $ram    "

      if {$ram == 1} {
          add_fileset_file single_clk_ram1.v VERILOG PATH \
    single_clk_ram1.v
      } else     {
          add_fileset_file single_clk_ram2.v VERILOG PATH \
    single_clk_ram2.v
    }

send_message INFO "Generating top-level file for \
CSR_ENABLED $csr_enabled"

generate_my_custom_hdl $csr_enabled demo_axi_memory_gen.sv

add_fileset_file demo_axi_memory_gen.sv VERILOG PATH \
demo_axi_memory_gen.sv
}
```

Related Information

- Specify Synthesis and Simulation Files in the Platform Designer Component Editor on page 167
- Component Interface Tcl Reference on page 666

3.9. Create a Composed Component or Subsystem

A composed component is a subsystem containing instances of other components. Unlike an HDL-based component, a composed component's HDL is created by generating HDL for the components in the subsystem, in addition to the Platform Designer interconnect to connect the subsystem instances.

You can add child instances in a composition callback of the `_hw.tcl` file.

With a composition callback, you can also instantiate and parameterize sub-components as a function of the composed component's parameter values. You define a composition callback by setting the `COMPOSITION_CALLBACK` module property to the name of the composition callback procedures.

A composition callback replaces the validation and elaboration phases. HDL for the subsystem is generated by generating all of the sub-components and the top-level that combines them.

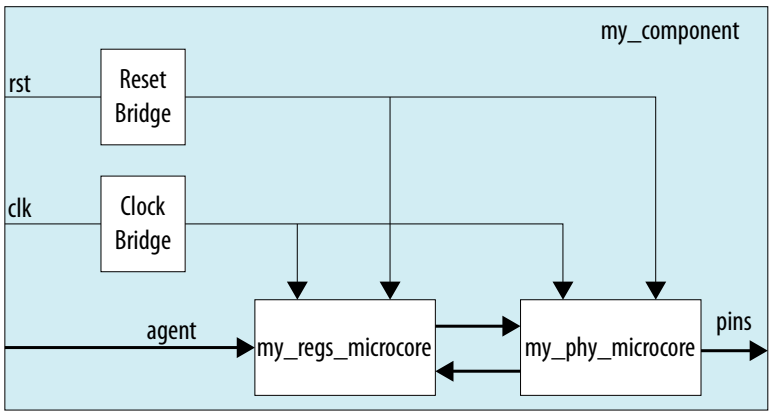
To connect instances of your component, you must define the component's interfaces. Unlike an HDL-based component, a composed component does not directly specify the signals that are exported. Instead, interfaces of submodules are chosen as the external interface, and each internal interface's ports are connected through the exported interface.

Exporting an interface means that you are making the interface visible from the outside of your component, instead of connecting it internally. You can set the `EXPORT_OF` property of the externally visible interface from the main program or the composition callback, to indicate that it is an exported view of the submodule's interface.

Exporting an interface is different than defining an interface. An exported interface is an exact copy of the subcomponent's interface, and you are not allowed to change properties on the exported interface. For example, if the internal interface is a 32-bit or 64-bit host without bursting, then the exported interface is the same. An interface on a subcomponent cannot be exported and also connected within the subsystem.

When you create an exported interface, the properties of the exported interface are copied from the subcomponent's interface without modification. Ports are copied from the subcomponent's interface with only one modification; the names of the exported ports on the composed component are chosen to ensure that they are unique.

Figure 155. Top-Level of a Composed Component



Example 18. Composed _hw.tcl File that Instantiates Two Sub-Components

Platform Designer connects the components, and also connects the clocks and resets. Note that clock and reset bridge components are required to allow both sub-components to see common clock and reset inputs.

```
package require -exact qsys 14.0
set_module_property name my_component
set_module_property COMPOSITION_CALLBACK composed_component

proc composed_component {} {
  add_instance clk altera_clock_bridge
  add_instance reset altera_reset_bridge
  add_instance regs my_regs_microcore
  add_instance phy my_phy_microcore

  add_interface clk clock end
  add_interface reset reset end
  add_interface agent avalon agent
  add_interface pins conduit end

  set_interface_property clk EXPORT_OF clk.in_clk
  set_instance_parameter_value reset synchronous_edges deassert
  set_interface_property reset EXPORT_OF reset.in_reset
  set_interface_property subordinate EXPORT_OF regs.subordinate
  set_interface_property pins  EXPORT_OF phy.pins

  add_connection clk.out_clk reset.clk
  add_connection clk.out_clk regs.clk
  add_connection clk.out_clk phy.clk
  add_connection reset.out_reset regs.reset
  add_connection reset.out_reset phy.clk_reset
  add_connection regs.output phy.input
  add_connection phy.output regs.input
}
```

Related Information

Component Interface Tcl Reference on page 666

3.10. Add Component Instances to a Static or Generated Component

You can create nested components by adding component instances to an existing component. Both static and generated components can create instances of other components. You can add child instances of a component in a `_hw.tcl` using elaboration callback.

With an elaboration callback, you can also instantiate and parameterize sub-components with the `add_hdl_instance` command as a function of the parent component's parameter values.

When you instantiate multiple nested components, you must create a unique variation name for each component with the `add_hdl_instance` command. Prefixing a variation name with the parent component name prevents conflicts in a system. The variation name can be the same across multiple parent components if the generated parameterization of the nested component is exactly the same.

Note: If you do not adhere to the above naming variation guidelines, Platform Designer validation-time errors occur, which are often difficult to debug.

Related Information

- Static IP Components on page 198
- Generated Components on page 199

3.10.1. Static IP Components

Static IP components always generate the same output, regardless of their parameterization. Components that instantiate static components must have only static children.

A design file that is static between all parameterizations of a component can only instantiate other static design files. Since static IPs always render the same HDL regardless of parameterization, Platform Designer generates static IPs only once across multiple instantiations, meaning they have the same top-level name set.

Example 19. Typical Usage of the add_hdl_instance Command for Static Components

```
package require -exact qsys 14.0

set_module_property name add_hdl_instance_example
add_fileset synth_fileset QUARTUS_SYNTH synth_callback
set_fileset_property synth_fileset TOP_LEVEL basic_static
set_module_property elaboration_callback elab

proc elab {} {
  # Actual API to instantiate an IP Core
  add_hdl_instance emif_instance_name altera_mem_if_ddr3_emif

  # Make sure the parameters are set appropriately
  set_instance_parameter_value emif_instance_name SPEED_GRADE {7}
  ...
  }
proc synth_callback { output_name } {
  add_fileset_file "basic_static.v" VERILOG PATH basic_static.v
}
```

Example 20. Top-Level HDL Instance and Wrapper File Created by Platform Designer

In this example, Platform Designer generates a wrapper file for the instance name specified in the _hw.tcl file.

```
//Top Level Component HDL
module basic_static (input_wire, output_wire, inout_wire);
input [31:0] input_wire;
output [31:0] output_wire;
inout [31:0] inout_wire;

// Instantiation of the instance added via add_hdl_instance
// command. This is an example of how the instance added via
// the add_hdl_instance command can be used
// in the top-level file of the component.

emif_instance_name fixed_name_instantiation_in_top_level(
.pll_ref_clk (input_wire), // pll_ref_clk.clk
.global_reset_n (input_wire), // global_reset.reset_n
.soft_reset_n (input_wire), // soft_reset.reset_n
...
... );
endmodule

//Wrapper for added HDL instance
// emif_instance_name.v
// Generated using ACDS version 14.0

`timescale 1 ps / 1 ps
module emif_instance_name (
input wire pll_ref_clk, // pll_ref_clk.clk
input wire global_reset_n, // global_reset.reset_n
input wire soft_reset_n, // soft_reset.reset_n
output wire afi_clk, // afi_clk.clk
...
...);
example_addhdlinstance_system
_add_hdl_instance_example_0_emif_instance
_name_emif_instance_name emif_instance_name (

.pll_ref_clk (pll_ref_clk), // pll_ref_clk.clk
.global_reset_n (global_reset_n), // global_reset.reset_n
.soft_reset_n (soft_reset_n), // soft_reset.reset_n
...
...);
endmodule
```

3.10.2. Generated Components

A generated component's fileset callback allows an instance of the component to create unique HDL design files based on the instance's parameter values. For example, you can write a fileset callback to include a control and status interface based on the value of a parameter. The callback overcomes a limitation of HDL languages, which do not allow run-time parameters.

Generated components change their generation output (HDL) based on their parameterization. If a component is generated, then any component that may instantiate it with multiple parameter sets must also be considered generated, since its HDL changes with its parameterization. This case has an effect that propagates up to the top-level of a design.

Since generated components are generated for each unique parameterized instantiation, when implementing the `add_hdl_instance` command, you cannot use the same fixed name (specified using `instance_name`) for the different variants of

the child HDL instances. To facilitate unique naming for the wrapper of each unique parameterized instantiation of child HDL instances, you must use the following command so that Platform Designer generates a unique name for each wrapper. You can then access this unique wrapper name with a fileset callback so that the instances are instantiated inside the component's top-level HDL.

- To declare auto-generated fixed names for wrappers, use the command:

```
set_instance_property instance_name HDLINSTANCE_USE_GENERATED_NAME \
true
```

  Note: You can only use this command with a generated component in the global context, or in an elaboration callback.

- To obtain auto-generated fixed name with a fileset callback, use the command:

```
get_instance_property instance_name HDLINSTANCE_GET_GENERATED_NAME
```

  Note: You can only use this command with a fileset callback. This command returns the value of the auto-generated fixed name, which you can then use to instantiate inside the top-level HDL.

Example 21. Typical Usage of the add_hdl_instance Command for Generated Components

Platform Designer generates a wrapper file for the instance name specified in the `_hw.tcl` file.

```
package require -exact qsys 14.0
set_module_property name generated_toplevel_component
set_module_property ELABORATION_CALLBACK elaborate
add_fileset QUARTUS_SYNTH QUARTUS_SYNTH generate
add_fileset SIM_VERILOG SIM_VERILOG generate
add_fileset SIM_VHDL SIM_VHDL generate

proc elaborate {} {

 # Actual API to instantiate an IP Core
 add_hdl_instance emif_instance_name altera_mem_if_ddr3_emif

 # Make sure the parameters are set appropriately
 set_instance_parameter_value emif_instance_name SPEED_GRADE {7}
 ...
 # instruct Platform Designer to use auto generated fixed name
 set_instance_property emif_instance_name \
 HDLINSTANCE_USE_GENERATED_NAME 1
}

proc generate { entity_name } {

 # get the autogenerated name for emif_instance_name added
 # via add_hdl_instance

 set autogeneratedfixedname [get_instance_property \
 emif_instance_name HDLINSTANCE_GET_GENERATED_NAME]

 set fileID [open "generated_toplevel_component.v" r]
 set temp ""

 # read the contents of the file

 while {[eof $fileID] != 1} {
 gets $fileID lineInfo

 # replace the top level entity name with the name provided
 # during generation

 regsub -all "substitute_entity_name_here" $lineInfo \
```

```
 "${entity_name}" lineInfo

 # replace the autogenerated name for emif_instance_name added
 # via add_hdl_instance

 regsub -all "substitute_autogenerated_emifinstancename_here" \
 $lineInfo"${autogeneratedfixedname}" lineInfo \
 append temp "${lineInfo}\n"
}

# adding a top level component file

add_fileset_file ${entity_name}.v VERILOG TEXT $temp
}
```

Example 22. Top-Level HDL Instance and Wrapper File Created By Platform Designer

```
// Top Level Component HDL

module substitute_entity_name_here (input_wire, output_wire,
inout_wire);

input [31:0] input_wire;
output [31:0] output_wire;
inout [31:0] inout_wire;

// Instantiation of the instance added via add_hdl_instance
// command. This is an example of how the instance added
// via add_hdl_instance command can be used
// in the top-level file of the component.

substitute_autogenerated_emifinstancename_here
fixed_name_instantiation_in_top_level (
.pll_ref_clk (input_wire), // pll_ref_clk.clk
.global_reset_n (input_wire), // global_reset.reset_n
.soft_reset_n (input_wire), // soft_reset.reset_n
...
... );
endmodule

// Wrapper for added HDL instance
// generated_toplevel_component_0_emif_instance_name.v is the
// auto generated //emif_instance_name
// Generated using ACDS version 13.

`timescale 1 ps / 1 ps
module generated_toplevel_component_0_emif_instance_name (
input wire pll_ref_clk, // pll_ref_clk.clk
input wire global_reset_n, // global_reset.reset_n
input wire soft_reset_n, // soft_reset.reset_n
output wire afi_clk, // afi_clk.clk
...
...);
example_addhdlinstance_system_add_hdl_instance_example_0_emif
_instance_name_emif_instance_name emif_instance_name (

.pll_ref_clk (pll_ref_clk), // pll_ref_clk.clk
.global_reset_n (global_reset_n), // global_reset.reset_n
.soft_reset_n (soft_reset_n), // soft_reset.reset_n
...
...);
endmodule
```

Related Information

Control File Generation Dynamically with Parameters and a Fileset Callback on page 194

3.10.3. Design Guidelines for Adding Component Instances

In order to promote standard and predictable results when generating static and generated components, follow these best-practices:

- For two different parameterizations of a component, a component must never generate a file of the same name with different instantiations. The contents of a file of the same name must be identical for every parameterization of the component.
- If a component generates a nested component, it must never instantiate two different parameterizations of the nested component using the same instance name. If the parent component's parameterization affects the parameters of the nested component, the parent component must use a unique instance name for each unique parameterization of the nested component.
- Static components that generate differently based on parameterization have the potential to cause problems in the following cases:
  - Different file names with the same entity names, results in same entity conflicts at compilation-time
  - Different contents with the same file name results in overwriting other instances of the component, and in either file, compile-time conflicts or unexpected behavior.
- Generated components that generate files not based on the output name and that have different content results in either compile-time conflicts, or unexpected behavior.
- If preserving a custom component as part of an Quartus Prime Archive (`.qar`), you must first explicitly add the component `_hw.tcl` file to the project to ensure that the `.qar` includes the component. Click **Project ➤ Add/Remove Files in Project** to add files to your project.

3.11. Add IP RTL Core Generated from the Intel oneAPI Base Toolkit

The Intel oneAPI Base Toolkit is a core set of tools and libraries for developing high-performance, data-centric applications across diverse architectures. The toolkit features the Intel oneAPI DPC++/C++ Compiler that implements SYCL*, an evolution of C++ for heterogeneous computing. The compiler uses SYCL* code to generate RTL IP cores, depending on the compilation target that you specify.

For RTL IP cores, you set the compilation target to a supported Intel FPGA device family or part number instead of a specific acceleration platform. Users of the Intel HLS Compiler are encouraged to migrate existing designs to the Intel oneAPI Base Toolkit, as Intel HLS Compiler is planned for deprecation after Quartus Prime Pro Edition version 23.4.

To learn more about the Intel oneAPI FPGA flows in Platform Designer that use SYCL HLS, refer to the *Intel oneAPI FPGA Handbook* and the *Platform Designer Sample Tutorial*.

Related Information

- Intel oneAPI FPGA Handbook
- Platform Designer Sample Tutorial

3.12. Creating Platform Designer Components Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout. • Added new *Add IP RTL Core Generated from the Intel oneAPI Base Toolkit* topic. • Removed deprecated "Adding Generic HLS Components" section.
2023.06.26	23.2	• Corrected code sample typo in *Create a Composed Component or Subsystem* topic.
2023.04.03	23.1	• Updated examples in *Declaring SystemVerilog Interfaces in _hw.tcl* topic. • Removed *Name HDL Signals for Automatic Interface and Type Recognition* topic as this feature is not currently supported. • The product family name is updated to "Intel Agilex 7" to reflect the different family members.
2022.04.02	22.1	• Updated entire chapter for new AXI "manager" and AXI "subordinate" replacement terms. Refer to the AMBA® AXI and ACE Protocol Specification.
2021.03.29	21.1	• Converted to "host" and "agent" inclusive terminology for Avalon memory mapped interface descriptions and related GUI elements throughout.
2020.12.14	20.4	• Added new "Exporting HDL Parameters to a System" topic. • Added new "HDL Parameters Tab Settings and Controls" topic.
2020.09.28	20.3	• Revised "Adding Generic Components to a System" to describes all implementations and refer to substeps for HDL and Blackbox parameters. • Revised "Adding Generic HDL Component Parameters" for latest steps and screenshot. • Added new "Adding Generic Blackbox Component Parameters" topic. • Added statement about `AFFECTS_GENERATION` to "Specify Parameters in the Platform Designer Component Editor" topic. • Reorganized the order of some topic to group similar items.
2020.05.01	20.1	• Added note about `.qar` file requirements to "Design Guidelines for Component Instances" topic.
2020.04.06	18.1.0	• Updated links and web page names in "Component Structure" and "Creating Platform Designer Components" topics.
2019.06.19	18.1.0	• Added descriptions of AXI parameters in "Specify Parameters in the Platform Designer Component Editor."
2018.12.15	18.1.0	• Replaced references to System Contents tab with new System View tab.
2018.05.07	18.0	• Added scripting support for wire-level expressions.
2017.11.06	17.1.0	• Changed instances of *Qsys Pro* to *Platform Designer* • Replaced mentions of altera_axi_default_slave to altera_error_response_slave • Added support for SystemVerilog interfaces with `_hw.tcl`. • Added support for user alterable HDL parameters with `_hw.tcl`. • Added support for High Level Synthesis file compilation.
2017.05.08	17.0.0	• Updated Figure: Address Span Extender

continued...

Document Version	Quartus Prime Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented Qsys rebranding. • Added topics for Generic Component.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Updated screen shots **Files** tab, Qsys Component Editor. • Added topic: *Specify Interfaces and Signals in the Qsys Component Editor*. • Added topic: *Create an HDL File in the Qsys Component Editor*. • Added topic: *Create an HDL File Using a Template in the Qsys Component Editor*.
November 2013	13.1.0	• `add_hdl_instance` • Added *Creating a Component With Differing Structural Qsys View and Generated Output Files*.
May 2013	13.0.0	• Consolidated content from other Qsys chapters. • Added *Upgrading IP Components to the Latest Version*. • Updated for AMBA APB support.
November 2012	12.1.0	• Added AMBA AXI4 support. • Added the **demo_axi_memory** example with screen shots and example `_hw.tcl` code.
June 2012	12.0.0	• Added new tab structure for the Component Editor. • Added AXI 3 support.
November 2011	11.1.0	Template update.
May 2011	11.0.0	• Removed beta status. • Added Avalon Tri-state Conduit (Avalon-TC) interface type. • Added many interface templates for Nios custom instructions and Avalon-TC interfaces.
December 2010	10.1.0	Initial release.

4. Optimizing Platform Designer System Performance

Platform Designer provides tools that allow you to optimize the performance of the system interconnect for Intel FPGA designs. This chapter presents techniques that leverage the available tools and the trade offs of each implementation.

The foundation of any system is the interconnect logic that connects hardware blocks or components. Creating interconnect logic is time consuming and prone to errors, and existing interconnect logic is difficult to modify when design requirements change. The Platform Designer system integration tool addresses these issues and provides an automatically generated and optimized interconnect designed to satisfy the system requirements.

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

Note: Recommended Intel practices may improve clock frequency, throughput, logic utilization, or power consumption of a Platform Designer design. When you design a Platform Designer system, use your knowledge of the design intent and goals to further optimize system performance beyond the automated optimization available in Platform Designer.

Related Information

- Creating a System with Platform Designer on page 11
- Creating Platform Designer Components on page 153
- Platform Designer Interconnect on page 251
- Avalon Interface Specifications
- AMBA Protocol Specifications

4.1. Designing with Avalon and AXI Interfaces

The Avalon and AXI interconnects for memory-mapped interfaces are flexible, partial crossbar logic. For AXI, the interconnect connects the AXI manager to the AXI subordinate interfaces. Similarly, the interconnect connects the Avalon host to the agent interfaces.(9)

Avalon Streaming links connect point-to-point, unidirectional interfaces and are typically used in data stream applications. Each pair of components is connected without any requirement to arbitrate between the data source and sink.

(9) This document now refers to the Avalon "host" and "agent," and the AXI "manager" and "subordinate," to replace formerly used terms. Refer to the current *AMBA AXI and ACE Protocol Specification* for the latest AMBA AXI and ACE protocol terminology.

Because Platform Designer supports multiplexed memory-mapped and streaming connections, you can implement systems that use multiplexed logic for control and streaming for data in a single design.

Related Information

Creating Platform Designer Components on page 153

4.1.1. Designing Streaming Components

When you design streaming component interfaces, you must consider integration and communication for each component in the system. One common consideration is buffering data internally to accommodate latency between components.

For example, if the component's Avalon streaming output or source of streaming data is back-pressured because the ready signal is deasserted, then the component must back-pressure its input or sink interface to avoid overflow.

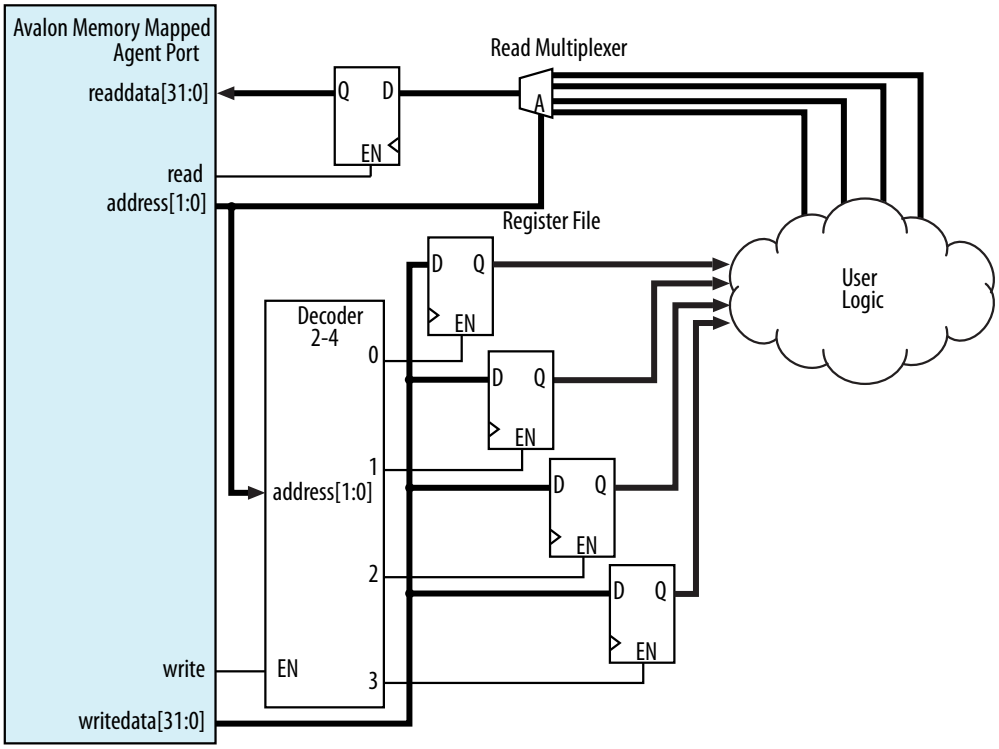
You can use a FIFO to back-pressure internally on the output side of the component so that the input can accept more data even if the output is back-pressured. Then, you can use the FIFO almost full flag to back-pressure the sink interface or input data when the FIFO has only enough space to satisfy the internal latency. You can drive the data valid signal of the output or source interface with the FIFO not empty flag when that data is available.

4.1.2. Designing Memory-Mapped Components

When designing with memory-mapped components, you can implement any component that contains multiple registers mapped to memory locations, for example, a set of four output registers to support software read back from logic. Components that implement read and write memory-mapped transactions require three main building blocks: an address decoder, a register file, and a read multiplexer.

The decoder enables the appropriate 32-bit or 64-bit register for writes. For reads, the address bits drive the multiplexer selection bits. The read signal registers the data from the multiplexer, adding a pipeline stage so that the component can achieve a higher clock frequency.

Figure 156. Control and Status Registers (CSR) in an Agent Component



This agent component has four write wait states and one read wait state. Alternatively, if you want high throughput, you may set both the read and write wait states to zero, and then specify a read latency of one, because the component also supports pipelined reads.

4.2. Using Hierarchy in Systems

You can use hierarchy to sub-divide a system into smaller subsystems that you can then connect in a top-level Platform Designer system. Additionally, if a design contains one or more identical functional units, the functional unit can be defined as a subsystem and instantiated multiple times within a top-level system.

Hierarchy can simplify verification control of agents connected to each host in a memory-mapped system. Before you implement subsystems in your design, you should plan the system hierarchical blocks at the top-level, using the following guidelines:

- **Plan shared resources**—Determine the best location for shared resources in the system hierarchy. For example, if two subsystems share resources, add the components that use those resources to a higher-level system for easy access.
- **Plan shared address space between subsystems**—Planning the address space ensures you can set appropriate sizes for bridges between subsystems.
- **Plan how much latency you may need to add to your system**—When you add an Avalon Memory Mapped Pipeline Bridge between subsystems, you may add latency to the overall system. You can reduce the added latency by parameterizing the bridge with zero cycles of latency, and by turning off the pipeline command and response signals.

Figure 157. Avalon Memory Mapped Pipeline Bridge

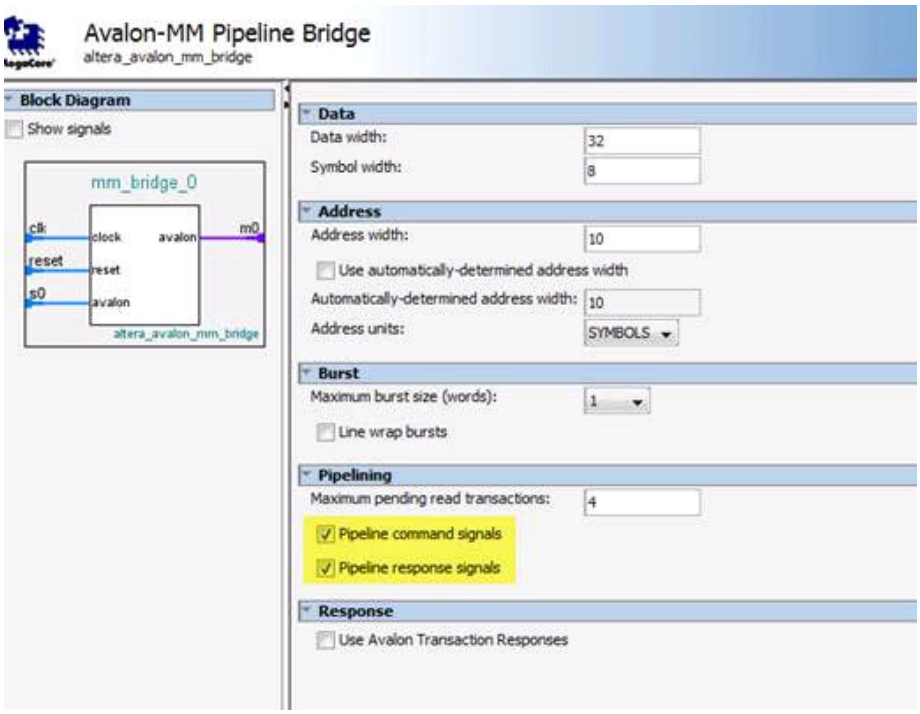
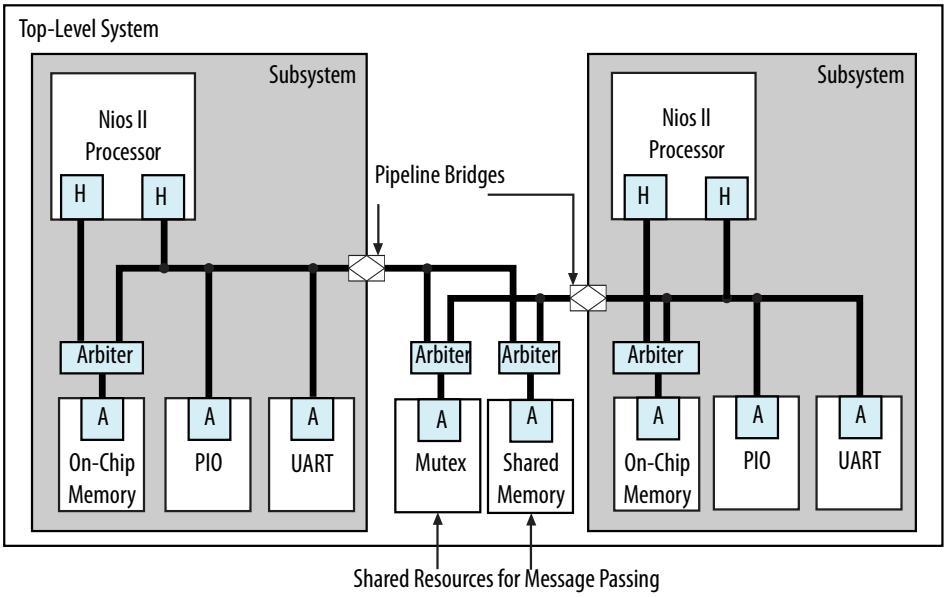
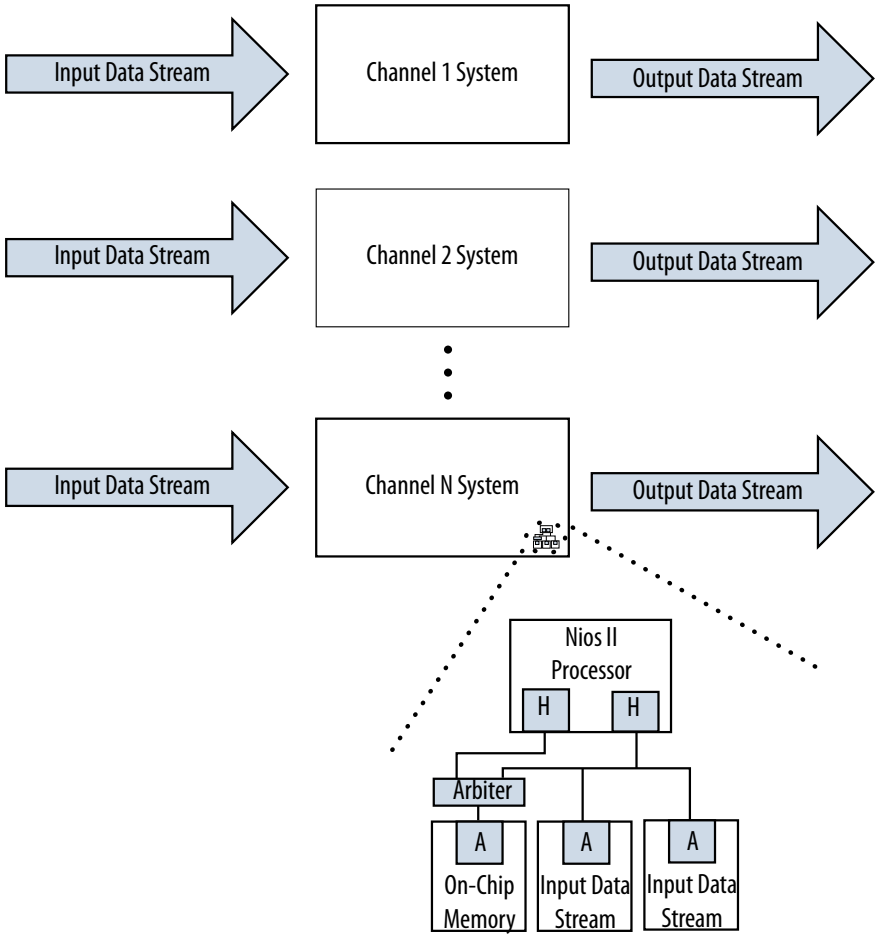


Figure 158. Passing Messages Between Subsystems



In this example, two Nios II processor subsystems share resources for message passing. Bridges in each subsystem export the Nios II data host to the top-level system that includes the mutex (mutual exclusion component) and shared memory component (which could be another on-chip RAM, or a controller for an off-chip RAM device).

Figure 159. Multi Channel System



You can also design systems that process multiple data channels by instantiating the same subsystem for each channel. This approach is easier to maintain than a larger, non-hierarchical system. Additionally, such systems are easier to scale because you can calculate the required resources as a multiple of the subsystem requirements.

4.3. Using Concurrency in Memory-Mapped Systems

Platform Designer interconnect uses parallel hardware in FPGAs, which allows you to design concurrency into your system and process transactions simultaneously.

4.3.1. Implementing Concurrency With Multiple Hosts

Implementing concurrency requires multiple hosts in a Platform Designer system. Systems that include a processor contain at least two host interfaces because the processors include separate instruction and data hosts. You can categorize host components as follows:

- General purpose processors, such as Nios II processors
- DMA (direct memory access) engines
- Communication interfaces, such as PCI Express

Because Platform Designer generates an interconnect with agent-side arbitration, every host interface in a system can issue transfers concurrently, if they are not posting transfers to the same agent. Concurrency is limited by the number of host interfaces sharing any particular agent interface. If a design requires higher data throughput, you can increase the number of host and agent interfaces to increase the number of transfers that occur simultaneously. The example below shows a system with three host interfaces.

Figure 160. Avalon Multiple Host Parallel Access

In this Avalon example, the DMA engine operates with Avalon memory mapped read and write hosts. The yellow lines represent active simultaneous connections.

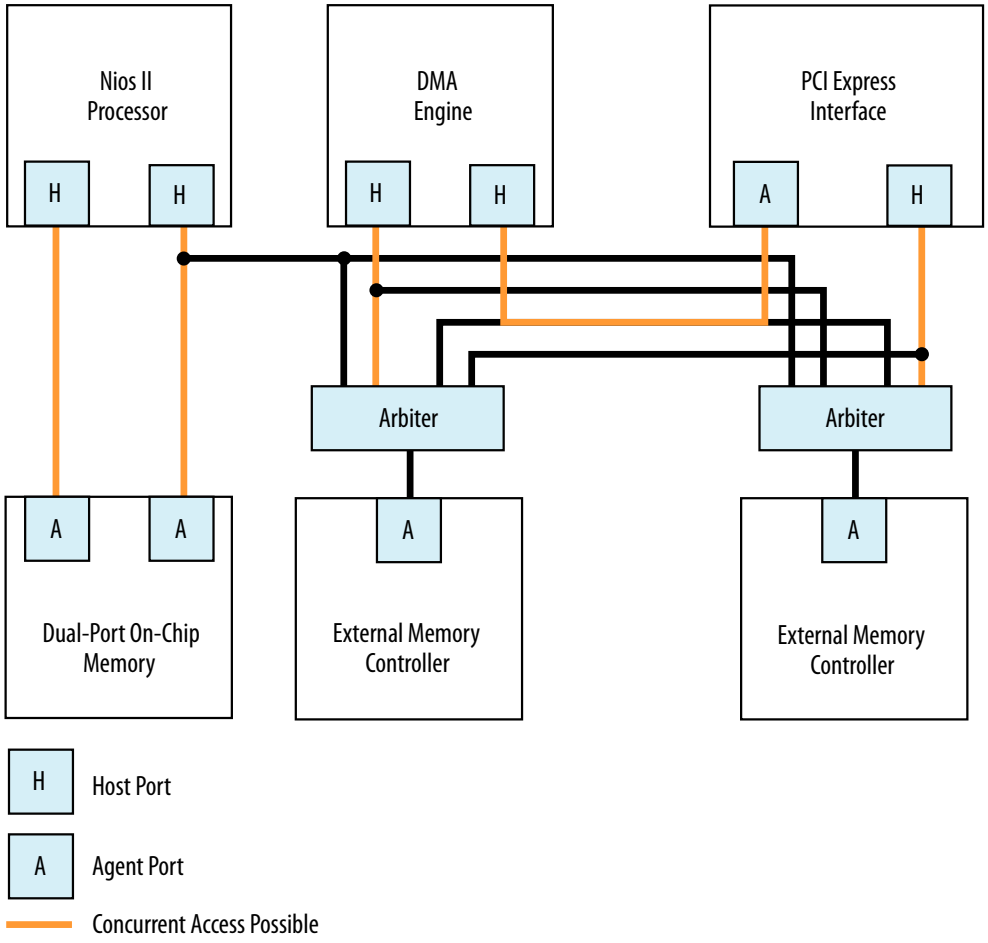
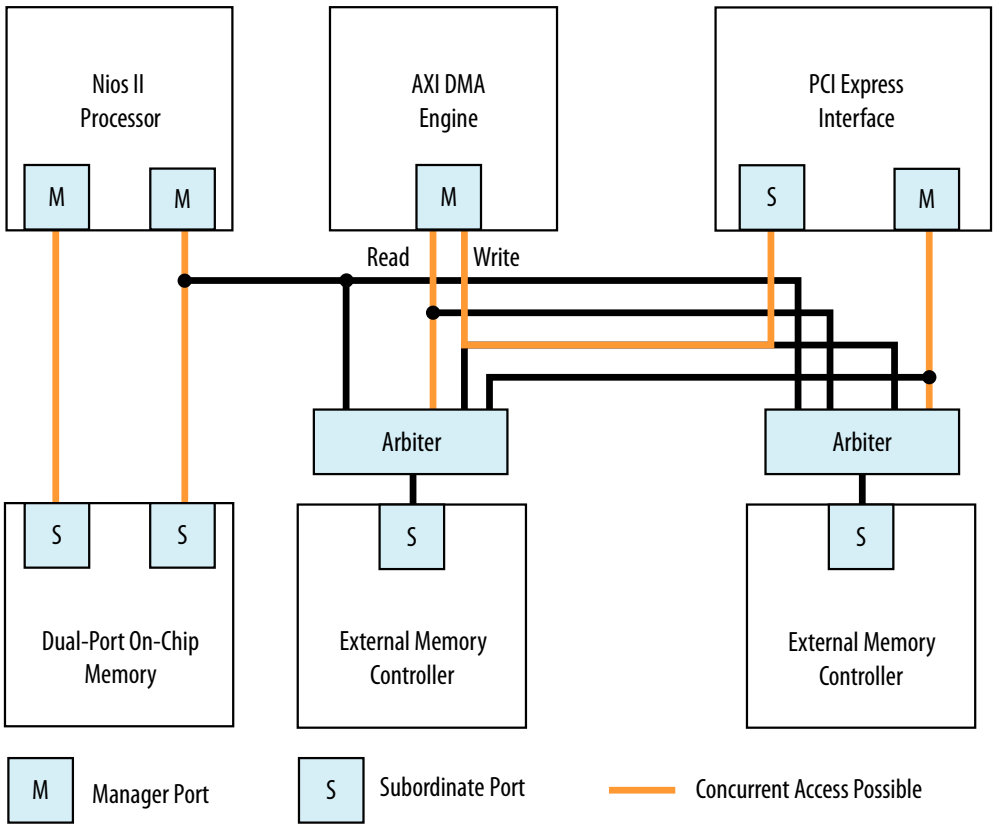


Figure 161. AXI Multiple Manager Parallel Access

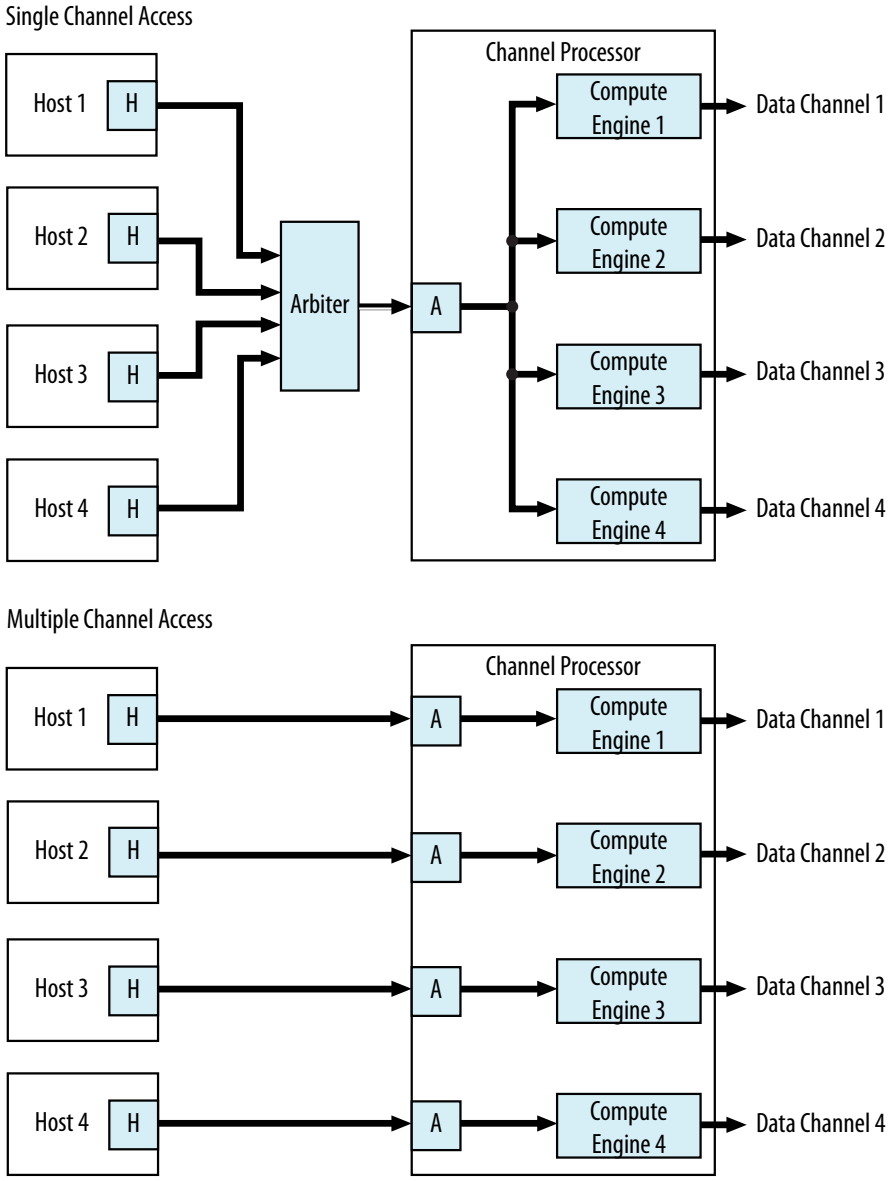
In this example, the DMA engine operates with a single manager, because in AXI, the write and read channels on the manager are independent and can process transactions simultaneously. There is concurrency between the read and write channels, with the yellow lines representing concurrent datapaths.



4.3.2. Implementing Concurrency With Multiple Agents

You can create multiple agent interfaces for a particular function to increase concurrency in your design.

Figure 162. Single Interface Versus Multiple Interfaces



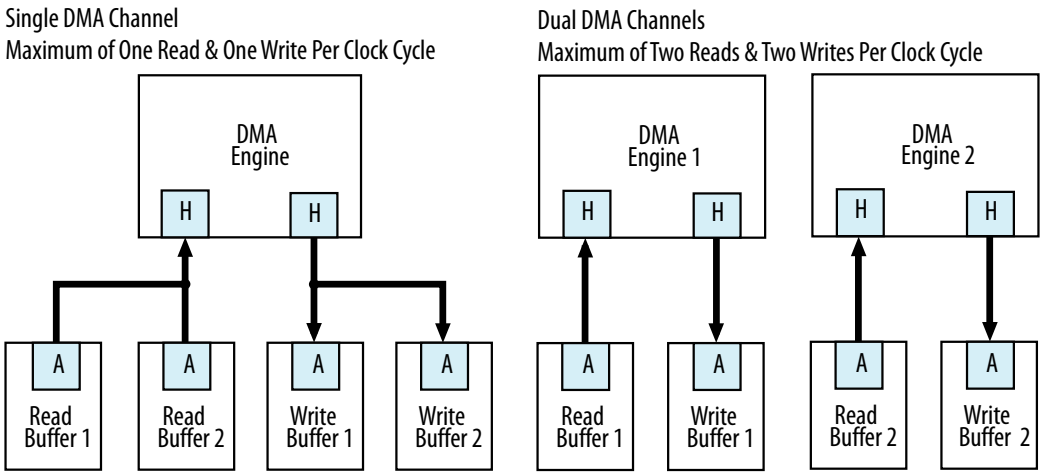
In this example, there are two channel processing systems. In the first, four hosts must arbitrate for the single agent interface of the channel processor. In the second, each host drives a dedicated agent interface, allowing all host interfaces to simultaneously access the agent interfaces of the component. Arbitration is not necessary when there is a single host and agent interface.

4.3.3. Implementing Concurrency with DMA Engines

In some systems, you can use DMA engines to increase throughput. You can use a DMA engine to transfer blocks of data between interfaces, which then frees the CPU from doing this task. A DMA engine transfers data between a programmed start and

end address without intervention, and the data throughput is dictated by the components connected to the DMA. Factors that affect data throughput include data width and clock frequency.

Figure 163. Single or Dual DMA Channels



In this example, the system can sustain more concurrent read and write operations by including more DMA engines. Accesses to the read and write buffers in the top system are split between two DMA engines, as shown in the Dual DMA Channels at the bottom of the figure.

The DMA engine operates with Avalon memory mapped write and read hosts. An AXI DMA typically has only one manager, because in AXI, the write and read channels on the manager are independent and can process transactions simultaneously.

4.4. Inserting Pipeline Stages to Increase System Frequency

Adding pipeline stages may increase the f_{MAX} of the design by reducing the combinational logic depth, at the cost of additional latency and logic utilization.

Platform Designer provides the **Limit interconnect pipeline stages to** option on the **Interconnect Requirements** tab to automatically add pipeline stages to the Platform Designer interconnect when you generate a system.

The **Limit interconnect pipeline stages to** parameter in the **Interconnect Requirements** tab allows you to define the maximum Avalon streaming pipeline stages that Platform Designer can insert during generation. You can specify between 0 to 4 pipeline stages, where 0 means that the interconnect has a combinational datapath. You can specify a unique interconnect pipeline stage value for each subsystem.

For more information, refer to *Interconnect Pipelining*.

Related Information

Pipelined Avalon Memory Mapped Interfaces on page 229

4.5. Using Bridges

You can use bridges to increase system frequency, minimize generated Platform Designer logic, minimize adapter logic, and to structure system topology when you want to control where Platform Designer adds pipelining. You can also use bridges with arbiters when there is concurrency in the system.

An Avalon bridge has an Avalon memory mapped agent interface and an Avalon memory mapped host interface. You can have many components connected to the bridge agent interface, or many components connected to the bridge host interface. You can also have a single component connected to a single bridge agent or host interface.

You can configure the data width of the bridge, which can affect how Platform Designer generates bus sizing logic in the interconnect. Both interfaces support Avalon memory mapped pipelined transfers with variable latency, and can also support configurable burst lengths.

Transfers to the bridge agent interface are propagated to the host interface, which connects to components downstream from the bridge. Bridges can provide more control over interconnect pipelining than the **Limit interconnect pipeline stages to** option.

Note: You can use Avalon bridges between AXI interfaces, and between Avalon domains. Platform Designer automatically creates interconnect logic between the AXI and Avalon interfaces, so you do not have to explicitly instantiate bridges between these domains. For more discussion about the benefits and disadvantages of shared and separate domains, refer to the *Platform Designer Interconnect*.

Related Information

- Bridges on page 367
- AMBA 3 APB Protocol Specification Support (version 1.0) on page 336

4.5.1. Using Bridges to Increase System Frequency

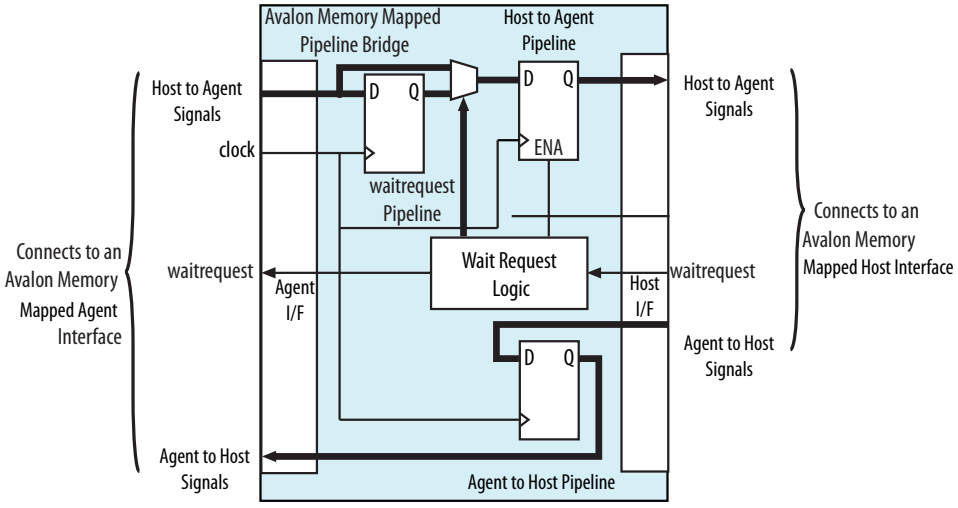
In Platform Designer, you can introduce interconnect pipeline stages or pipeline bridges to increase clock frequency in your system. Bridges control the system interconnect topology and allow you to subdivide the interconnect, giving you more control over pipelining and clock crossing functionality.

4.5.1.1. Inserting Pipeline Bridges

You can insert an Avalon memory mapped pipeline bridge to insert registers in the path between the bridges and its host and agents. If a critical register-to-register delay occurs in the interconnect, a pipeline bridge can help reduce this delay and improve system f_{MAX}.

The Avalon memory mapped pipeline bridge component integrates into any Platform Designer system. The pipeline bridge options can increase logic utilization and read latency. The change in topology may also reduce concurrency if multiple hosts arbitrate for the bridge. You can use the Avalon memory mapped pipeline bridge to control topology without adding a pipeline stage. A pipeline bridge that does not add a pipeline stage is optimal in some latency-sensitive applications. For example, a CPU may benefit from minimal latency when accessing memory.

Figure 164. Avalon Memory Mapped Pipeline Bridge

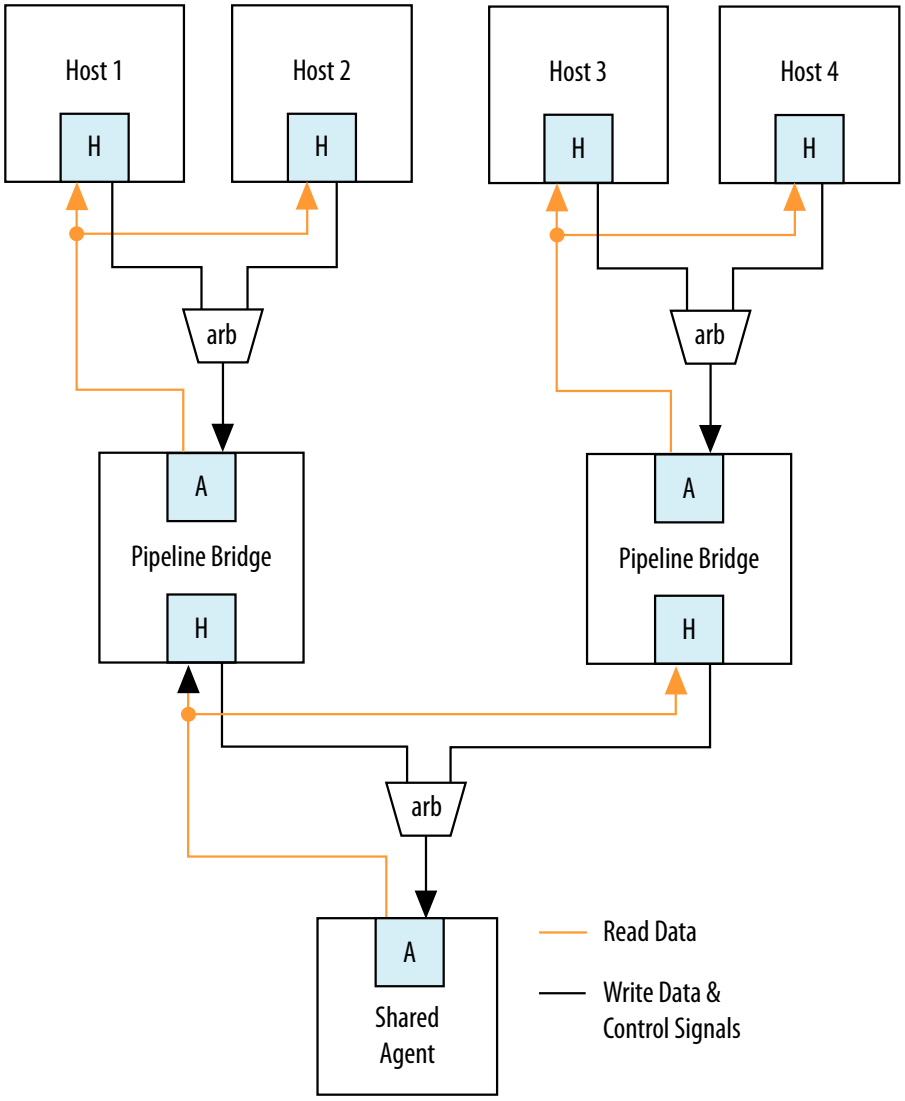


4.5.1.1.1. Implementing Command Pipelining (Host-to-Agent)

When multiple hosts share an agent device, you can use command pipelining to improve performance.

The arbitration logic for the agent interface must multiplex the `address`, `writedata`, and `burstcount` signals. The multiplexer width increases proportionally with the number of hosts connecting to a single agent interface. The increased multiplexer width may become a timing critical path in the system. If a single pipeline bridge does not provide enough pipelining, you can instantiate multiple instances of the bridge in a tree structure to increase the pipelining and further reduce the width of the multiplexer at the agent interface.

Figure 165. Tree of Bridges



4.5.1.1.2. Implementing Response Pipelining (Agent-to-Host)

When hosts connect to multiple agents that support read transfers, you can use agent-to-host pipelining to improve performance.

The interconnect inserts a multiplexer for every read datapath back to the host. As the number of agents supporting read transfers connecting to the host increases, the width of the read data multiplexer also increases. If the performance increase is insufficient with one bridge, you can use multiple bridges in a tree structure to improve f_{MAX}.

4.5.1.2. Using Clock Crossing Bridges

The clock crossing bridge contains a pair of clock crossing FIFOs, which isolate the host and agent interfaces in separate, asynchronous clock domains. Transfers to the agent interface are propagated to the host interface.

When you use a FIFO clock crossing bridge for the clock domain crossing, you add data buffering. Buffering allows pipelined read hosts to post multiple reads to the bridge, even if the agents downstream from the bridge do not support pipelined transfers.

You can also use a clock crossing bridge to place high and low frequency components in separate clock domains. If you limit the fast clock domain to the portion of your design that requires high performance, you may achieve a higher f_{MAX} for this portion of the design. For example, the majority of processor peripherals in embedded designs do not need to operate at high frequencies, therefore, you do not need to use a high-frequency clock for these components. When you compile a design with the Quartus Prime software, compilation may take more time when the clock frequency requirements are difficult to meet because the Fitter needs more time to place registers to achieve the required f_{MAX}. To reduce the amount of effort that the Fitter uses on low priority and low performance components, you can place these behind a clock crossing bridge operating at a lower frequency, allowing the Fitter to increase the effort placed on the higher priority and higher frequency datapaths.

4.5.2. Using Bridges to Minimize Design Logic

Bridges can reduce interconnect logic by reducing the amount of arbitration and multiplexer logic that Platform Designer generates. This reduction occurs because bridges limit the number of concurrent transfers that can occur.

4.5.2.1. Avoiding Speed Optimizations That Increase Logic

You can add an additional pipeline stage with a pipeline bridge between hosts and agents to reduce the amount of combinational logic between registers, which can increase system performance. If you can increase the f_{MAX} of your design logic, you may be able to turn off the Quartus Prime software optimization settings, such as the **Perform register duplication** setting. Register duplication creates duplicate registers in two or more physical locations in the FPGA to reduce register-to-register delays. You may also want to choose **Speed** for the optimization method, which typically results in higher logic utilization due to logic duplication. By making use of the registers or FIFOs available in the bridges, you can increase the design speed and avoid needless logic duplication or speed optimizations, thereby reducing the logic utilization of the design.

4.5.2.2. Limiting Concurrency

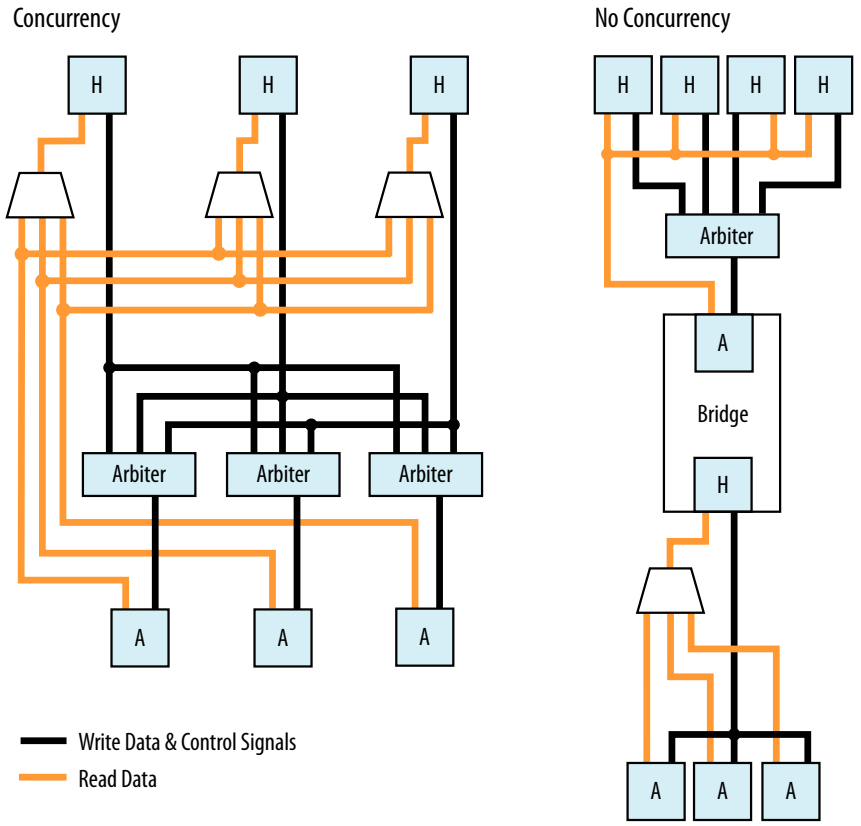
The amount of logic generated for the interconnect often increases as the system becomes larger because Platform Designer creates arbitration logic for every agent interface that is shared by multiple host interfaces. Platform Designer inserts multiplexer logic between host interfaces that connect to multiple agent interfaces if both support read datapaths.

Most embedded processor designs contain components that are either incapable of supporting high data throughput, or do not need to be accessed frequently. These components can contain host or agent interfaces. Because the interconnect supports concurrent accesses, you may want to limit concurrency by inserting bridges into the datapath to limit the amount of arbitration and multiplexer logic generated.

For example, if a system contains three host and three agent interfaces that are interconnected, Platform Designer generates three arbiters and three multiplexers for the read datapath. If these hosts do not require a significant amount of simultaneous throughput, you can reduce the resources that your design consumes by connecting the three hosts to a pipeline bridge. The bridge controls the three agent interfaces and reduces the interconnect into a bus structure. Platform Designer creates one arbitration block between the bridge and the three hosts, and a single read datapath multiplexer between the bridge and three agents, and prevents concurrency. This implementation is similar to a standard bus architecture.

You should not use this method for high throughput datapaths to ensure that you do not limit overall system performance.

Figure 166. Differences Between Systems With and Without a Pipeline Bridge



4.5.3. Using Bridges to Minimize Adapter Logic

Platform Designer generates adapter logic for clock crossing, width adaptation, and burst support when there is a mismatch between the clock domains, widths, or bursting capabilities of the host and agent interface pairs.

Platform Designer creates burst adapters when the maximum burst length of the host is greater than the host burst length of the agent. The adapter logic creates extra logic resources, which can be substantial when your system contains host interfaces connected to many components that do not share the same characteristics. By placing bridges in your design, you can reduce the amount of adapter logic that Platform Designer generates.

4.5.3.1. Determining Effective Placement of Bridges

To determine the effective placement of a bridge, you should initially analyze each host in your system to determine if the connected agent devices support different bursting capabilities or operate in a different clock domain. The maximum burstcount of a component is visible as the `burstcount` signal in the HDL file of the component. The maximum burst length is $2^{(\text{width(burstcount -1)})}$, therefore, if the `burstcount` width is four bits, the maximum burst length is eight. If no `burstcount` signal is present, the component does not support bursting or has a burst length of 1.

To determine if the system requires a clock crossing adapter between the host and agent interfaces, check the **Clock** column for the host and agent interfaces. If the clock is different for the host and agent interfaces, Platform Designer inserts a clock crossing adapter between them. To avoid creating multiple adapters, you can place the components containing agent interfaces behind a bridge so that Platform Designer creates a single adapter. By placing multiple components with the same burst or clock characteristics behind a bridge, you limit concurrency and the number of adapters.

You can also use a bridge to separate AXI and Avalon domains to minimize burst adaptation logic. For example, if there are multiple Avalon agents that are connected to an AXI manager, you can consider inserting a bridge to access the adaptation logic once before the bridge, instead of once per agent. This implementation results in latency, and you would also lose concurrency between reads and writes.

4.5.3.2. Changing the Response Buffer Depth

When you use automatic clock-crossing adapters, Platform Designer determines the required depth of FIFO buffering based on the agent properties. If an agent has a high **Maximum Pending Reads** parameter, the resulting deep response buffer FIFO that Platform Designer inserts between the host and agent can consume a lot of device resources. To control the response FIFO depth, you can use a clock crossing bridge and manually adjust its FIFO depth to trade off throughput with smaller memory utilization.

For example, if you have hosts that cannot saturate the agent, you do not need response buffering. Using a bridge reduces the FIFO memory depth and reduces the **Maximum Pending Reads** available from the agent.

4.5.4. Considering the Effects of Using Bridges

Before you use pipeline or clock crossing bridges in a design, you should carefully consider their effects. Bridges can have any combination of consequences on your design, which could be positive or negative. Benchmarking your system before and after inserting bridges can help you determine the impact to the design.

4.5.4.1. Increased Latency

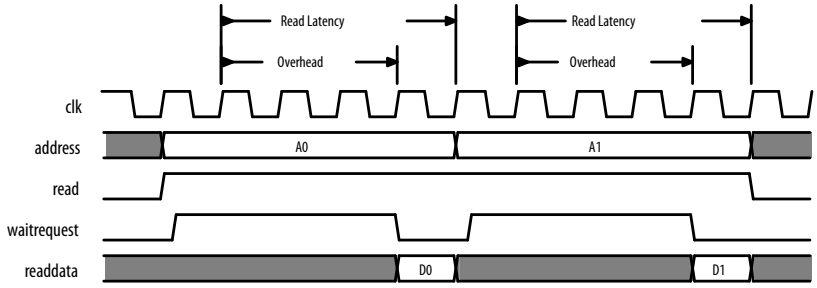
Adding a bridge to a design has an effect on the read latency between the host and the agent. Depending on the system requirements and the type of host and agent, this latency increase may not be acceptable in your design.

4.5.4.1.1. Acceptable Latency Increase

For a pipeline bridge, Platform Designer adds a cycle of latency for each pipeline option that is enabled. The buffering in the clock crossing bridge also adds latency. If you use a pipelined or burst host that posts many read transfers, the increase in latency does not impact performance significantly because the latency increase is very small compared to the length of the data transfer.

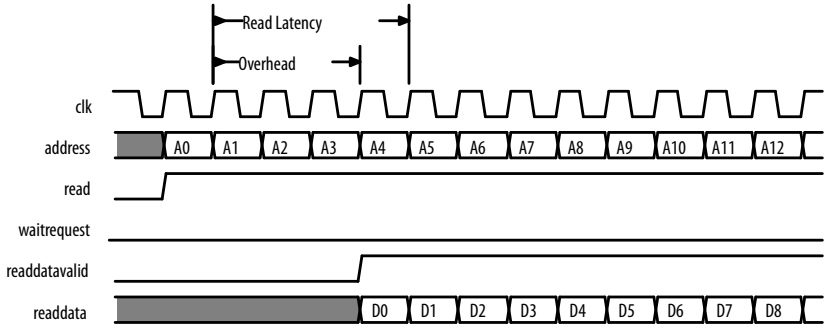
For example, if you use a pipelined read host such as a DMA controller to read data from a component with a fixed read latency of four clock cycles, but only perform a single word transfer, the overhead is three clock cycles out of the total of four. This is true when there is no additional pipeline latency in the interconnect. The read throughput is only 25%.

Figure 167. Low-Efficiency Read Transfer



However, if 100 words of data are transferred without interruptions, the overhead is three cycles out of the total of 103 clock cycles. This corresponds to a read efficiency of approximately 97% when there is no additional pipeline latency in the interconnect. Adding a pipeline bridge to this read path adds two extra clock cycles of latency. The transfer requires 105 cycles to complete, corresponding to an efficiency of approximately 94%. Although the efficiency decreased by 3%, adding the bridge may increase the f_{MAX} by 5%. For example, if the clock frequency can be increased, the overall throughput would improve. As the number of words transferred increases, the efficiency increases to nearly 100%, whether or not a pipeline bridge is present.

Figure 168. High Efficiency Read Transfer

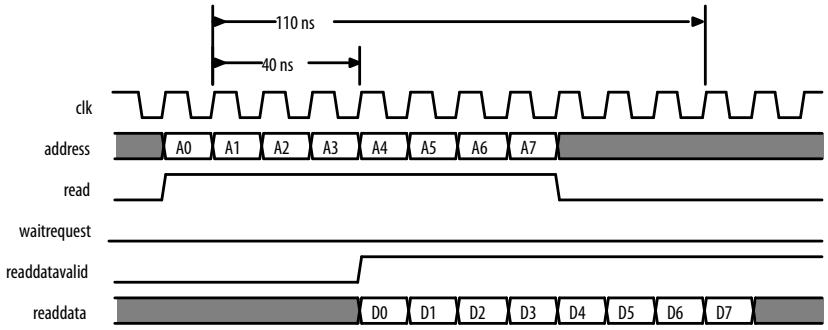


4.5.4.1.2. Unacceptable Latency Increase

Processors are sensitive to high latency read times and typically retrieve data for use in calculations that cannot proceed until the data arrives. Before adding a bridge to the datapath of a processor instruction or data host, determine whether the clock frequency increase justifies the added latency.

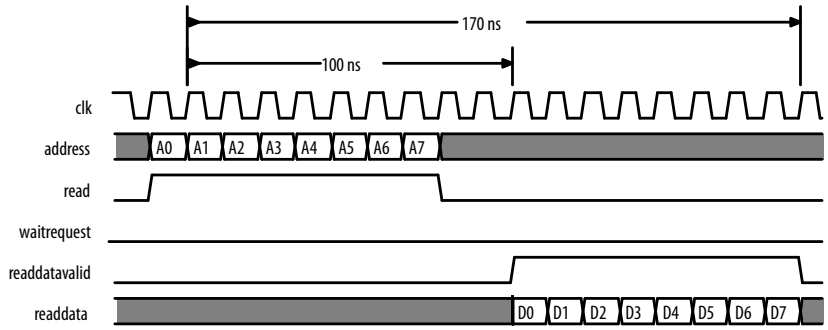
A Nios II processor instruction master has a cache memory with a read latency of four cycles, which is eight sequential words of data return for each read. At 100 MHz, the first read takes 40 ns to complete. Each successive word takes 10 ns so that eight reads complete in 110 ns.

Figure 169. Performance of a Nios II Processor and Memory Operating at 100 MHz



Adding a clock crossing bridge allows the memory to operate at 125 MHz. However, this increase in frequency is negated by the increase in latency because if the clock crossing bridge adds six clock cycles of latency at 100 MHz, then the memory continues to operate with a read latency of four clock cycles. Consequently, the first read from memory takes 100 ns, and each successive word takes 10 ns because reads arrive at the frequency of the processor, which is 100 MHz. In total, eight reads complete after 170 ns. Although the memory operates at a higher clock frequency, the frequency at which the host operates limits the throughput.

Figure 170. Performance of a Nios II Processor and Eight Reads with Ten Cycles Latency



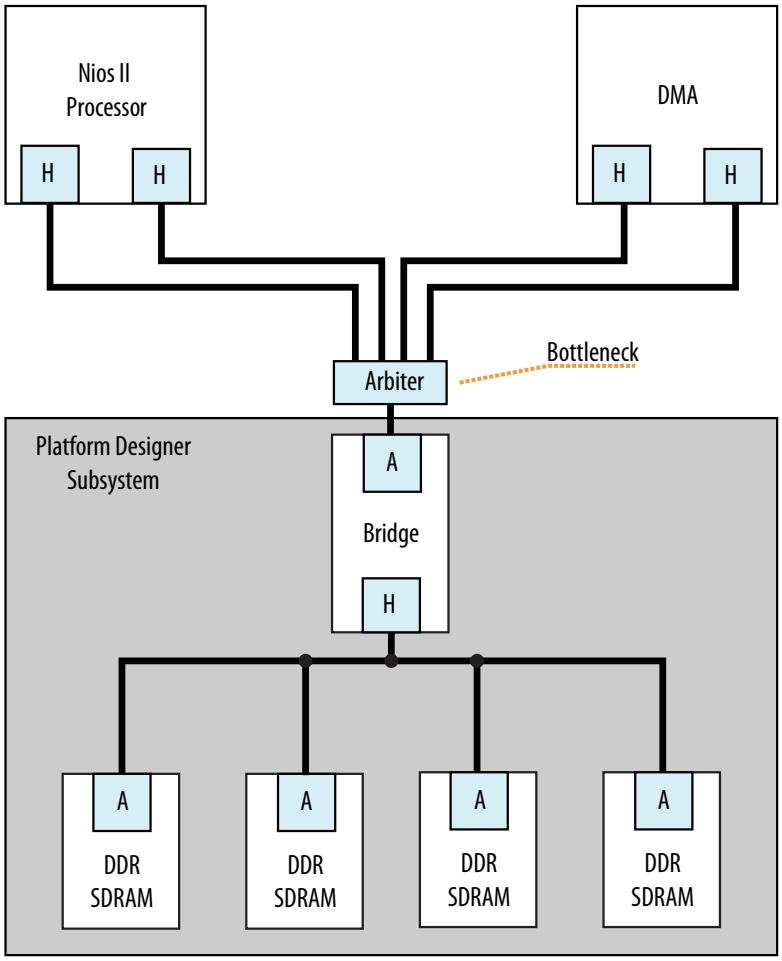
4.5.4.2. Limited Concurrency

Placing a bridge between multiple host and agent interfaces limits the number of concurrent transfers your system can initiate. This limitation is the same when connecting multiple host interfaces to a single agent interface. The agent interface of

the bridge is shared by all the hosts and, as a result, Platform Designer creates arbitration logic. If the components placed behind a bridge are infrequently accessed, this concurrency limitation may be acceptable.

Bridges can have a negative impact on system performance if you use them inappropriately. For example, if multiple memories are used by several hosts, you should not place the memory components behind a bridge. The bridge limits memory performance by preventing concurrent memory accesses. Placing multiple memory components behind a bridge can cause the separate agent interfaces to appear as one large memory to the hosts accessing the bridge; all hosts must access the same agent interface.

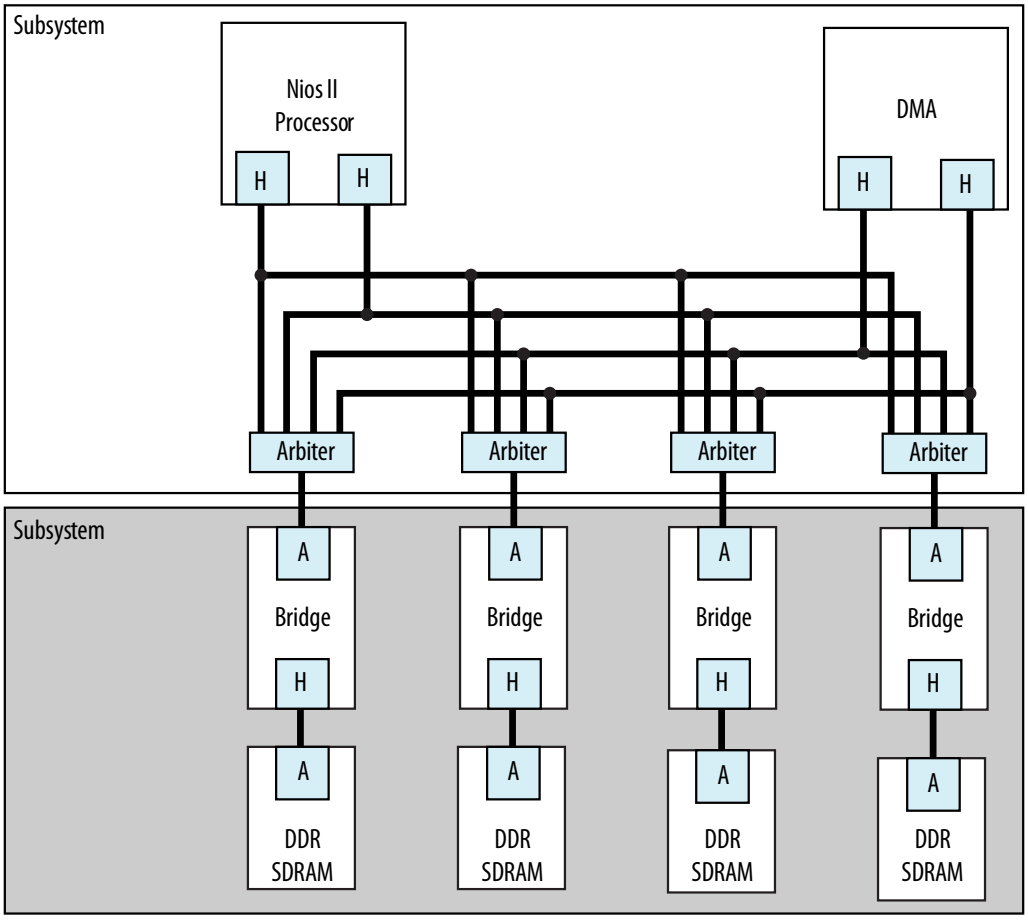
Figure 171. Inappropriate Use of a Bridge in a Hierarchical System



A memory subsystem with one bridge that acts as a single agent interface for the Avalon memory mapped Nios II and DMA hosts, which results in a bottleneck architecture. The bridge acts as a bottleneck between the two hosts and the memories.

If the f_{MAX} of your memory interfaces is low and you want to use a pipeline bridge between subsystems, you can place each memory behind its own bridge, which increases the f_{MAX} of the system without sacrificing concurrency.

Figure 172. Efficient Memory Pipelining Without a Bottleneck in a Hierarchical System

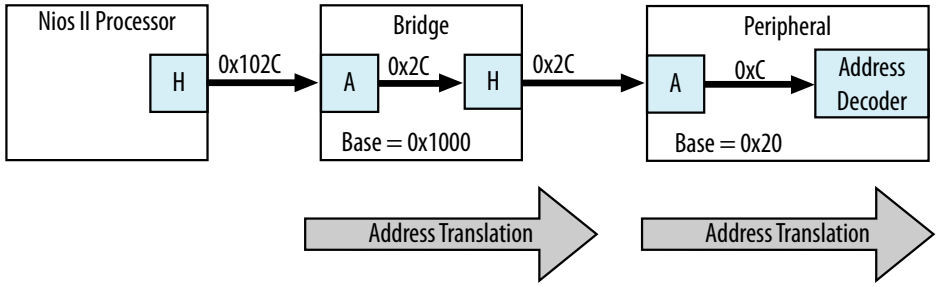


4.5.4.3. Address Space Translation

The agent interface of a pipeline or clock crossing bridge has a base address and address span. You can set the base address, or allow Platform Designer to set it automatically. The address of the agent interface is the base offset address of all the components connected to the bridge. The address of components connected to the bridge is the sum of the base offset and the address of that component.

The host interface of the bridge drives only the address bits that represent the offset from the base address of the bridge agent interface. Any time a host accesses an agent through a bridge, both addresses must be added together, otherwise the transfer fails. The **Address Map** tab displays the addresses of the agents connected to each host and includes address translations caused by system bridges.

Figure 173. Bridge Address Translation

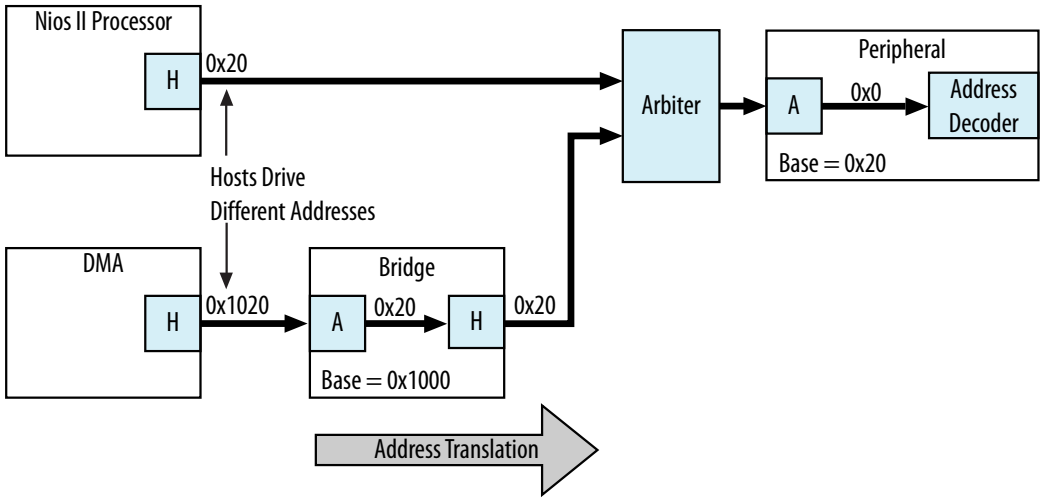


In this example, the Nios II processor connects to a bridge located at base address 0x1000, an agent connects to the bridge host interface at an offset of 0x20, and the processor performs a write transfer to the fourth 32-bit or 64-bit word within the agent. Nios II drives the address 0x102C to interconnect, which is within the address range of the bridge. The bridge host interface drives 0x2C, which is within the address range of the agent, and the transfer completes.

4.5.4.4. Address Coherency

To simplify the system design, all hosts should access agents at the same location. In many systems, a processor passes buffer locations to other hosting components, such as a DMA controller. If the processor and DMA controller do not access the agent at the same location, Platform Designer must compensate for the differences.

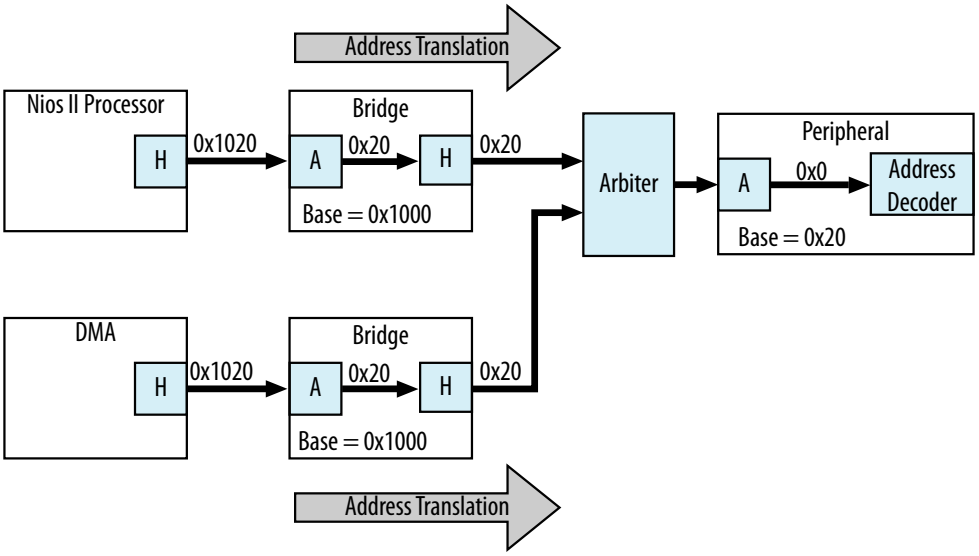
Figure 174. Agents at Different Addresses and Complicating the System



A Nios II processor and DMA controller access an agent interface located at address 0x20. The processor connects directly to the agent interface. The DMA controller connects to a pipeline bridge located at address 0x1000, which then connects to the agent interface. Because the DMA controller accesses the pipeline bridge first, it must drive 0x1020 to access the first location of the agent interface. Because the processor accesses the agent from a different location, you must maintain two base addresses for the agent device.

To avoid the requirement for two addresses, you can add an additional bridge to the system, set its base address to 0x1000, and then disable all the pipelining options in the second bridge so that the bridge has minimal impact on system timing and resource utilization. Because this second bridge has the same base address as the original bridge, the processor and DMA controller access the agent interface with the same address range.

Figure 175. Address Translation Corrected With Bridge



4.6. Increasing Transfer Throughput

Increasing the transfer efficiency of the host and agent interfaces in your system increases the throughput of your design. Designs with strict cost or power requirements benefit from increasing the transfer efficiency because you can then use less expensive, lower frequency devices. Designs requiring high performance also benefit from increased transfer efficiency because increased efficiency improves the performance of frequency–limited hardware.

Throughput is the number of symbols (such as bytes) of data that Platform Designer can transfer in a given clock cycle. Read latency is the number of clock cycles between the address and data phase of a transaction. For example, a read latency of two means that the data is valid two cycles after the address is posted. If the host must wait for one request to finish before the next begins, such as with a processor, then the read latency is very important to the overall throughput.

You can measure throughput and latency in simulation by observing the waveforms, or using the verification IP monitors.

Related Information

- Avalon Interface Specifications
- Siemens EDA* AXI Verification IP Suite

4.6.1. Using Pipelined Transfers

Pipelined transfers increase the read efficiency by allowing a host to post multiple reads before data from an earlier read returns. Hosts that support pipelined transfers post transfers continuously, relying on the `readdatavalid` signal to indicate valid data. Agents support pipelined transfers by including the `readdatavalid` signal or operating with a fixed read latency.

AXI managers declare how many outstanding writes and reads it can issue with the `writeIssuingCapability` and `readIssuingCapability` parameters. In the same way, an agent can declare how many reads it can accept with the `readAcceptanceCapability` parameter. AXI managers with a read issuing capability greater than one are pipelined in the same way as Avalon hosts and the `readdatavalid` signal.

4.6.1.1. Using the Maximum Pending Reads Parameter

If you create a custom component with an agent interface supporting variable-latency reads, you must specify the **Maximum Pending Reads** parameter in the Component Editor. Platform Designer uses this parameter to generate the appropriate interconnect and represent the maximum number of read transfers that your pipelined agent component can process. If the number of reads presented to the agent interface exceeds the **Maximum Pending Reads** parameter, then the agent interface must assert `waitrequest`.

Optimizing the value of the **Maximum Pending Reads** parameter requires an understanding of the latencies of your custom components. This parameter should be based on the component's highest read latency for the various logic paths inside the component. For example, if your pipelined component has two modes, one requiring two clock cycles and the other five, set the **Maximum Pending Reads** parameter to 5 to allow your component to pipeline five transfers, and eliminating dead cycles after the initial five-cycle latency.

You can also determine the correct value for the **Maximum Pending Reads** parameter by monitoring the number of reads that are pending during system simulation or while running the hardware. To use this method, set the parameter to a high value and use a host that issues read requests on every clock. You can use a DMA for this task if the data is written to a location that does not frequently assert `waitrequest`. If you implement this method, you can observe your component with a logic analyzer or built-in monitoring hardware.

Choosing the correct value for the **Maximum Pending Reads** parameter of your custom pipelined read component is important. If you underestimate the parameter value, you may cause a host interface to stall with a `waitrequest` until the agent responds to an earlier read request and frees a FIFO position.

The **Maximum Pending Reads** parameter controls the depth of the response FIFO inserted into the interconnect for each host connected to the agent. This FIFO does not use significant hardware resources. Overestimating the **Maximum Pending Reads** parameter results in a slight increase in hardware utilization. For these reasons, if you are not sure of the optimal value, you should overestimate this value.

If your system includes a bridge, you must set the **Maximum Pending Reads** parameter on the bridge as well. To allow maximum throughput, this value should be equal to or greater than the **Maximum Pending Reads** value for the connected agent that has the highest value. You can limit the maximum pending reads of an

agent and reduce the buffer depth by reducing the parameter value on the bridge if the high throughput is not required. If you do not know the **Maximum Pending Reads** value for all the agent components, you can monitor the number of reads that are pending during system simulation while running the hardware. To use this method, set the **Maximum Pending Reads** parameter to a high value and use a host that issues read requests on every clock, such as a DMA. Then, reduce the number of maximum pending reads of the bridge until the bridge reduces the performance of any hosts accessing the bridge.

4.6.2. Arbitration Shares and Bursts

Arbitration shares provide control over the arbitration process. By default, the arbitration algorithm allocates evenly, with all hosts receiving one share.

You can adjust the arbitration process by assigning a larger number of shares to hosts that need greater throughput. The larger the arbitration share, the more transfers are allocated to the host to access an agent. The hosts gets uninterrupted access to the agent for its number of shares when the host is reading or writing.

If a host cannot post a transfer, and other hosts are waiting to gain access to a particular agent, the arbiter grants access to another host. This mechanism prevents a host from wasting arbitration cycles if it cannot post back-to-back transfers. A bursting transaction contains multiple beats (or words) of data, starting from a single address. Bursts allow a host to maintain access to an agent for more than a single word transfer. If a bursting host posts a write transfer with a burst length of eight, it is guaranteed arbitration for eight write cycles.

You can assign arbitration shares to an Avalon memory mapped bursting host and AXI hosts (which are always considered a bursting host). Each share consists of one burst transaction (such as multi cycle write), and allows a host to complete a number of bursts before arbitration switches to the next host.

Related Information

Arbitration on page 264

4.6.2.1. Differences Between Arbitration Shares and Bursts

The following three key characteristics distinguish arbitration shares and bursts:

- Arbitration Lock
- Sequential Addressing
- Burst Adapters

Arbitration Lock

When a host posts a burst transfer, the arbitration is locked for that host; consequently, the bursting host should be capable of sustaining transfers for the duration of the locked period. If, after the fourth write, the host deasserts the write signal (Avalon memory mapped write or AXI `wvalid`) for fifty cycles, all other hosts continue to wait for access during this stalled period.

To avoid wasted bandwidth, your host designs should wait until a full burst transfer is ready before requesting access to an agent device. Alternatively, you can avoid wasted bandwidth by posting `burstcounts` equal to the amount of data that is ready. For example, if you create a custom bursting write host with a maximum `burstcount`

of eight, but only three words of data are ready, you can present a `burstcount` of three. This strategy does not result in optimal use of the system band width if the agent is capable of handling a larger burst; however, this strategy prevents stalling and allows access for other hosts in the system.

Sequential Addressing

An Avalon memory mapped burst transfer includes a base address and a `burstcount`, which represents the number of words of data that are transferred, starting from the base address and incrementing sequentially. Burst transfers are common for processors, DMAs, and buffer processing accelerators; however, sometimes a host must access non-sequential addresses. Consequently, a bursting host must set the `burstcount` to the number of sequential addresses, and then reset the `burstcount` for the next location.

The arbitration share algorithm has no restrictions on addresses; therefore, your custom host can update the address it presents to the interconnect for every read or write transaction.

Burst Adapters

Platform Designer allows you to create systems that mix bursting and non-bursting host and agent interfaces. This design strategy allows you to connect bursting host and agent interfaces that support different maximum burst lengths, with Platform Designer generating burst adapters when appropriate.

Platform Designer inserts a burst adapter whenever a host interface burst length exceeds the burst length of the agent interface, or if the host issues a burst type that the agent cannot support. For example, if you connect an AXI manager to an Avalon agent, a burst adapter is inserted. Platform Designer assigns non-bursting hosts and agent interfaces a burst length of one. The burst adapter divides long bursts into shorter bursts. As a result, the burst adapter adds logic to the address and `burstcount` paths between the host and agent interfaces.

4.6.2.2. Choosing Avalon Memory Mapped Interface Types

To avoid inefficient Avalon memory mapped transfers, custom host or agent interfaces must use the appropriate simple, pipelined, or burst interfaces.

4.6.2.2.1. Simple Avalon Memory Mapped Interfaces

Simple interface transfers do not support pipelining or bursting for reads or writes; consequently, their performance is limited. Simple interfaces are appropriate for transfers between hosts and infrequently used agent interfaces. In Platform Designer, the PIO, UART, and Timer include agent interfaces that use simple transfers.

4.6.2.2.2. Pipelined Avalon Memory Mapped Interfaces

Pipelined read transfers allow a pipelined host interface to start multiple read transfers in succession without waiting for prior transfers to complete. Pipelined transfers allow host-agent pairs to achieve higher throughput, even though the agent port may require one or more cycles of latency to return data for each transfer.

In many systems, read throughput becomes inadequate if simple reads are used and pipelined transfers can increase throughput. If you define a component with a fixed read latency, Platform Designer automatically provides the pipelining logic necessary

to support pipelined reads. You can use fixed latency pipelining as the default design starting point for agent interfaces. If your agent interface has a variable latency response time, use the `readdatavalid` signal to indicate when valid data is available. The interconnect implements read response FIFO buffering to handle the maximum number of pending read requests.

To use components that support pipelined read transfers, and to use a pipelined system interconnect efficiently, your system must contain pipelined hosts. You can use pipelined hosts as the default starting point for new host components. Use the `readdatavalid` signal for these host interfaces.

Because host and agents sometimes have mismatched pipeline latency, the interconnect contains logic to reconcile the differences.

Table 43. Pipeline Latency in a Host-Agent Pair

Host	Agent	Pipeline Management Logic Structure
No pipeline	No pipeline	Platform Designer interconnect does not instantiate logic to handle pipeline latency.
No pipeline	Pipelined with fixed or variable latency	Platform Designer interconnect forces the host to wait through any agent-side latency cycles. This host-agent pair gains no benefits from pipelining, because the host waits for each transfer to complete before beginning a new transfer. However, while the host is waiting, the agent can accept transfers from a different host.
Pipelined	No pipeline	Platform Designer interconnect carries out the transfer as if neither host nor agent were pipelined, causing the host to wait until the agent returns data. An example of a non-pipeline agent is an asynchronous off-chip interface.
Pipelined	Pipelined with fixed latency	Platform Designer interconnect allows the host to capture data at the exact clock cycle when data from the agent is valid, to enable maximum throughput. An example of a fixed latency agent is an on-chip memory.
Pipelined	Pipelined with variable latency	The agent asserts a signal when its `readdata` is valid, and the host captures the data. The host-agent pair can achieve maximum throughput if the agent has variable latency. Examples of variable latency agents include SDRAM and FIFO memories.

4.6.2.2.3. Burst Avalon Memory Mapped Interfaces

Burst transfers are commonly used for latent memories such as SDRAM and off-chip communication interfaces, such as PCI Express. To use a burst-capable agent interface efficiently, you must connect to a bursting host. Components that require bursting to operate efficiently typically have an overhead penalty associated with short bursts or non-bursting transfers.

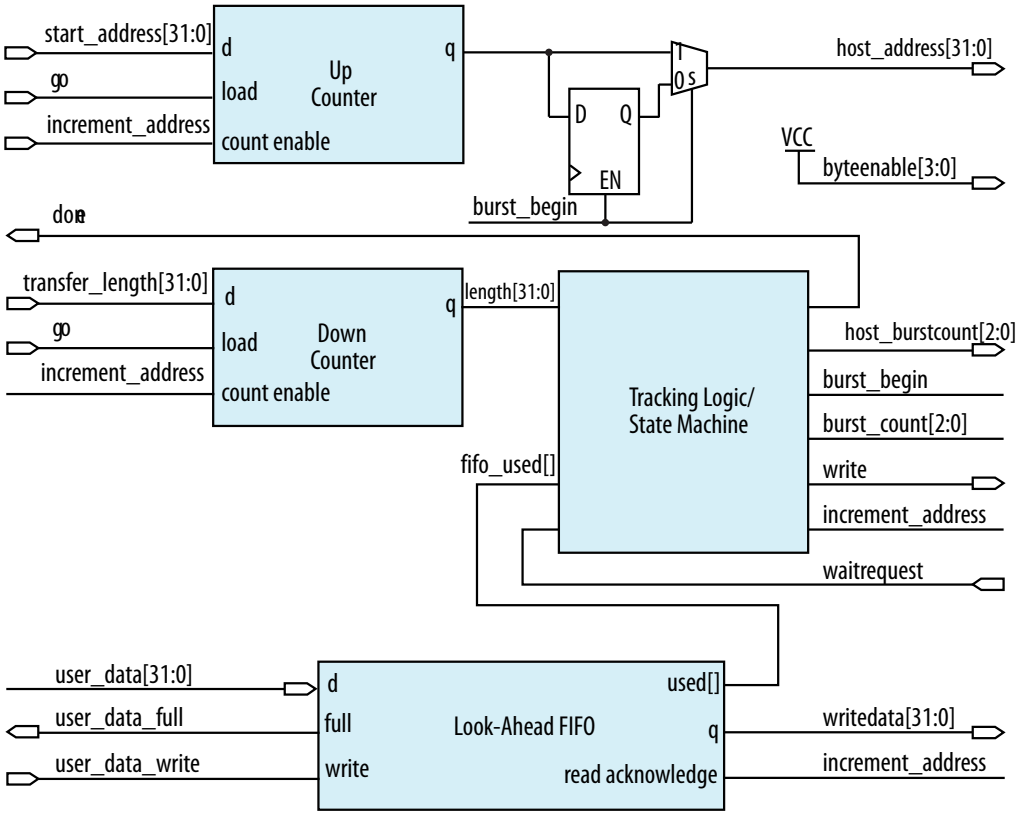
You can use a burst-capable agent interface if you know that your component requires sequential transfers to operate efficiently. Because SDRAM memories incur a penalty when switching banks or rows, performance improves when SDRAM memories are accessed sequentially with bursts.

Architectures that use the same signals to transfer address and data also benefit from bursting. Whenever an address is transferred over shared address and data signals, the throughput of the data transfer is reduced. Because the address phase adds overhead, using large bursts increases the throughput of the connection.

4.6.2.3. Avalon Memory Mapped Burst Host Example

Figure 176. Avalon Bursting Write Host

This example shows the architecture of a bursting write host that receives data from a FIFO and writes the contents to memory. You can use a bursting host as a starting point for your own bursting components, such as custom DMAs, hardware accelerators, or off-chip communication interfaces.



The host performs word accesses and writes to sequential memory locations. When `go` is asserted, the `start_address` and `transfer_length` are registered. On the next clock cycle, the control logic asserts `burst_begin`, which synchronizes the internal control signals in addition to the `host_address` and `host_burstcount` presented to the interconnect. The timing of these two signals is important because during bursting write transfers `byteenable` and `burstcount` must be held constant for the entire burst.

To avoid inefficient writes, the host posts a burst when enough data is buffered in the FIFO. To maximize the burst efficiency, the host should stall only when an agent asserts `waitrequest`. In this example, the FIFO's used signal tracks the number of words of data that are stored in the FIFO and determines when enough data has been buffered.

The `address` register increments after every word transfer, and the `length` register decrements after every word transfer. The address remains constant throughout the burst. Because a transfer is not guaranteed to complete on burst boundaries, additional logic is necessary to recognize the completion of short bursts and complete the transfer.

4.7. Reducing Logic Utilization

You can minimize logic size of Platform Designer systems. Typically, there is a trade-off between logic utilization and performance. Reducing logic utilization applies to both Avalon and AXI interfaces.

4.7.1. Minimizing Interconnect Logic to Reduce Logic Unitization

In Platform Designer, changes to the connections between host and agent reduce the amount of interconnect logic required in the system.

Related Information

Limited Concurrency on page 222

4.7.1.1. Creating Dedicated Host and Agent Connections to Minimize Interconnect Logic

You can create a system where a host interface connects to a single agent interface. This configuration eliminates address decoding, arbitration, and return data multiplexing, which simplifies the interconnect. Dedicated host-to-agent connections attain the same clock frequencies as Avalon streaming connections.

Typically, these one-to-one connections include an Avalon memory-mapped bridge or hardware accelerator. For example, if you insert a pipeline bridge between an agent and all other host interfaces, the logic between the bridge host and agent interface is reduced to wires. If a hardware accelerator connects only to a dedicated memory, no system interconnect logic is generated between the host and agent pair.

4.7.1.2. Removing Unnecessary Connections to Minimize Interconnect Logic

The number of connections between host and agent interfaces affects the f_{MAX} of your system. Every host interface that you connect to an agent interface increases the width of the multiplexer width. As a multiplexer width increases, so does the logic depth and width that implements the multiplexer in the FPGA. To improve system performance, connect hosts and agents only when necessary.

When you connect a host interface to many agent interfaces, the multiplexer for the read data signal grows. Avalon typically uses a `readdata` signal. AXI read data signals add a response status and last indicator to the read response channel using `rdata`, `rresp`, and `rlast`. Additionally, bridges help control the depth of multiplexers.

Related Information

Implementing Command Pipelining (Host-to-Agent) on page 216

4.7.1.3. Simplifying Address Decode Logic

If address code logic is in the critical path, you may be able to change the address map to simplify the decode logic. Experiment with different address maps, including a one-hot encoding, to see if results improve.

4.7.2. Minimizing Arbitration Logic by Consolidating Multiple Interfaces

As the number of components in a design increases, the amount of logic required to implement the interconnect also increases. The number of arbitration blocks increases for every agent interface that is shared by multiple host interfaces. The width of the read data multiplexer increases as the number of agent interfaces supporting read transfers increases on a per host interface basis. For these reasons, consider implementing multiple blocks of logic as a single interface to reduce interconnect logic utilization.

4.7.2.1. Logic Consolidation Trade-Offs

You should consider the following trade-offs before making modifications to your system or interfaces:

- Consider the impact on concurrency that results when you consolidate components. When a system has four host components and four agent interfaces, it can initiate four concurrent accesses. If you consolidate the four agent interfaces into a single interface, then the four hosts must compete for access. Consequently, you should only combine low priority interfaces such as low speed parallel I/O devices if the combination does not impact the performance.
- Determine whether consolidation introduces new decode and multiplexing logic for the agent interface that the interconnect previously included. If an interface contains multiple read and write address locations, the interface already contains the necessary decode and multiplexing logic. When you consolidate interfaces, you typically reuse the decoder and multiplexer blocks already present in one of the original interfaces; however, combining interfaces may simply move the decode and multiplexer logic, rather than eliminate duplication.
- Consider whether consolidating interfaces makes the design complicated. If so, you should not consolidate interfaces.

Related Information

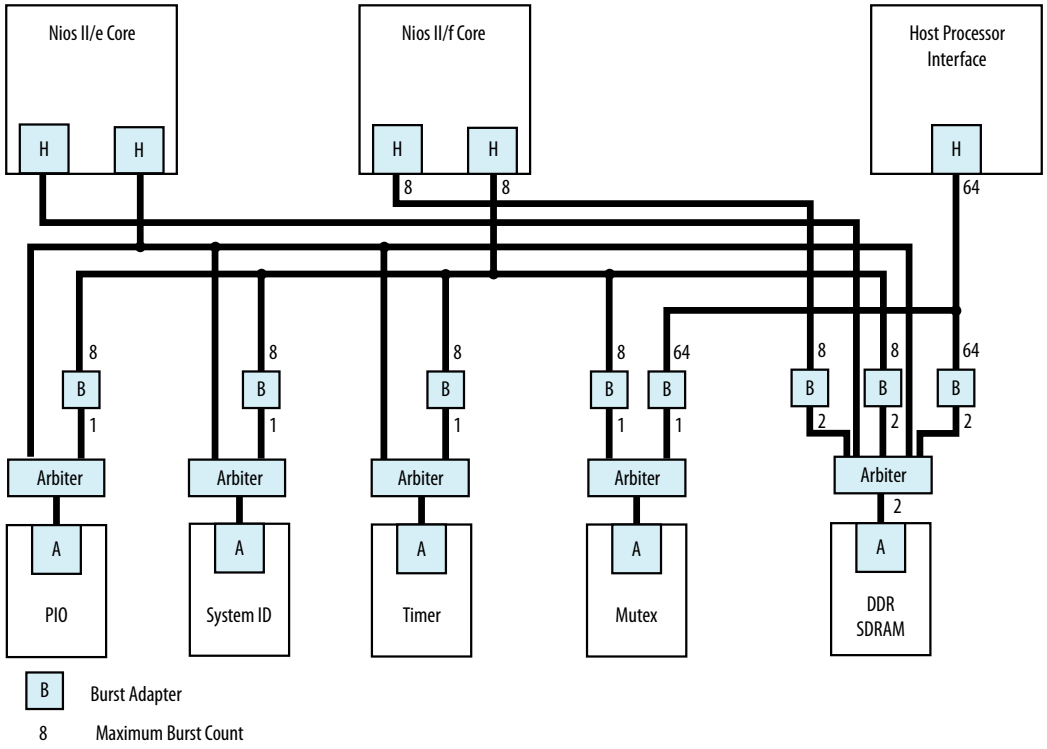
Using Concurrency in Memory-Mapped Systems on page 210

4.7.2.2. Consolidating Interfaces

The following example shows a system with a mix of components, each having different burst capabilities: a Nios II/e core, a Nios II/f core, and an external processor, which off-loads some processing tasks to the Nios II/f core.

The Nios II/f core supports a maximum burst size of eight. The external processor interface supports a maximum burst length of 64. The Nios II/e core does not support bursting. The memory in the system is SDRAM with an Avalon maximum burst length of two.

Figure 177. Mixed Bursting System

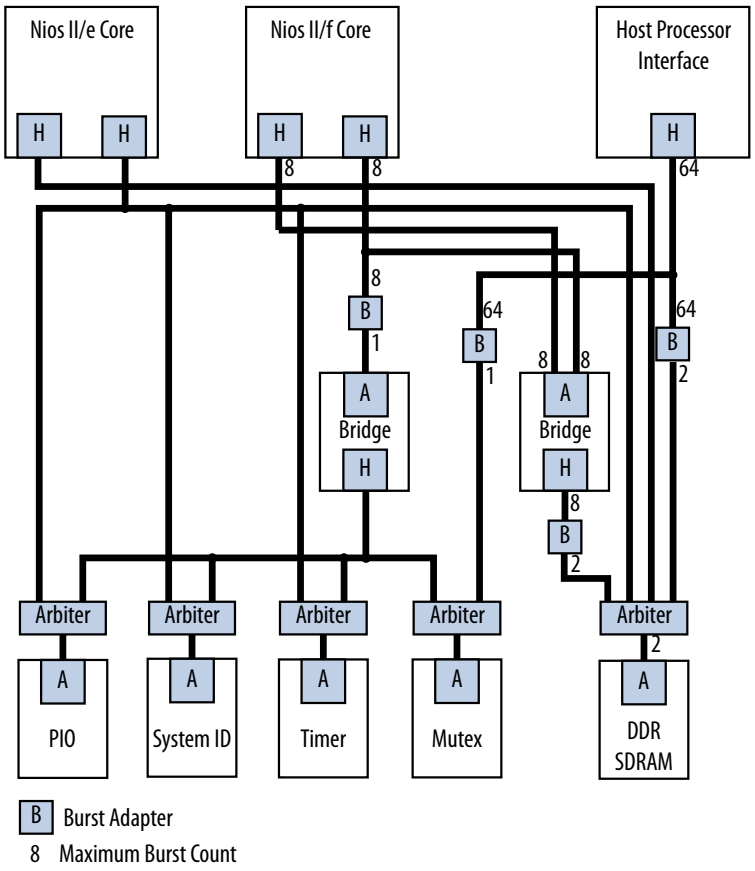


Platform Designer automatically inserts burst adapters to compensate for burst length mismatches. The adapters reduce bursts to a single transfer, or the length of two transfers. For the external processor interface connecting to DDR SDRAM, a burst of 64 words is divided into 32 burst transfers, each with a burst length of two. When you generate a system, Platform Designer inserts burst adapters based on maximum `burstcount` values; consequently, the interconnect logic includes burst adapters between hosts and agent pairs that do not require bursting, if the host is capable of bursts.

In this example, Platform Designer inserts a burst adapter between the Nios II processors and the timer, system ID, and PIO peripherals. These components do not support bursting and the Nios II processor performs a single word read and write accesses to these components.

Figure 178. Mixed Bursting System with Bridges

To reduce the number of adapters, you can add pipeline bridges. The pipeline bridge, between the Nios II/f core and the peripherals that do not support bursts, eliminates three burst adapters from the previous example. A second pipeline bridge between the Nios II/f core and the DDR SDRAM, with its maximum burst size set to eight, eliminates another burst adapter, as shown below.



4.7.3. Reducing Logic Utilization With Multiple Clock Domains

You specify clock domains in Platform Designer on the **System View** tab. Clock sources can be driven by external input signals to Platform Designer, or by PLLs inside Platform Designer. Clock domains are differentiated based on the name of the clock. You can create multiple asynchronous clocks with the same frequency.

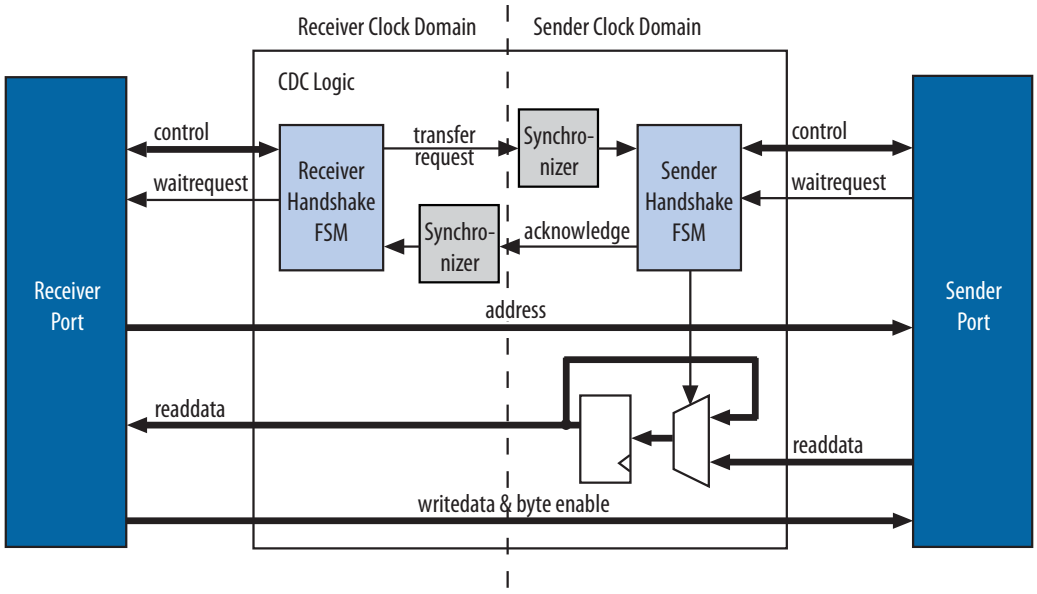
Platform Designer generates Clock Domain Crossing (CDC) logic that hides the details of interfacing components operating in different clock domains. The interconnect supports the memory-mapped protocol with each port independently, and therefore hosts do not need to incorporate clock adapters in order to interface to agents on a different domain. Platform Designer interconnect logic propagates transfers across clock domain boundaries automatically.

Clock-domain adapters provide the following benefits:

- Allows component interfaces to operate at different clock frequencies.
- Eliminates the need to design CDC hardware.
- Allows each memory-mapped port to operate in only one clock domain, which reduces design complexity of components.
- Enables hosts to access any agent without communication with the agent clock domain.
- Allows you to focus performance optimization efforts on components that require fast clock speed.

A clock domain adapter consists of two finite state machines (FSM), one in each clock domain, that use a hand-shaking protocol to propagate transfer control signals (`read_request`, `write_request`, and the host `waitrequest` signals) across the clock boundary.

Figure 179. Clock Crossing Adapter



This example illustrates a clock domain adapter between one host and one agent. The synchronizer blocks use multiple stages of flipflops to eliminate the propagation of meta-stable events on the control signals that enter the handshake FSMs. The CDC logic works with any clock ratio.

The typical sequence of events for a transfer across the CDC logic is as follows:

- The host asserts address, data, and control signals.
- The host handshake FSM captures the control signals and immediately forces the host to wait. The FSM uses only the control signals, not address and data. For example, the host simply holds the address signal constant until the agent side has safely captured it.
- The host handshake FSM initiates a transfer request to the agent handshake FSM.
- The transfer request is synchronized to the agent clock domain.

- The agent handshake FSM processes the request, performing the requested transfer with the agent.
- When the agent transfer completes, the agent handshake FSM sends an acknowledge back to the host handshake FSM. The acknowledge is synchronized back to the host clock domain.
- The host handshake FSM completes the transaction by releasing the host from the wait condition.

Transfers proceed as normal on the agent and the host side, without a special protocol to handle crossing clock domains. From the perspective of an agent, there is nothing different about a transfer initiated by a host in a different clock domain. From the perspective of a host, a transfer across clock domains simply requires extra clock cycles. Similar to other transfer delay cases (for example, arbitration delay or wait states on the agent side), the Platform Designer forces the host to wait until the transfer terminates. As a result, pipeline host ports do not benefit from pipelining when performing transfers to a different clock domain.

Platform Designer automatically determines where to insert CDC logic based on the system and the connections between components, and places CDC logic to maintain the highest transfer rate for all components. Platform Designer evaluates the need for CDC logic for each host and agent pair independently, and generates CDC logic wherever necessary.

4.7.4. Duration of Transfers Crossing Clock Domains

CDC logic extends the duration of host transfers across clock domain boundaries. In the worst case, which is for reads, each transfer is extended by five host clock cycles and five agent clock cycles. Assuming the default value of 2 for the host domain synchronizer length and the agent domain synchronizer length, the components of this delay are the following:

- Four additional host clock cycles, due to the host-side clock synchronizer.
- Four additional agent clock cycles, due to the agent-side clock synchronizer.
- One additional clock in each direction, due to potential metastable events as the control signals cross clock domains.

Note: Systems that require a higher performance clock should use the Avalon memory mapped clock crossing bridge instead of the automatically inserted CDC logic. The clock crossing bridge includes a buffering mechanism so that multiple reads and writes can be pipelined. After paying the initial penalty for the first read or write, there is no additional latency penalty for pending reads and writes, increasing throughput by up to four times, at the expense of added logic resources.

4.8. Reducing Power Consumption

Platform Designer provides various low power design changes that enable you to reduce the power consumption of the interconnect and custom components.

4.8.1. Reducing Power Consumption With Multiple Clock Domains

When you use multiple clock domains, you should put non-critical logic in the slower clock domain. Platform Designer automatically reconciles data crossing over asynchronous clock domains by inserting clock crossing logic (handshake or FIFO).

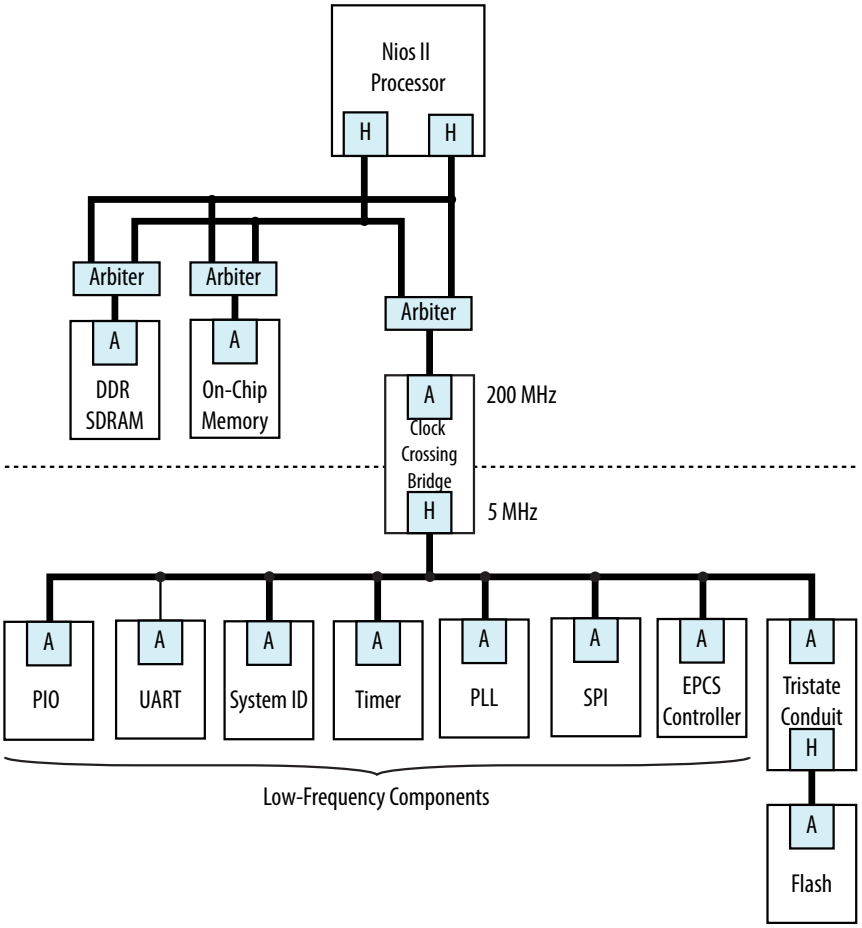
You can use clock crossing in Platform Designer to reduce the clock frequency of the logic that does not require a high frequency clock, which allows you to reduce power consumption. You can use either handshaking clock crossing bridges or handshaking clock crossing adapters to separate clock domains.

You can use the clock crossing bridge to connect host interfaces operating at a higher frequency to agent interfaces running at a lower frequency. Only connect low throughput or low priority components to a clock crossing bridge that operates at a reduced clock frequency. The following are examples of low throughput or low priority components:

- PIOs
- UARTs (JTAG or RS-232)
- System identification (SysID)
- Timers
- PLL (instantiated within Platform Designer)
- Serial peripheral interface (SPI)
- EPCS controller
- Tristate bridge and the components connected to the bridge

By reducing the clock frequency of the components connected to the bridge, you reduce the dynamic power consumption of the design. Dynamic power is a function of toggle rates and decreasing the clock frequency decreases the toggle rate.

Figure 180. Reducing Power Utilization Using a Bridge to Separate Clock Domains



Platform Designer automatically inserts clock crossing adapters between host and agent interfaces that operate at different clock frequencies. You can choose the type of clock crossing adapter in the Platform Designer **Project Settings** tab. Adapters do not appear in the **Connections** column because you do not insert them. The following clock crossing adapter types are available in Platform Designer:

- **Handshake**—Uses a simple handshaking protocol to propagate transfer control signals and responses across the clock boundary. This adapter uses fewer hardware resources because each transfer is safely propagated to the target domain before the next transfer begins. The Handshake adapter is appropriate for systems with low throughput requirements.
- **FIFO**—Uses dual-clock FIFOs for synchronization. The latency of the FIFO adapter is approximately two clock cycles more than the handshake clock crossing component, but the FIFO-based adapter can sustain higher throughput because it supports multiple transactions simultaneously. The FIFO adapter requires more resources, and is appropriate for memory-mapped transfers requiring high throughput across clock domains.
- **Auto**—Platform Designer specifies the appropriate FIFO adapter for bursting links and the Handshake adapter for all other links.

Because the clock crossing bridge uses FIFOs to implement the clock crossing logic, it buffers transfers and data. Clock crossing adapters are not pipelined, so that each transaction is blocking until the transaction completes. Blocking transactions may lower the throughput substantially; consequently, if you want to reduce power consumption without limiting the throughput significantly, you should use the clock crossing bridge or the FIFO clock crossing adapter. However, if the design requires single read transfers, a clock crossing adapter is preferable because the latency is lower.

The clock crossing bridge requires few logic resources other than on-chip memory. The number of on-chip memory blocks used is proportional to the address span, data width, buffering depth, and bursting capabilities of the bridge. The clock crossing adapter does not use on-chip memory and requires a moderate number of logic resources. The address span, data width, and the bursting capabilities of the clock crossing adapter determine the resource utilization of the device.

When you decide to use a clock crossing bridge or clock crossing adapter, you must consider the effects of throughput and memory utilization in the design. If on-chip memory resources are limited, you may be forced to choose the clock crossing adapter. Using the clock crossing bridge to reduce the power of a single component may not justify using more resources. However, if you can place all of the low priority components behind a single clock crossing bridge, you may reduce power consumption in the design.

4.8.2. Reducing Power Consumption by Minimizing Toggle Rates

A Platform Designer system consumes power whenever logic transitions between on and off states. When the state is held constant between clock edges, no charging or discharging occurs. You can use the following design methodologies to reduce the toggle rates of your design:

- Registering component boundaries
- Using clock enable signals
- Inserting bridges

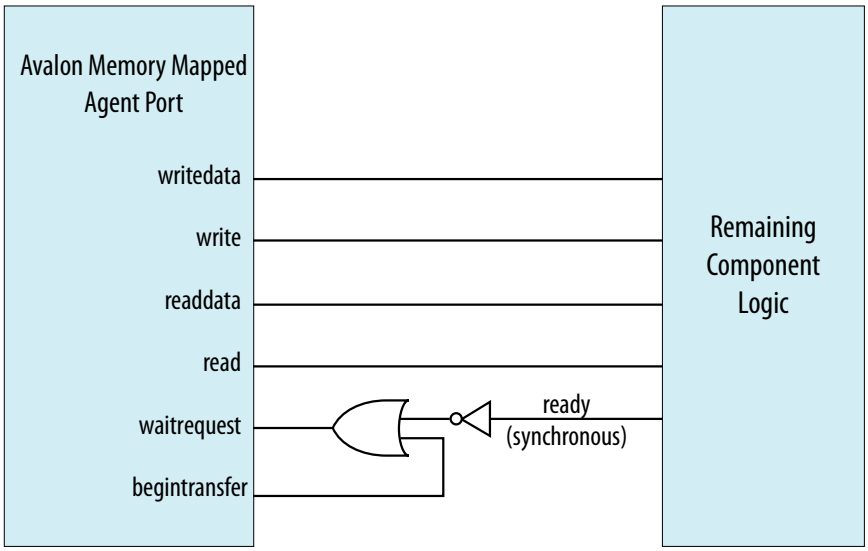
Platform Designer interconnect is uniquely combinational when no adapters or bridges are present and there is no interconnect pipelining. When an agent interface is not selected by a host, various signals may toggle and propagate into the component. By registering the boundary of your component at the host or agent interface, you can minimize the toggling of the interconnect and your component. In addition, registering boundaries can improve operating frequency. When you register the signals at the interface level, you must ensure that the component continues to operate within the interface standard specification.

Avalon memory mapped `waitrequest` is a difficult signal to synchronize when you add registers to your component. The `waitrequest` signal must be asserted during the same clock cycle that a host asserts read or write to in order to prolong the transfer. A host interface can read the `waitrequest` signal too early and post more reads and writes prematurely.

Note: There is no direct AXI equivalent for `waitrequest` and `burstcount`, though the *AMBA Protocol Specification* implies that the AXI `ready` signal cannot depend combinationally on the AXI `valid` signal. Therefore, Platform Designer typically buffers AXI component boundaries for the `ready` signal.

For agent interfaces, the interconnect manages the `begintransfer` signal, which is asserted during the first clock cycle of any read or write transfer. If the `waitrequest` is one clock cycle late, you can logically OR the `waitrequest` and the `begintransfer` signals to form a new `waitrequest` signal that is properly synchronized. Alternatively, the component can assert `waitrequest` before it is selected, guaranteeing that the `waitrequest` is already asserted during the first clock cycle of a transfer.

Figure 181. Variable Latency



Using Clock Enables

You can use clock enables to hold the logic in a steady state, and the `write` and `read` signals as clock enables for agent components. Even if you add registers to your component boundaries, the interface can potentially toggle without the use of clock enables. You can also use the clock enable to disable combinational portions of the component.

For example, you can use an active high clock enable to mask the inputs into the combinational logic to prevent it from toggling when the component is inactive. Before preventing inactive logic from toggling, you must determine if the masking causes the circuit to function differently. If masking causes a functional failure, it may be possible to use a register stage to hold the combinational logic constant between clock cycles.

Inserting Bridges

You can use bridges to reduce toggle rates, if you do not want to modify the component by using boundary registers or clock enables. A bridge acts as a repeater where transfers to the agent interface are repeated on the host interface. If the bridge is not accessed, the components connected to its host interface are also not accessed. The host interface of the bridge remains idle until a host accesses the bridge agent interface.

Bridges can also reduce the toggle rates of signals that are inputs to other host interfaces. These signals are typically `readdata`, `readdatavalid`, and `waitrequest`. Subordinate interfaces that support read accesses drive the

`readdata`, `readdatavalid`, and `waitrequest` signals. A bridge inserts either a register or clock crossing FIFO between the agent interface and the host to reduce the toggle rate of the host input signals.

4.8.3. Reducing Power Consumption by Disabling Logic

There are typically two types of low power modes: volatile and non-volatile. A volatile low power mode holds the component in a reset state. When the logic is reactivated, the previous operational state is lost. A non-volatile low power mode restores the previous operational state. You can use either software-controlled or hardware-controlled sleep modes to disable a component in order to reduce power consumption.

Software-Controlled Sleep Mode

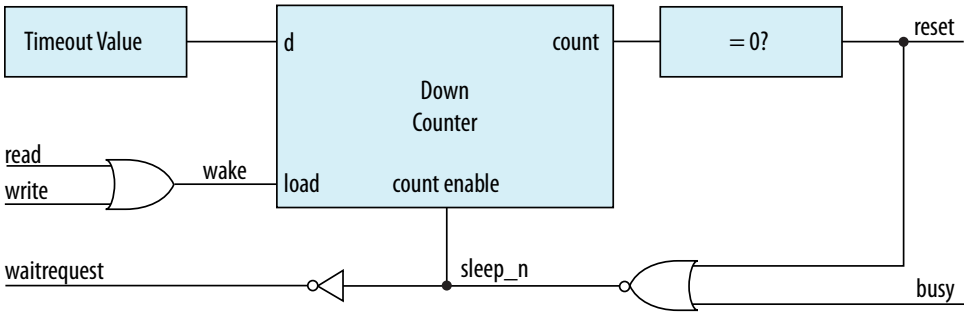
To design a component that supports software-controlled sleep mode, create a single memory-mapped location that enables and disables logic by writing a zero or one. You can use the register's output as a clock enable or reset, depending on whether the component has non-volatile requirements. The agent interface must remain active during sleep mode so that the enable bit is set when the component needs to be activated.

If multiple hosts can access a component that supports sleep mode, you can use the Mutex Intel FPGA IP to provide mutually exclusive accesses to your component. You can also build in the logic to re-enable the component on the very first access by any host in your system. If the component requires multiple clock cycles to re-activate, then it must assert a wait request to prolong the transfer as it exits sleep mode.

Hardware-Controlled Sleep Mode

Alternatively, you can implement a timer in your component that automatically causes the component to enter a sleep mode based on a timeout value specified in clock cycles between read or write accesses. Each access resets the timer to the timeout value. Each cycle with no accesses decrements the timeout value by one. If the counter reaches zero, the hardware enters sleep mode until the next access.

Figure 182. Hardware-Controlled Sleep Components



This example provides a schematic for the hardware-controlled sleep mode. If restoring the component to an active state takes a long time, use a long timeout value so that the component is not continuously entering and exiting sleep mode. The agent interface must remain functional while the rest of the component is in sleep mode. When the component exits sleep mode, the component must assert the `waitrequest` signal until it is ready for read or write accesses.

Related Information

Mutex Core

4.9. Reset Polarity and Synchronization in Platform Designer

When you add a component interface with a reset signal, Platform Designer defines its polarity as `reset`(active-high) or `reset_n` (active-low).

You can view the polarity status of a reset signal by selecting the signal in the **Hierarchy** tab, and then view its expanded definition in the open **Parameters** and **Block Symbol** tabs. When you generate your component, Platform Designer interconnect automatically inverts polarities as needed.

Figure 183. Reset Signal (Active-High)

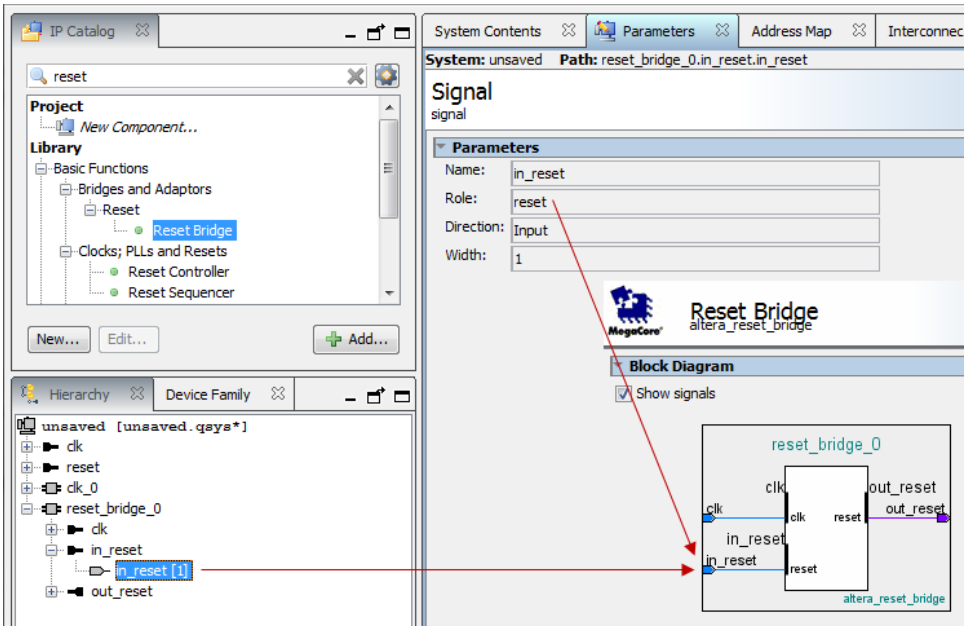
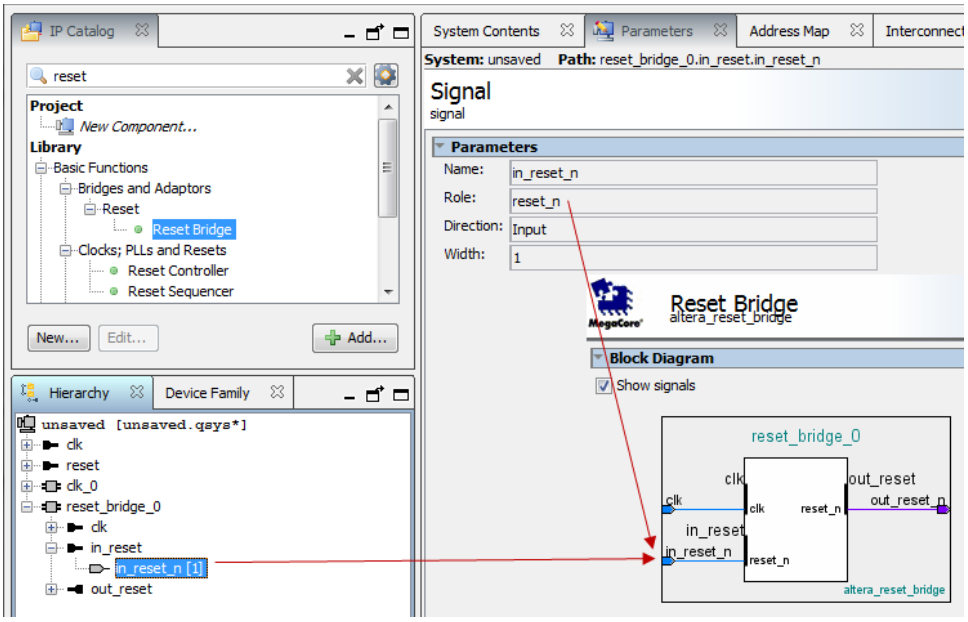


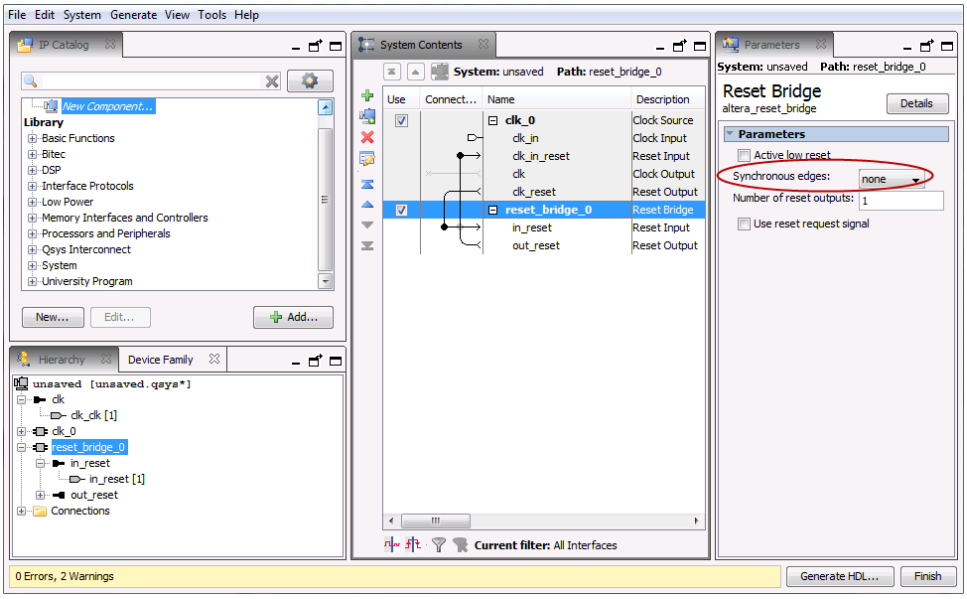
Figure 184. Reset Signal Active-Low



Each Platform Designer component has its own requirements for reset synchronization. Some blocks have internal synchronization and have no requirements, whereas other blocks require an externally synchronized reset. You can define how resets are synchronized in your Platform Designer system with the **Synchronous edges** parameter. In the clock source or reset bridge component, set the value of the **Synchronous edges** parameter to one of the following, depending on how the reset is externally synchronized:

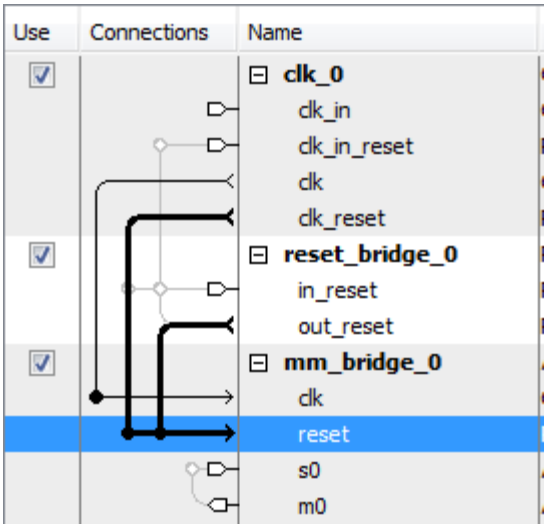
- **None**—There is no synchronization on this reset.
- **Both**—The reset is synchronously asserted and deasserted with respect to the input clock.
- **Deassert**—The reset is synchronously asserted with respect to the input clock, and asynchronously deasserted.

Figure 185. Synchronous Edges Parameter



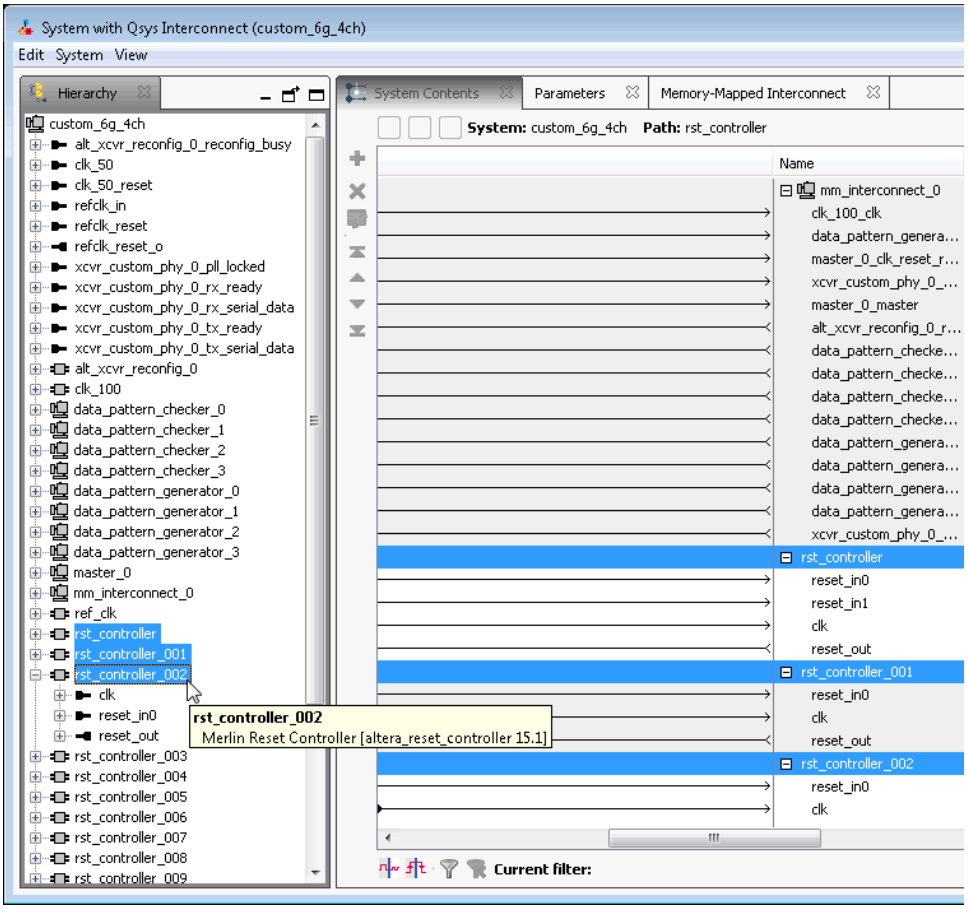
You can combine multiple reset sources to reset a particular component.

Figure 186. Combine Multiple Reset Sources



When you generate your component, Platform Designer inserts adapters to synchronize or invert resets if there are mismatches in polarity or synchronization between the source and destination. You can view inserted adapters on the **Memory-Mapped Interconnect** tab with the **System ➤ Show System with Platform Designer Interconnect** command.

Figure 187. Platform Designer Interconnect



4.10. Optimizing Platform Designer System Performance Design Examples

Avalon Pipelined Read Host Example on page 246

Multiplexer Examples on page 248

4.10.1. Avalon Pipelined Read Host Example

For a high throughput system using the Avalon memory mapped standard, you can design a pipelined read host that allows a system to issue multiple read requests before data returns. Pipelined read hosts hide the latency of read operations by posting reads as frequently as every clock cycle. You can use this type of host when the address logic is not dependent on the data returning.

4.10.1.1. Avalon Pipelined Read Host Example Design Requirements

You must carefully design the logic for the control and datapaths of pipelined read hosts. The control logic must extend a read cycle whenever the `waitrequest` signal is asserted. This logic must also control the host `address`, `byteenable`, and `read`

signals. To achieve maximum throughput, pipelined read hosts should post reads continuously while `waitrequest` is deasserted. While `read` is asserted, the address presented to the interconnect is stored.

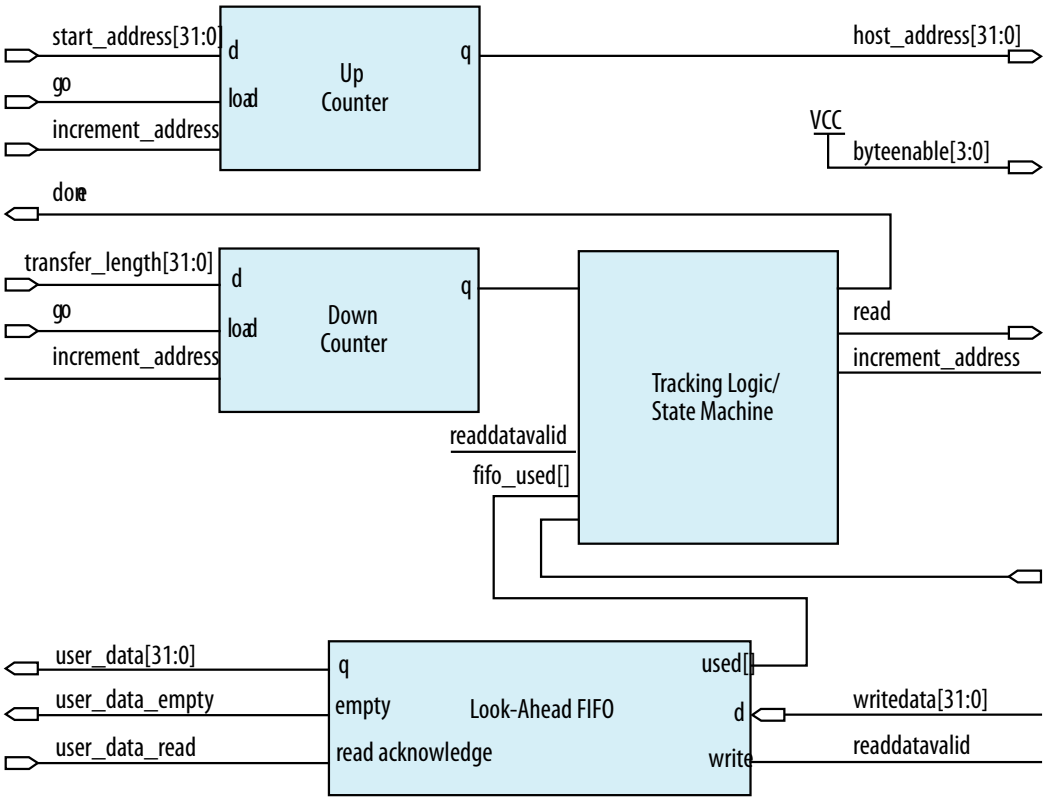
The datapath logic includes the `readdata` and `readdatavalid` signals. If your host can accept data on every clock cycle, you can register the data with the `readdatavalid` as an enable bit. If your host cannot process a continuous stream of read data, it must buffer the data in a FIFO. The control logic must stop issuing reads when the FIFO reaches a predetermined fill level to prevent FIFO overflow.

4.10.1.2. Expected Throughput Improvement

The throughput improvement that you can achieve with a pipelined read host is typically directly proportional to the pipeline depth of the interconnect and the agent interface. For example, if the total latency is two cycles, you can double the throughput by inserting a pipelined read host, assuming the agent interface also supports pipeline transfers. If either the host or agent does not support pipelined read transfers, then the interconnect asserts `waitrequest` until the transfer completes. You can also gain throughput when there are some cycles of overhead before a read response.

Where reads are not pipelined, the throughput is reduced. When both the host and agent interfaces support pipelined read transfers, data flows in a continuous stream after the initial latency. You can use a pipelined read host that stores data in a FIFO to implement a custom DMA, hardware accelerator, or off-chip communication interface.

Figure 188. Pipelined Read Host



This example shows a pipelined read host that stores data in a FIFO. The host performs word accesses that are word-aligned and reads from sequential memory addresses. The transfer length is a multiple of the word size.

When the `go` bit is asserted, the host registers the `start_address` and `transfer_length` signals. The host begins issuing reads continuously on the next clock cycle until the length register reaches zero. In this example, the word size is four bytes so that the address always increments by four, and the length decrements by four. The `read` signal remains asserted unless the FIFO fills to a predetermined level. The address register increments and the length register decrements if the length has not reached 0 and a read is posted.

The host posts a read transfer every time the `read` signal is asserted and the `waitrequest` is deasserted. The host issues reads until the entire buffer has been read or `waitrequest` is asserted. An optional tracking block monitors the `done` bit. When the length register reaches zero, some reads are outstanding. The tracking logic prevents assertion of `done` until the last read completes, and monitors the number of reads posted to the interconnect so that it does not exceed the space remaining in the `readdata` FIFO. This example includes a counter that verifies that the following conditions are met:

- If a read is posted and `readdatavalid` is deasserted, the counter increments.
- If a read is not posted and `readdatavalid` is asserted, the counter decrements.

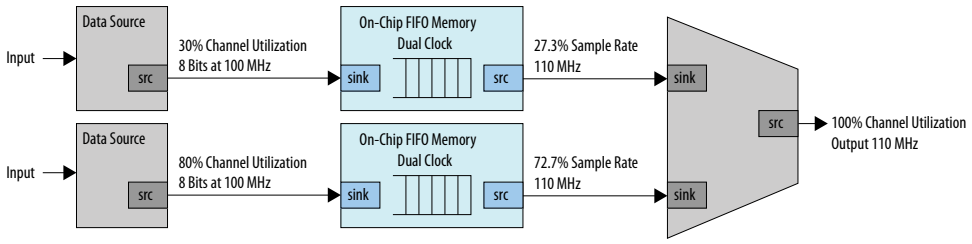
When the `length` register and the tracking logic counter reach zero, all the reads have completed and the `done` bit is asserted. The `done` bit is important if a second host overwrites the memory locations that the pipelined read host accesses. This bit guarantees that the reads have completed before the original data is overwritten.

4.10.2. Multiplexer Examples

You can combine adapters with streaming components to create datapaths whose input and output streams have different properties. The following examples demonstrate datapaths in which the output stream exhibits higher performance than the input stream.

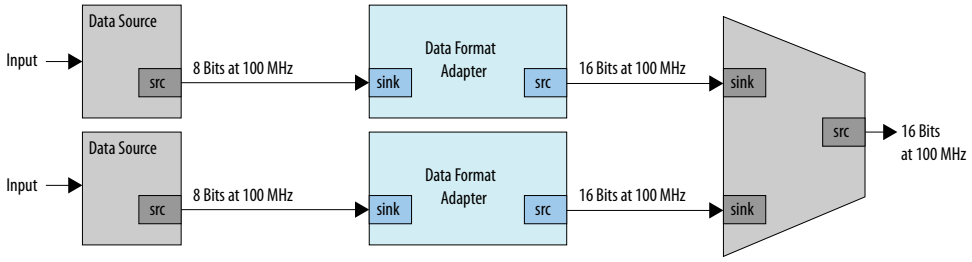
The diagram below illustrates a datapath that uses the dual clock version of the on-chip FIFO memory to boost the frequency of input data from 100 MHz to 110 MHz by sampling two input streams at differential rates. The on-chip FIFO memory has an input clock frequency of 100 MHz, and an output clock frequency of 110 MHz. The channel multiplexer runs at 110 MHz and samples one input stream 27.3 percent of the time, and the second 72.7 percent of the time. You must know what the typical and maximum input channel utilizations are before for this type of design. For example, if the first channel hits 50% utilization, the output stream exceeds 100% utilization.

Figure 189. Datapath that Doubles the Clock Frequency



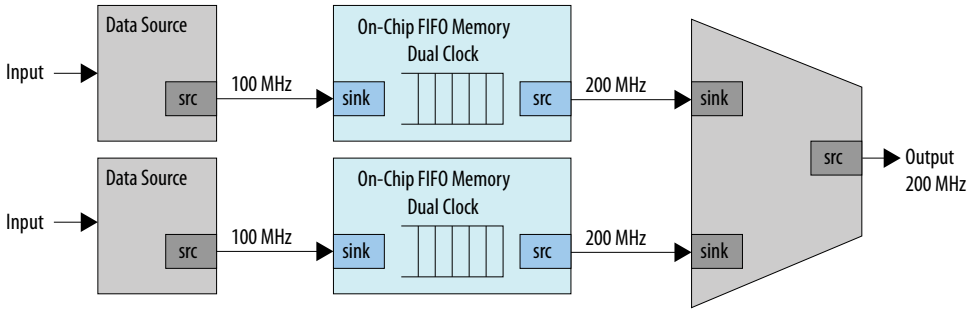
The diagram below illustrates a datapath that uses a data format adapter and Avalon streaming channel multiplexer to merge the 8-bit 100 MHz input from two streaming data sources into a single 16-bit 100 MHz streaming output. This example shows an output with double the throughput of each interface with a corresponding doubling of the data width.

Figure 190. Datapath to Double Data Width and Maintain Original Frequency



The diagram below illustrates a datapath that uses the dual clock version of the on-chip FIFO memory and Avalon streaming channel multiplexer to merge the 100 MHz input from two streaming data sources into a single 200 MHz streaming output. This example shows an output with double the throughput of each interface with a corresponding doubling of the clock frequency.

Figure 191. Datapath to Boost the Clock Frequency



4.11. Optimizing Platform Designer System Performance Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2022.04.02	22.1	• Updated entire chapter for new AXI "manager" and AXI "subordinate" replacement terms. Refer to the AMBA® AXI and ACE Protocol Specification.
2021.03.29	21.1	• Converted to "host" and "agent" inclusive terminology for Avalon memory mapped interface descriptions and related GUI elements throughout.
2018.12.15	18.1.0	• Replaced references to System Contents tab with new System View tab.
2017.11.06	17.1.0	• Changed instances of *Qsys Pro* to *Platform Designer*
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented Qsys rebranding.
2015.11.02	15.1.0	• Added:*Reset Polarity and Synchronization in Qsys*. • Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	*Multiplexer Examples*, rearranged description text for the figures.
May 2013	13.0.0	AMBA APB support.
November 2012	12.1.0	AMBA AXI4 support.
June 2012	12.0.0	AMBA AXI3 support.
November 2011	11.1.0	New document release.

5. Platform Designer Interconnect

Platform Designer interconnect is a high-bandwidth structure that allows you to connect IP components to other IP components with various interfaces.

Platform Designer allows you to establish connections between Avalon and AXI interfaces by generating an interconnect logic. This logic enables you to handle the protocol differences. Platform Designer creates the interconnect logic by converting all the protocols to a proprietary packet format. Then, the tool routes the packet through network switches to the appropriate AXI subordinates or Avalon agents. Here, the packet converts to the subordinate's protocol.(10)

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

The video *AMBA AXI and Intel Avalon Interoperation Using Platform Designer* describes seamless integration of IP components using the AMBA AXI and the Intel Avalon interfaces.

Synchronous Reset Support

Platform Designer interconnect supports synchronous reset of registers in the interconnect. Use of synchronous reset can result in higher performance for Stratix 10 designs because Stratix 10 Hyper-Registers lack a reset signal. If a register in your Stratix 10 design uses asynchronous reset, the Compiler cannot implement the register as a Hyper-Register, potentially reducing performance.

When **Use synchronous reset** is set to **True** in the **Domains** tab, all registers in the interconnect use synchronous reset. The **Use synchronous reset** option is enabled by default for Stratix 10 devices, but is disabled by default for all other devices.

Note: In Platform Designer systems with no clock domain crossing, the initial reset requires asserting for at least 16 cycles. This action prevents the propagation of incorrect values that the reset tree skew may generate during the initial reset release, ensuring the resetting of all the Platform Designer components and interconnect. If system has multiple clocks, reset must be held high for at least 16 slowest clock cycles.

Related Information

- Avalon Interface Specifications
- Creating a System with Platform Designer on page 11
- Creating Platform Designer Components on page 153

(10) This document now refers to the Avalon "host" and "agent," and the AXI "manager" and "subordinate," to replace formerly used terms. Refer to the current AMBA® AXI and ACE Protocol Specification for this latest AMBA AXI and ACE protocol terminology.

- Platform Designer System Design Components on page 367
- AMBA AXI and Intel Avalon Interoperation Using Platform Designer
- Specifying Interconnect Parameters on page 71

5.1. Memory-Mapped Interfaces

Platform Designer supports the implementation of memory-mapped interfaces for Avalon, AXI, and APB protocols.

Platform Designer interconnect transmits memory-mapped transactions between hosts and agents in packets. The command network transports read and write packets from host interfaces to agent interfaces. The response network transports response packets from agent interfaces to host interfaces.

For each component interface, Platform Designer interconnect manages memory-mapped transfers and interacts with signals on the connected interface. Host and agent interfaces can implement different signals based on interface parameterizations, and Platform Designer interconnect provides any necessary adaptation between them. In the path between host and agents, Platform Designer interconnect may introduce registers for timing synchronization, finite state machines for event sequencing, or nothing at all, depending on the services required by the interfaces.

Platform Designer interconnect supports the following implementation scenarios:

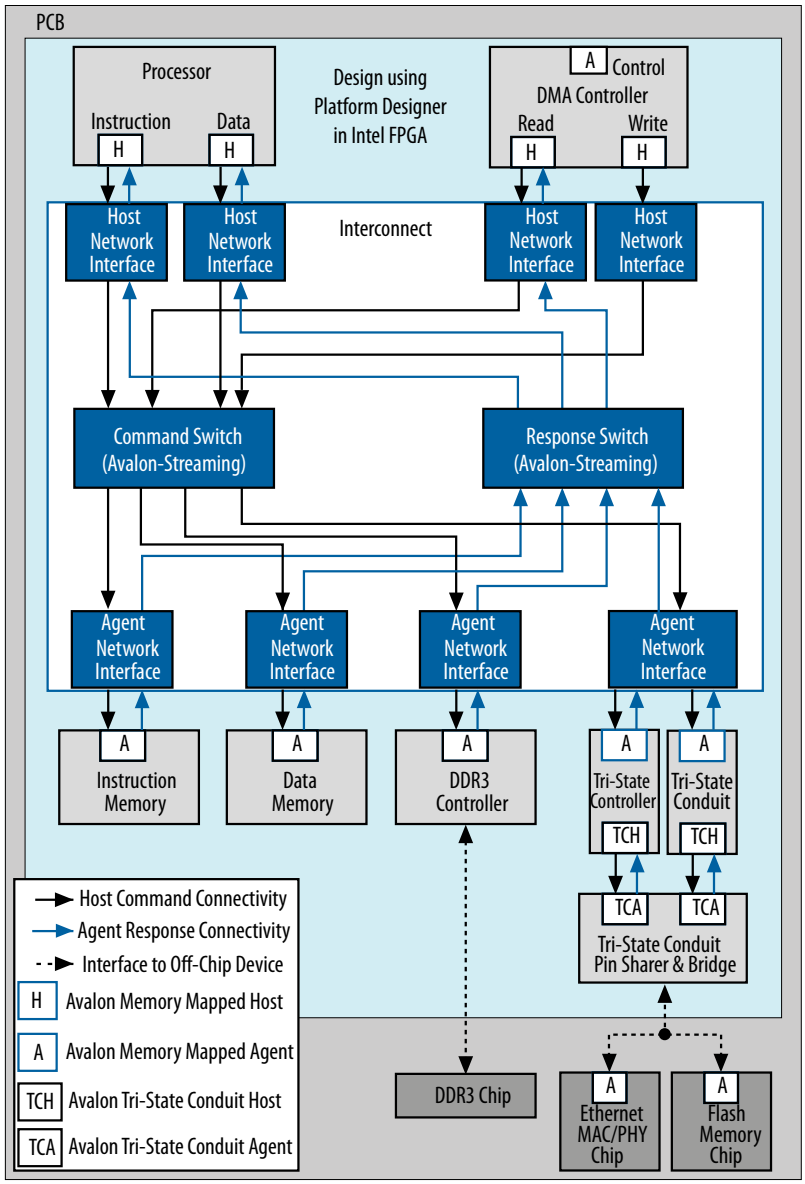
- Any number of components with host and agent interfaces. The host-to-agent relationship can be one-to-one, one-to-many, many-to-one, or many-to-many.
- Hosts and agents of different data widths.
- Hosts and agents operating in different clock domains.
- IP Components with different interface properties and signals. Platform Designer adapts the component interfaces so that interfaces with the following differences can be connected:
  - Avalon and AXI interfaces that use active-high and active-low signaling. AXI signals are active high, except for the reset signal.
  - Interfaces with different burst characteristics.
  - Interfaces with different latencies.
  - Interfaces with different data widths.
  - Interfaces with different optional interface signals.

    Note: Since interface connections between AMBA 3 AXI and AMBA 4 AXI declare a fixed set of signals with variable latency, there is no need for adapting between active-low and active-high signaling, burst characteristics, different latencies, or port signatures. Adaptation might be necessary between Avalon interfaces.

In this example, there are two components hosting the system, a processor and a DMA controller, each with two host interfaces. The hosts connect through the Platform Designer interconnect to agents in the Platform Designer system.

The dark blue blocks represent interconnect components. The dark gray boxes indicate items outside of the Platform Designer system and the Quartus Prime software design, and show how to export component interfaces and how to connect these interfaces to external devices.

Figure 192. Platform Designer interconnect for an Avalon Memory Mapped System with Multiple Hosts



5.1.1. Platform Designer Packet Format

The Platform Designer packet format supports Avalon, AXI, and APB transactions. Memory-mapped transactions between hosts and agents are encapsulated in Platform Designer packets. For Avalon systems without AXI or APB interfaces, some fields are ignored or removed.

5.1.1.1. Fields in the Platform Designer Packet Format

The fields of the Platform Designer packet format are of variable length to minimize resource usage. However, if most components in a design have a single data width, for example 32-bits, and a single component has a data width of 64-bits, Platform Designer inserts a width adapter to accommodate 64-bit transfers.

Table 44. Platform Designer Packet Format for Memory-Mapped Host and Agent Interfaces

Command	Description
`Address`	Specifies the byte address for the lowest byte in the current cycle. There are no restrictions on address alignment.
`Size`	Encodes the run-time size of the transaction. In conjunction with address, this field describes the segment of the payload that contains valid data for a beat within the packet.
`Address Sideband`	Carries "address" sideband signals. The interconnect passes this field from host to agent. This field is valid for each beat in a packet, even though it is only produced and consumed by an address cycle. Up to 8-bit sideband signals are supported for both read and write address channels.
`Cache`	Carries the AXI cache signals.
`Transaction (Exclusive)`	Indicates whether the transaction has exclusive access.
`Transaction (Posted)`	Used to indicate non-posted writes (writes that require responses).
`Data`	For command packets, carries the data to be written. For read response packets, carries the data that has been read.
`Byteenable`	Specifies which symbols are valid. AXI can issue or accept any byteenable pattern. For compatibility with Avalon, Intel recommends that you use the following legal values for 32-bit data transactions between Avalon hosts and agents: • **1111**—Writes full 32 bits • **0011**—Writes lower 2 bytes • **1100**—Writes upper 2 bytes • **0001**—Writes byte 0 only • **0010**—Writes byte 1 only • **0100**—Writes byte 2 only • **1000**—Writes byte 3 only
`Source_ID`	The ID of the host or agent that initiated the command or response.
`Destination_ID`	The ID of the host or agent to which the command or response is directed.
`Response`	Carries the AXI response signals.
`Thread ID`	Carries the AXI transaction ID values.
`Byte count`	The number of bytes remaining in the transaction, including this beat. Number of bytes requested by the packet.

continued...

Command	Description
`Burstwrap`	The burstwrap value specifies the wrapping behavior of the current burst. The burstwrap value is of the form $2^{<n>}$ -1. The following types are defined: • Variable wrap–Variable wrap bursts can wrap at any integer power of 2 value. When the burst reaches the wrap boundary, it wraps back to the previous burst boundary so that only the low order bits are used for addressing. For example, a burst starting at address 0x1C, with a burst wrap boundary of 32 bytes and a burst size of 20 bytes, would write to addresses 0x1C, 0x0, 0x4, 0x8, and 0xC. • For a burst wrap boundary of size <*m*>, `Burstwrap` = <*m*> - 1, or for this case `Burstwrap` = (32 - 1) = 31 which is 2^5 -1. • For AXI managers, the burstwrap boundary value (m) is based on the different AXBURST: — Burstwrap set to all 1's. For example, for a 6-bit burstwrap, burstwrap is `6'b111111`. — For `WRAP` bursts, burstwrap = AXLEN * size – 1. — For `FIXED` bursts, burstwrap = size – 1. — Sequential bursts increment the address for each transfer in the burst. For sequential bursts, the `Burstwrap` field is set to all 1s. For example, with a 6-bit `Burstwrap` field, the value for a sequential burst is 6'b111111 or 63, which is 2^6 - 1. For Avalon hosts, Platform Designer adaptation logic sets a hardwired value for the burstwrap field, according to the declared host burst properties. For example, for a host that declares sequential bursting, the burstwrap field is set to ones. Similarly, hosts that declare burst have their burstwrap field set to the appropriate constant value. AXI managers choose their burst type at run-time, depending on the value of the `AW` or ARBURST signal. The interconnect calculates the burstwrap value at run-time for AXI managers.(11)
`Protection`	Access level protection. When the lowest bit is 0, the packet has normal access. When the lowest bit is 1, the packet has privileged access. For Avalon memory mapped interfaces, this field maps directly to the privileged access signal, which allows a memory-mapped host to write to an on-chip memory ROM instance. The other bits in this field support AXI secure accesses and uses the same encoding, as described in the AXI specification.
`QoS`	QoS (Quality of Service Signaling) is a 4-bit field that is part of the AMBA 4 AXI interface that carries QoS information for the packet from the AXI manager to the AXI subordinate. Transactions from AMBA 3 AXI and Avalon hosts have the default value `4'b0000`, that indicates that they are not participating in the QoS scheme. QoS values are dropped for agents that do not support QoS.
`Data sideband`	Carries data sideband signals for the packet. On a write command, the data sideband directly maps to `WUSER`. On a read response, the data sideband directly maps to RUSER. On a write response, the data sideband directly maps to BUSER.

5.1.1.2. Transaction Types for Memory-Mapped Interfaces

Table 45. Transaction Types for Memory-Mapped Interfaces

The table below describes the information that each bit transports in the packet format's transaction field.

Bit	Name	Definition
`0`	`PKT_TRANS_READ`	When asserted, indicates a read transaction.
`1`	`PKT_TRANS_COMPRESSED_READ`	For read transactions, specifies whether the read command can be expressed in a single cycle (all `byteenables` asserted on every cycle).

continued...

(11) This document refers to the new AXI "manager" and AXI "subordinate" inclusive terms to replace outmoded terms, as the latest version of the AMBA® AXI and ACE Protocol Specification describes.

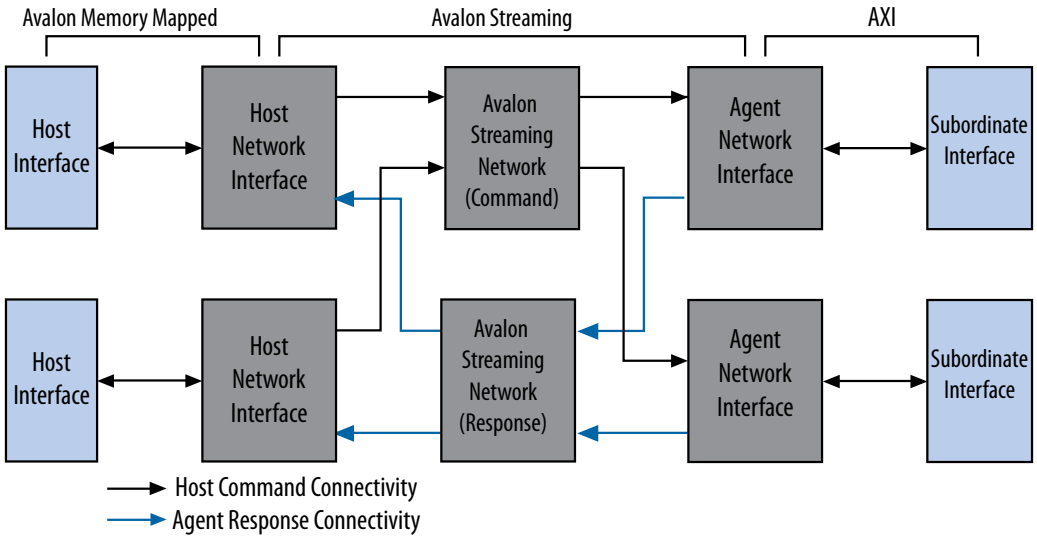
Bit	Name	Definition
2	PKT_TRANS_WRITE	When asserted, indicates a write transaction.
3	PKT_TRANS_POSTED	When asserted, no response is required.
4	PKT_TRANS_LOCK	When asserted, indicates arbitration is locked. Applies to write packets.

5.1.1.3. Platform Designer Transformations

The memory-mapped host and agent components connect to network interface modules that encapsulate the transaction in Avalon streaming packets. The memory-mapped interfaces have no information about the encapsulation or the function of the layer transporting the packets. The interfaces operate in accordance with memory-mapped protocol and use the read and write signals and transfers.

Figure 193. Transformation when Generating a System with Memory-Mapped Components

Platform Designer components that implement the blocks appear shaded.



Related Information

- Avalon Host and AXI Manager Network Interfaces on page 258
- Avalon Agent and AXI Subordinate Network Interfaces on page 261

5.1.2. Interconnect Domains

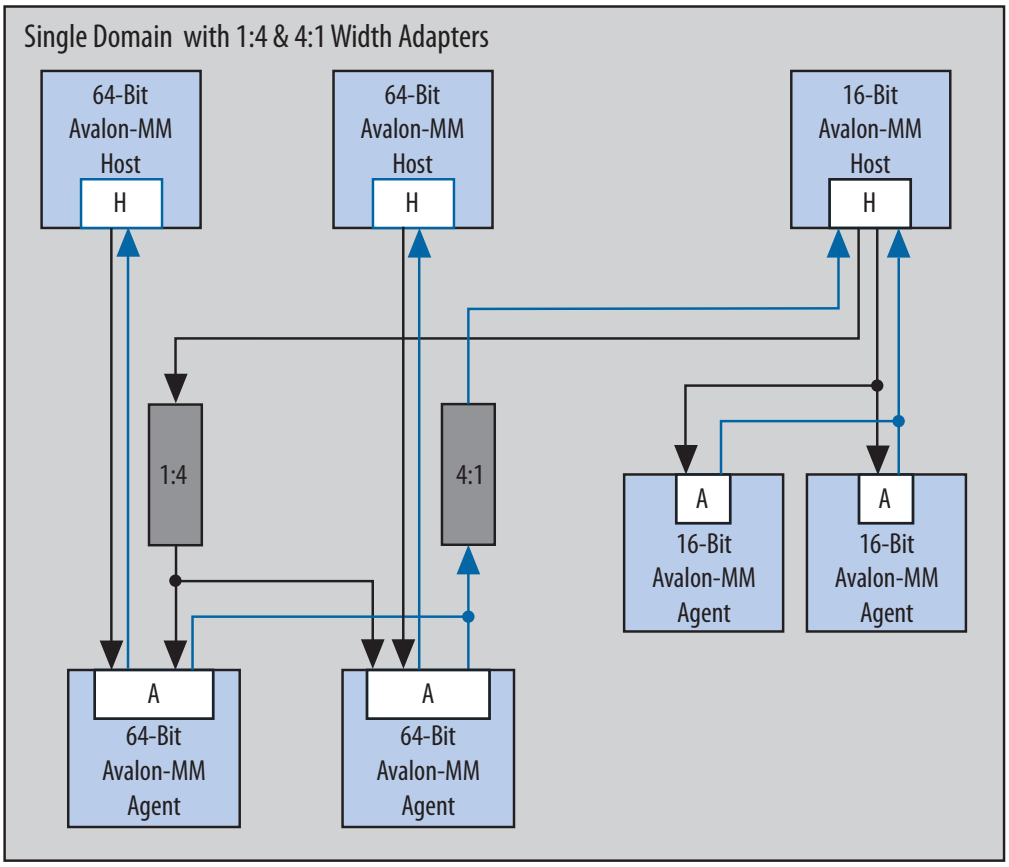
An interconnect domain is a group of connected memory-mapped hosts and agents that share the same interconnect. The components in a single interconnect domain share the same packet format.

5.1.2.1. Using One Domain with Width Adaptation

When one of the hosts in a system connects to all the agents, Platform Designer creates a single domain with two packet formats: one with 64-bit data, and one with 16-bit data. A width adapter manages accesses between the 16-bit host and 64-bit

agents. In the following example, two 64-bit hosts access two 64-bit agents. One 16-bit host, accesses two 16-bit agents and two 64-bit agents. The 16-bit Avalon host connects through a 1:4 adapter, then a 4:1 adapter to reach its 16-bit agents.

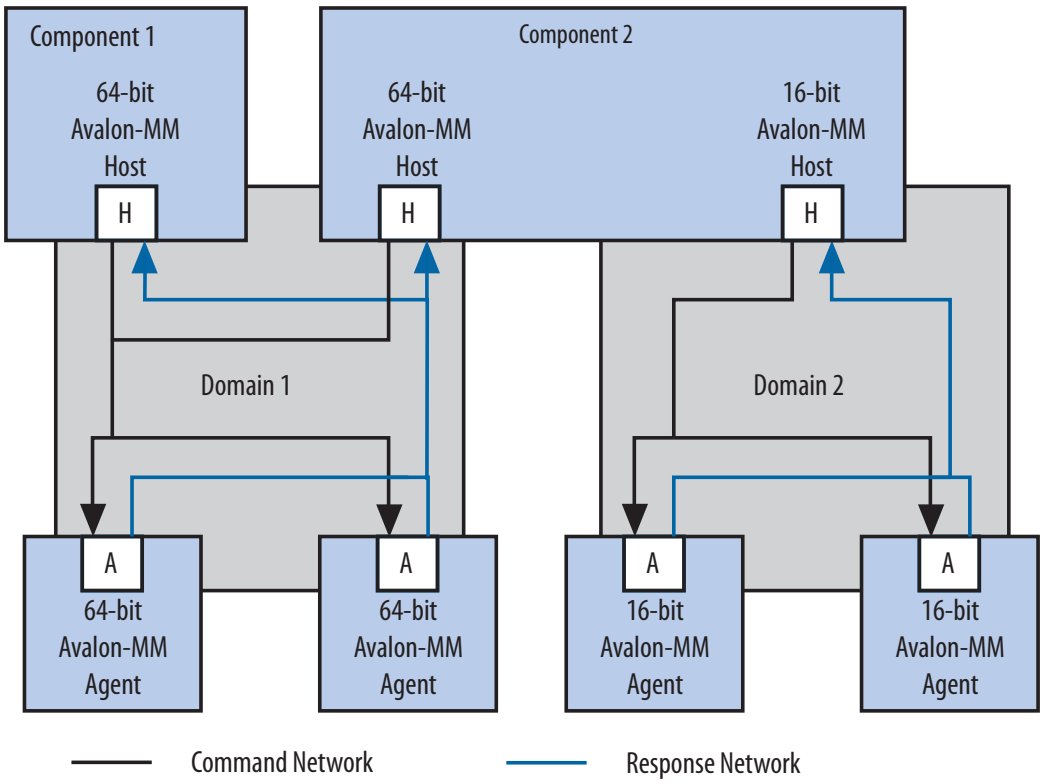
Figure 194. One Domain with 1:4 and 4:1 Width Adapters



5.1.2.2. Using Two Separate Domains

In the following example, Platform Designer uses two separate domains. The first domain includes two 64-bit hosts connected to two 64-bit agents. A second domain includes one 16-bit host connected to two 16-bit agents. Because the interfaces in Domain 1 and Domain 2 do not share any connections, Platform Designer can optimize the packet format for the two separate domains. In this example, the first domain uses a 64-bit data width and the second domain uses 16-bit data.

Figure 195. Two Separate Domains

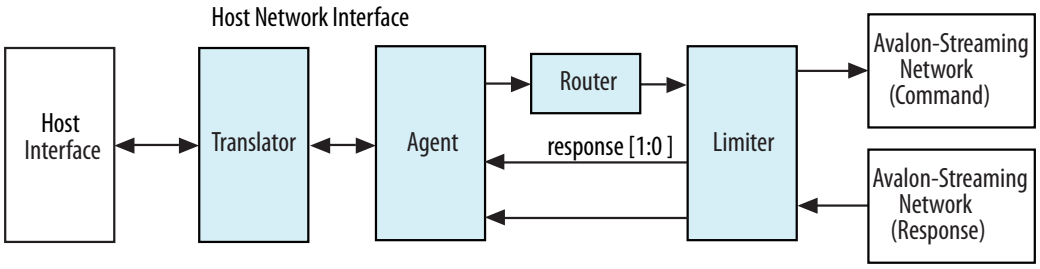


5.1.3. Avalon Host and AXI Manager Network Interfaces

Figure 196. Avalon Memory Mapped Host Network Interface

Avalon network interfaces drive default values for the QoS and BUSER, WUSER, and RUSER packet fields in the host agent, and drop the packet fields in the agent.

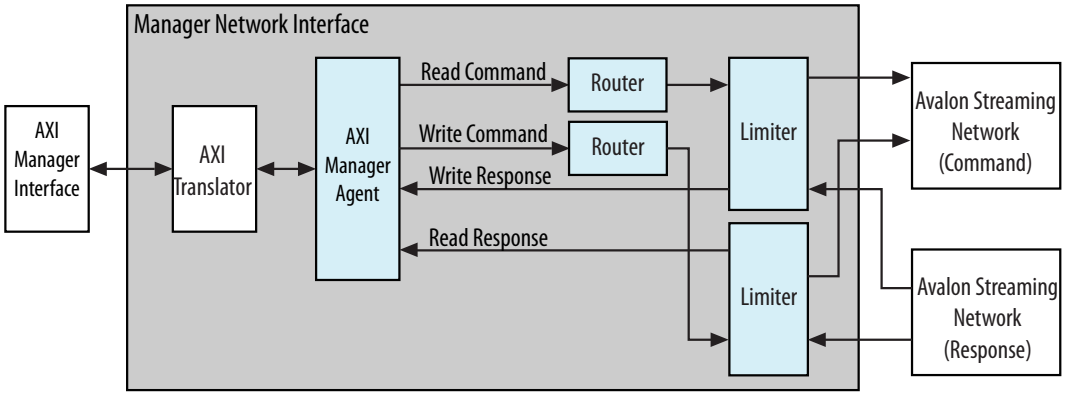
Note: The `response` signal from the Limiter to the Agent is optional.



An AMBA 4 AXI manager supports `INCR` bursts up to 256 beats, QoS signals, and data sideband signals.(12)

(12) This document refers to the new AXI "manager" and AXI "subordinate" inclusive terms to replace outmoded terms, as the latest version of the AMBA® AXI and ACE Protocol Specification describes.

Figure 197. AXI Manager Network Interface



Note: For a complete definition of the optional read response signal, refer to *Avalon Memory-Mapped Interface Signal Types* in the *Avalon Interface Specifications*.

Related Information

- Avalon Interface Specifications
- Creating a System with Platform Designer on page 11

5.1.3.1. Avalon Memory Mapped Host Agent

The Avalon Memory Mapped Host Agent translates Avalon memory mapped host transactions into Platform Designer command packets and translates the Platform Designer Avalon memory mapped agent response packets into Avalon memory mapped responses.

5.1.3.2. Avalon Memory Mapped Host Translator

The Avalon Memory Mapped Host Translator interfaces with an Avalon memory mapped host component and converts the Avalon memory mapped host interface to a simpler representation for use in Platform Designer.

The Avalon Memory Mapped Host translator performs the following functions:

- Translates active-low signaling to active-high signaling
- Inserts wait states to prevent an Avalon memory mapped host from reading invalid data
- Translates word and symbol addresses
- Translates word and symbol burst counts
- Manages re-timing and re-sequencing bursts
- Removes unnecessary address bits

5.1.3.3. AXI Manager Agent

An AXI Manager Agent accepts AXI commands and produces Platform Designer command packets. It also accepts Platform Designer response packets and converts those into AXI responses. This component has separate packet channels for read

commands, write commands, read responses, and write responses. The Avalon host agent drives the QoS and BUSER, WUSER, and RUSER packet fields with default values AXQO and b0000, respectively.

Note: For signal descriptions, refer to *Platform Designer Packet Format*.

Related Information

Fields in the Platform Designer Packet Format on page 254

5.1.3.4. AXI Translator

AMBA 4 AXI allows omitting signals from interfaces. The translator bridges between these "incomplete" AMBA 4 AXI interfaces and the "complete" AMBA 4 AXI interface on the network interfaces.

Attention: If you connect an Avalon agent or to a host, or connect an AXI subordinate to a manager, without response ports, the interconnect could ignore transaction responses, such as SLAVEERROR or DECODEERROR. This situation could lead to returning invalid data to the manager or host.

The AXI translator is inserted for both AXI managers and subordinates and performs the following functions:

- Matches ID widths between the manager and subordinates in 1x1 systems.
- Drives default values as defined in the *AMBA Protocol Specifications* for missing signals.
- Performs lock transaction bit conversion when an AMBA 3 AXI manager connects to an AMBA 4 AXI subordinate in 1x1 systems.

Related Information

Arm AMBA Protocol Specifications

5.1.3.5. APB Manager Agent

An APB manager agent accepts APB commands and produces or generates Platform Designer command packets. It also converts Platform Designer response packets to APB responses.

5.1.3.6. APB Subordinate Agent

An APB subordinate agent issues resulting transaction to the APB interface. It also accepts creates Platform Designer response packets.

5.1.3.7. APB Translator

An APB peripheral does not require pslverr signals to support additional signals for the APB debug interface.

The APB translator is inserted for both the manager and subordinate and performs the following functions:

- Sets the response value default to OKAY if the APB subordinate does not have a `pslverr` signal.
- Turns on or off additional signals between the APB debug interface, which is used with HPS (Intel SoC's Hard Processor System).

5.1.3.8. AHB Subordinate Agent

The Platform Designer interconnect supports non-bursting Advanced High-performance Bus (AHB) subordinate interfaces.

5.1.3.9. Memory-Mapped Router

The Memory-Mapped Router routes command packets from the host to the agent, and response packets from the agent to the host. For host command packets, the router uses the address to set the `Destination_ID` and Avalon streaming channel. For the agent response packet, the router uses the `Destination_ID` to set the Avalon streaming channel. The demultiplexers use the Avalon streaming channel to route the packet to the correct destination.

5.1.3.10. Memory-Mapped Traffic Limiter

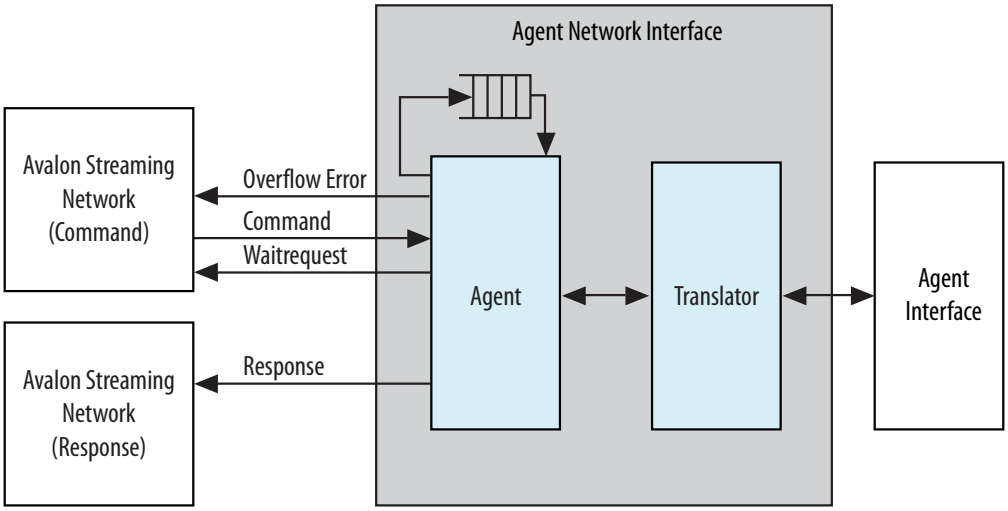
The Memory-Mapped Traffic Limiter ensures the responses arrive in order. It prevents any command from being sent if the response could conflict with the response for a command that has already been issued. By guaranteeing in-order responses, the Traffic Limiter simplifies the response network.

5.1.4. Avalon Agent and AXI Subordinate Network Interfaces

5.1.4.1. Avalon Memory Mapped Agent Translator

The Avalon Memory Mapped Agent Translator converts the Avalon memory mapped agent interface to a simplified representation that the Platform Designer network can use.

Figure 198. Avalon Memory Mapped Agent Network Interface



An Avalon Memory Mapped Agent Translator performs the following functions:

- Drives the `beginbursttransfer` and `byteenable` signals.
- Supports Avalon memory mapped agents that operate using fixed timing and or agents that use the `readdatavalid` signal to identify valid data.
- Translates the `read`, `write`, and `chipselect` signals into the representation that the Avalon streaming agent response network uses.
- Converts active low signals to active high signals.
- Translates word and symbol addresses and burstcounts.
- Handles burstcount timing and sequencing.
- Removes unnecessary address bits.

Related Information

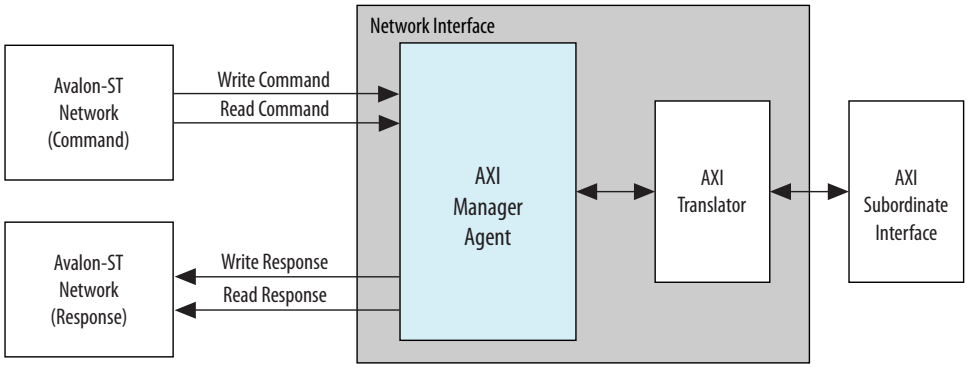
Avalon Agent and AXI Subordinate Network Interfaces on page 261

5.1.4.2. AXI Translator

AMBA 4 AXI allows omitting signals from interfaces. The translator bridges between these "incomplete" AMBA 4 AXI interfaces and the "complete" AMBA 4 AXI interface on the network interfaces.

Figure 199. AXI Subordinate Network Interface

An AMBA 4 AXI subordinate supports up to 256 beat INCR bursts, QoS signals, and data sideband signals.



The AXI translator is inserted for both AMBA 4 AXI manager and subordinate, and performs the following functions:

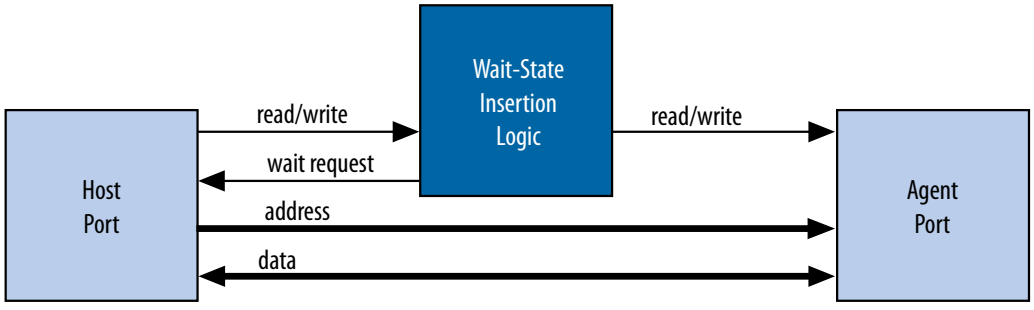
- Matches ID widths between manager and subordinate in 1x1 systems.
- Drives default values as defined in the *AMBA Protocol Specifications* for missing signals.
- Performs lock transaction bit conversion when an AMBA 3 AXI manager connects to an AMBA 4 AXI subordinate in 1x1 systems.

5.1.4.3. Wait State Insertion

Wait states extend the duration of a transfer by one or more cycles. Wait state insertion logic accommodates the timing needs of each agent, and causes the host to wait until the agent can proceed. Platform Designer interconnect inserts wait states into a transfer when the target agent cannot respond in a single clock cycle, as well as in cases when agent `read` and `write` signals have setup or hold time requirements.

Wait state insertion logic is a small finite-state machine that translates control signal sequencing between the agent side and the host side. Platform Designer interconnect can force a host to wait for the wait state needs of an agent; for example, arbitration logic in a multi-host system. Platform Designer generates wait state insertion logic based on the properties of all agents in the system.

Figure 200. Wait State Insertion Logic for One Host and One Agent



5.1.4.4. Avalon Memory Mapped Agent Component

The Avalon Memory Mapped Agent component accepts command packets and issues the resulting transactions to the Avalon interface. For pipelined agents, an Avalon streaming FIFO stores information about pending transactions. The size of this FIFO is the maximum number of pending responses that you specify when creating the agent component. The Avalon Memory Mapped Agent component also `backpressures` the Avalon memory mapped host command interface when the FIFO is full if the agent component includes the `waitrequest` signal.

5.1.4.5. AXI Subordinate Agent

An AXI subordinate Agent works like a reverse AXI manager agent. The AXI subordinate Agent accepts Platform Designer command packets to create AXI commands, and accepts AXI responses to create Platform Designer response packets. This component has separate packet channels for read commands, write commands, read responses, and write responses.

5.1.5. Arbitration

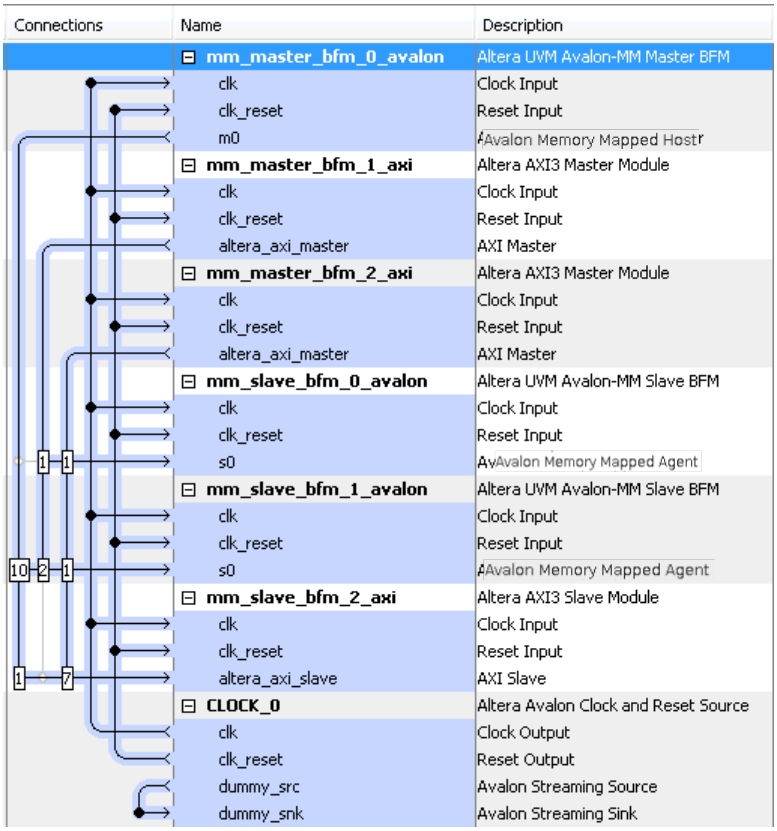
When multiple hosts contend for access to an agent, Platform Designer automatically inserts arbitration logic, which grants access in fairness-based, round-robin order. You can alternatively choose to designate an agent as a fixed priority arbitration agent, and then manually assign priorities in the Platform Designer GUI.

5.1.5.1. Round-Robin Arbitration

When multiple hosts contend for access to an agent, Platform Designer automatically inserts arbitration logic which grants access in fairness-based, round-robin order.

In a fairness-based arbitration protocol, each host has an integer value of transfer *shares* with respect to an agent. One share represents permission to perform one transfer. The default arbitration scheme is equal share round-robin that grants equal, sequential access to all requesting hosts. You can change the arbitration scheme to weighted round-robin by specifying a relative number of arbitration shares to the hosts that access a given agent. AXI subordinates have separate arbitration for their independent read and write channels, and the **Arbitration Shares** setting affects both the read and write arbitration. To display arbitration settings, right-click an instance on the **System View** tab, and then click **Show Arbitration Shares**.

Figure 201. Arbitration Shares in the Connections Column



5.1.5.1.1. Fairness-Based Shares

In a fairness-based arbitration scheme, each host-to-agent connection provides a transfer share count. This count is a request for the arbiter to grant a specific number of transfers to this host before giving control to a different host. One share represents permission to perform one transfer.

Figure 202. Arbitration of Continuous Transfer Requests from Two Hosts

Consider a system with two hosts connected to a single agent. Host 1 has its arbitration shares set to three, and Host 2 has its arbitration shares set to four. Host 1 and Host 2 continuously attempt to perform back-to-back transfers to the agent. The arbiter grants Host 1 access to the agent for three transfers, and then grants Host 2 access to the agent for four transfers. This cycle repeats indefinitely. The figure below describes the waveform for this scenario.

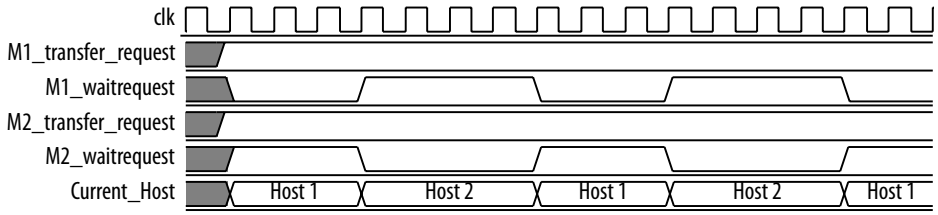
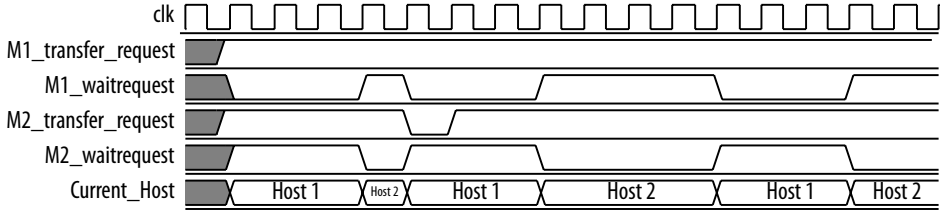


Figure 203. Arbitration of Two Hosts with a Gap in Transfer Requests

If a host stops requesting transfers before it exhausts its shares, it forfeits all its remaining shares, and the arbiter grants access to another requesting host. After completing one transfer, Host 2 stops requesting for one clock cycle. As a result, the arbiter grants access back to Host 1, which gets a replenished supply of shares.



5.1.5.1.2. Round-Robin Scheduling

When multiple hosts contend for access to an agent, the arbiter grants shares in round-robin order. Platform Designer includes only requesting hosts in the arbitration for each agent transaction.

5.1.5.2. Fixed Priority Arbitration

Fixed priority arbitration is an alternative arbitration scheme to the default round-robin scheme.

You can selectively apply fixed priority arbitration to any agent in a Platform Designer system. You can design Platform Designer systems where a subset of agents use the default round-robin arbitration, and other agents use fixed priority arbitration. Fixed priority arbitration uses a fixed priority algorithm to grant access to an agent amongst its connected hosts.

Set a fixed priority agent arbitration under **Interconnect Parameters** in the **Domains** tab. You can then assign an arbitration priority number for each host connected to a fixed priority agent in the **System View** tab, where the highest numeric value receives the highest priority. When multiple hosts request access to a fixed priority arbitrated agent, the arbiter gives the host with the highest priority first access to the agent.

For example, when a fixed priority agent receives requests from three hosts on the same cycle, the arbiter grants the host with highest assigned priority first access to the agent, and backpressures the other two hosts.

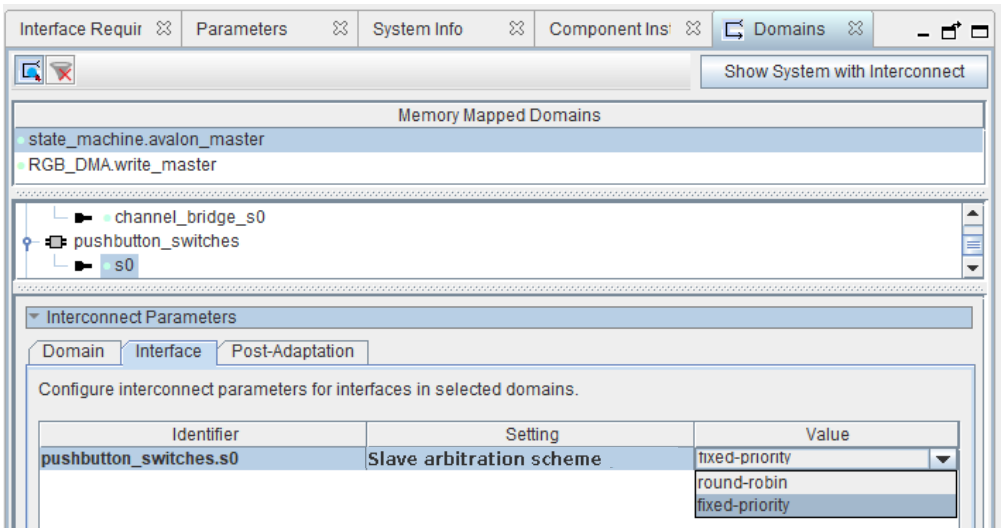
Note: When you connect an AXI manager to an Avalon memory mapped agent designated to use a fixed priority arbitrator, the interconnect instantiates a command-path intermediary round-robin multiplexer in front of the designated agent.

5.1.5.2.1. Designate a Platform Designer Agent to Use Fixed Priority Arbitration

You can designate any agent in your Platform Designer system to use fixed priority arbitration. You must assign each host connected to a fixed priority agent a numeric priority. The host with the highest higher priority receives first access to the agent. No two hosts can have the same priority.

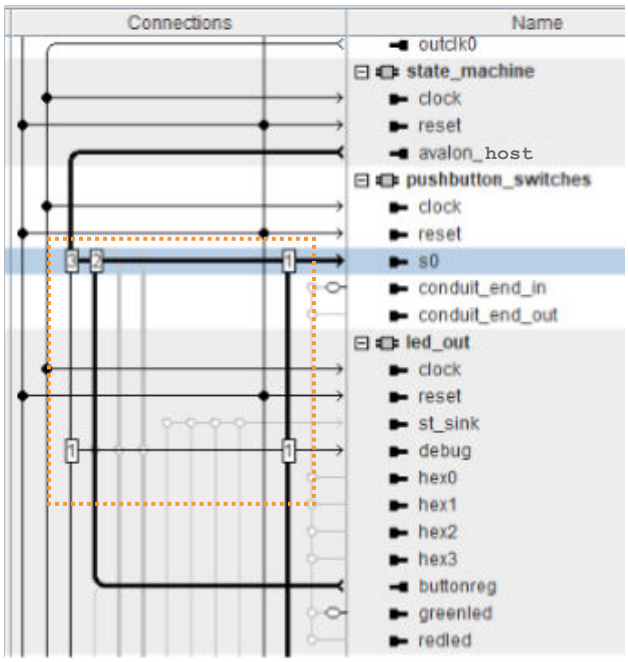
1. In Platform Designer, click **View ➤ Domains**.
2. Under **Interconnect Parameters**, click **Add** on the **Interface** tab to add a new requirement.
3. In the **Identifier** column, select the agent for fixed priority arbitration.

Figure 204. Interface Tab of Domains Tab



4. In the **Setting** column, select **Slave arbitration scheme**.
5. In the **Value** column, select **fixed-priority**.
6. Navigate to the **System View** tab.
7. In the **System View** tab, right-click the designated fixed priority agent, and then select **Show Arbitration Shares**.
8. For each host connected to the fixed priory arbitration agent, type a numerical arbitration priority in the box that appears in place of the connection circle.

Figure 205. Arbitration Priority Box



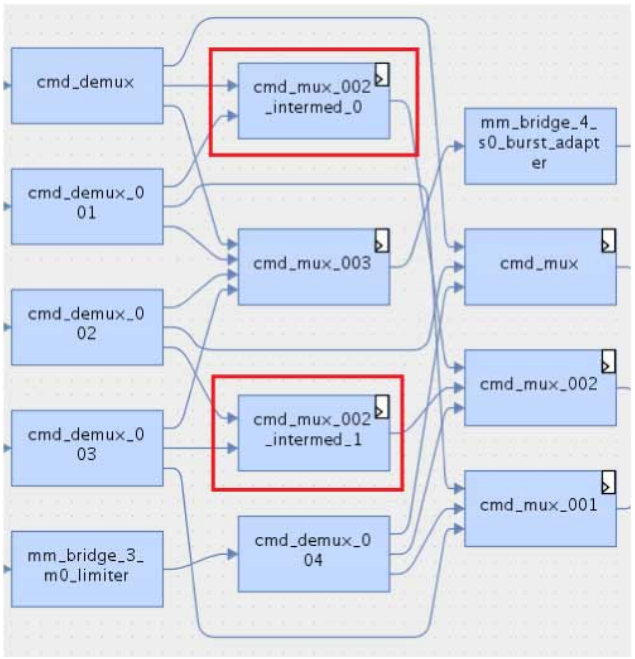
9. Right click the designated fixed priority agent and uncheck **Show Arbitration Shares** to return to the connection circles.

5.1.5.2.2. Fixed Priority Arbitration with AXI Managers and Avalon Memory Mapped Agents

When an AXI manager is connected to a designated fixed priority arbitration Avalon memory mapped agent, Platform Designer interconnect automatically instantiates an intermediary multiplexer in front of the Avalon memory mapped agent.

Since AXI managers have separate read and write channels, each channel appears as two separate managers to the Avalon memory mapped agent. To support fairness between the AXI manager's read and write channels, the instantiated round-robin intermediary multiplexer arbitrates between simultaneous read and write commands from the AXI manager to the fixed-priority Avalon memory mapped agent.

Figure 206. Intermediary Multiplexer Between AXI Manager and Avalon Memory Mapped Agent



When an AXI manager is connected to a fixed priority AXI subordinate, the manager's read and write channels are directly connected to the AXI subordinate's fixed-priority multiplexers. In this case, there is one multiplexer for the read command, and one multiplexer for the write command and therefore an intermediary multiplexer is not required.

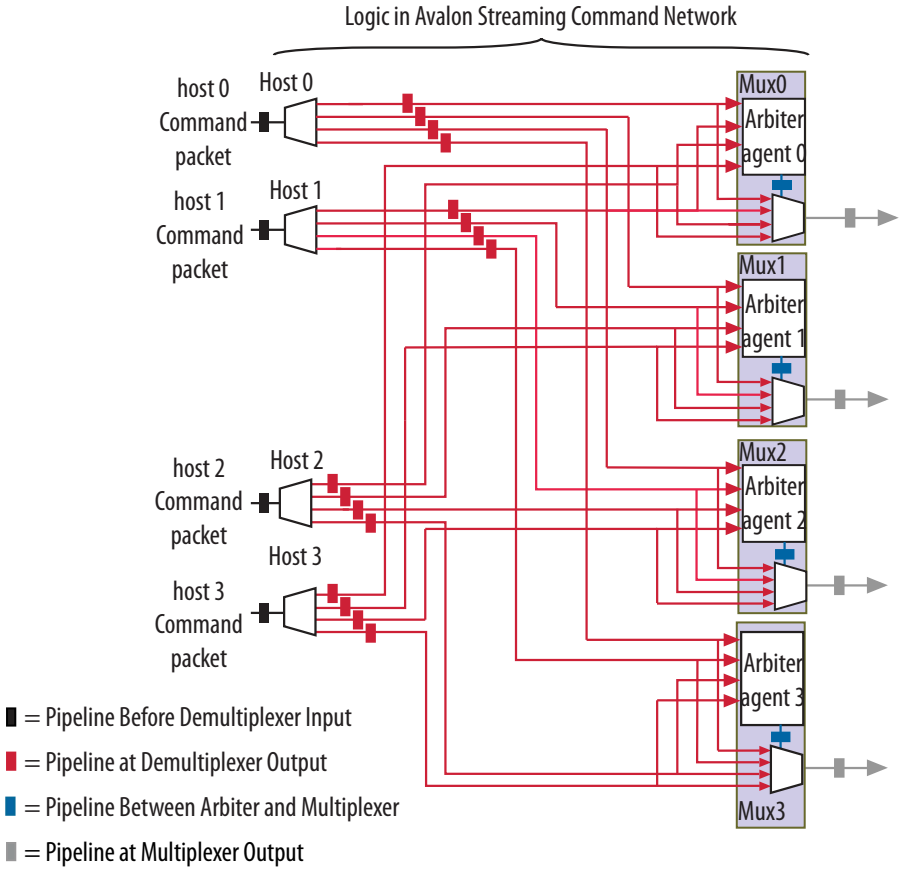
The red rectangles indicate placement of the intermediary multiplexer between the AXI manager and Avalon memory mapped agent due to the separate read and write channels of the AXI manager.

5.1.6. Memory-Mapped Arbiter

The input to the Memory-Mapped Arbiter is the command packet for all hosts requesting access to a specific agent. The arbiter outputs the channel number for the selected host. This channel number controls the output of a multiplexer that selects the agent device.

Figure 207. Arbitration Logic

In this example, four Avalon memory-mapped hosts connect to four Avalon memory-mapped agents. In each cycle, an arbiter positioned in front of each Avalon memory-mapped agent selects among the requesting Avalon memory-mapped hosts.



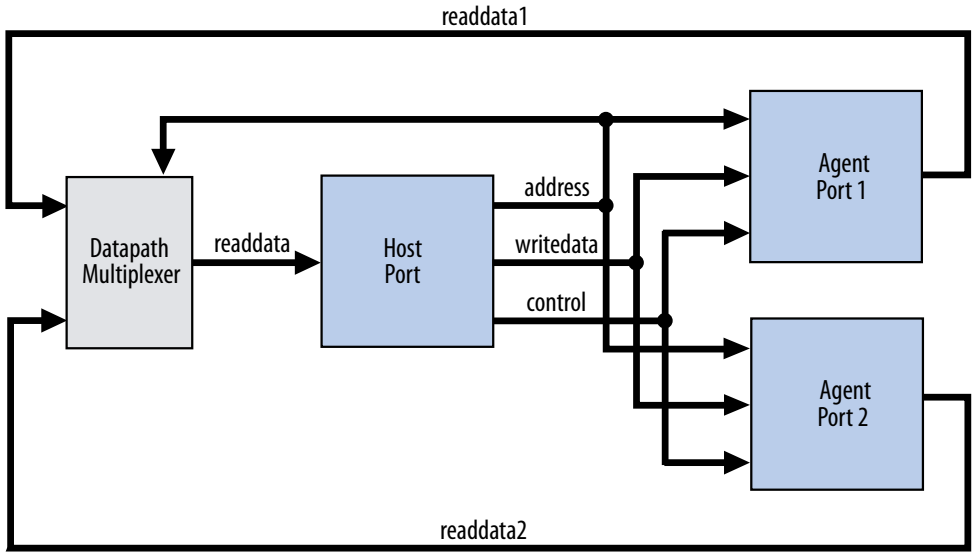
Note: If you specify a **Limit interconnect pipeline stages to** parameter greater than zero, the output of the Arbiter is registered. Registering this output reduces the amount of combinational logic between the host and the interconnect, increasing the f_{MAX} of the system.

Note: You can use the Memory-Mapped Arbiter for both round-robin and fixed priority arbitration.

5.1.7. Datapath Multiplexing Logic

Datapath multiplexing logic drives the `writedata` signal from the granted host to the selected agent, and the `readdata` signal from the selected agent back to the requesting host. Platform Designer generates separate datapath multiplexing logic for every host in the system (`readdata`), and for every agent in the system (`writedata`). Platform Designer does not generate multiplexing logic if it is not needed.

Figure 208. Datapath Multiplexing Logic for One Host and Two Agents



5.1.8. Width Adaptation

Platform Designer width adaptation converts between Avalon memory-mapped host and agents with different data and byte enable widths, and manages the run-time size requirements of AXI. Width adaptation for AXI to Avalon interfaces is also supported.

5.1.8.1. Memory-Mapped Width Adapter

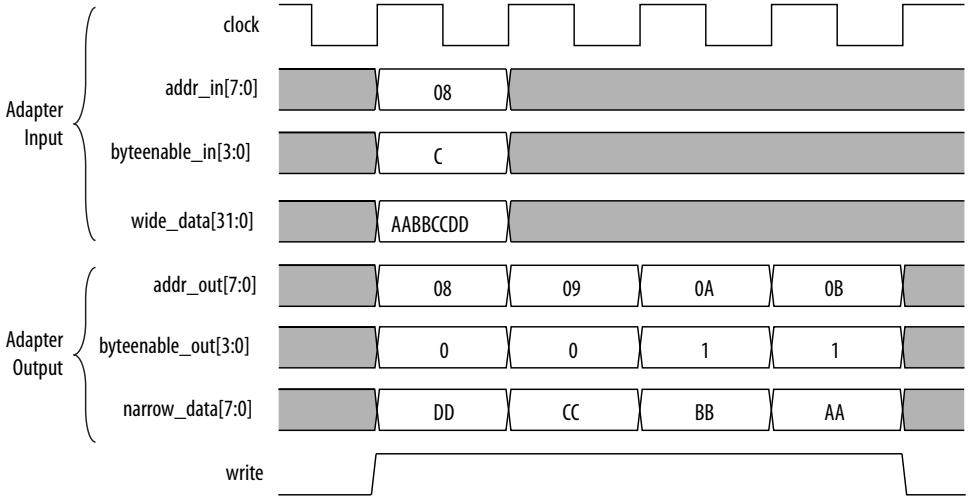
The Memory-Mapped Width Adapter is used in the Avalon streaming domain and operates with information contained in the packet format.

The memory-mapped width adapter accepts packets on its sink interface with one data width and produces output packets on its source interface with a different data width. The ratio of the narrow data width must be a power of two, such as 1:4, 1:8, and 1:16. The ratio of the wider data width to the narrower width must also be a power of two, such as 4:1, 8:1, and 16:1 These output packets may have a different size if the input size exceeds the output data bus width, or if data packing is enabled.

When the width adapter converts from narrow data to wide data, each input beat's data and byte enables are copied to the appropriate segment of the wider output data and byte enables signals.

Figure 209. Width Adapter Timing for a 4:1 Adapter

This adapter assumes that the field ordering of the input and output packets is the same, with the only difference being the width of the data and accompanying byte enable fields. When the width adapter converts from wide data to narrow data, the narrower data is transmitted over several beats. The first output beat contains the lowest addressed segment of the input data and byte enables.



5.1.8.1.1. AXI Wide-to-Narrow Adaptation

For all cases of AXI wide-to-narrow adaptation, read data is re-packed to match the original size. Responses are merged, with the following error precedence: DECERR, SLVERR, 0KAY, and EX0KAY.

Table 46. AXI Wide-to-Narrow Adaptation (Downsizing)

Burst Type	Behavior
Incrementing	If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted to an incrementing burst with a larger length and size equal to the output width. If the resulting burst is unsuitable for the subordinate, the burst is converted to multiple sequential bursts of the largest allowable lengths. For example, for a 2:1 downsizing ratio, an `INCR9` burst is converted into `INCR16 + INCR2` bursts. This is true if the maximum burstcount a subordinate can accept is 16, which is the case for AMBA 3 AXI subordinates. Avalon subordinates have a maximum burstcount of 64.
Wrapping	If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted to a wrapping burst with a larger length, with a size equal to the output width. If the resulting burst is unsuitable for the subordinate, the burst is converted to multiple sequential bursts of the largest allowable lengths; respecting wrap boundaries. For example, for a 2:1 downsizing ratio, a `WRAP16` burst is converted into two or three `INCR` bursts, depending on the address.
Fixed	If the transaction size is less than or equal to the output width, the burst is unmodified. Otherwise, it is converted into repeated sequential bursts over the same addresses. For example, for a 2:1 downsizing ratio, a FIXED single burst is converted into an `INCR2` burst.

5.1.8.1.2. AXI Narrow-to-Wide Adaptation

Table 47. AXI Narrow-to-Wide Adaptation (Upsizing)

Burst Type	Behavior
Incrementing	The burst (and its response) passes through unmodified. Data and write strobes are placed in the correct output segment.
Wrapping	The burst (and its response) passes through unmodified.
Fixed	The burst (and its response) passes through unmodified.

5.1.9. Burst Adapter

Platform Designer interconnect uses the memory-mapped burst adapter to accommodate the burst capabilities of each interface in the system, including interfaces that do not support burst transfers.

The maximum burst length for each interface is a property of the interface and is independent of other interfaces in the system. Therefore, a specific host may be capable of initiating a burst longer than an agent's maximum supported burst length. In this case, the burst adapter translates the large host burst into smaller bursts, or into individual agent transfers if the agent does not support bursting. Until the host completes the burst, arbiter logic prevents other hosts from accessing the target agent. For example, if a host initiates a burst of 16 transfers to an agent with maximum burst length of 8, the burst adapter initiates 2 bursts of length 8 to the agent.

Avalon memory mapped burst transactions allow a host uninterrupted access to an agent for a specified number of transfers. The host specifies the number of transfers when it initiates the burst. Once a burst begins between a host and agent, arbiter logic is locked until the burst completes. For burst hosts, the length of the burst is the number of cycles that the host has access to the agent, and the selected arbitration shares have no effect.

Note: AXI managers can issue burst types that Avalon cannot accept, for example, fixed bursts. In this case, the burst adapter converts the fixed burst into a sequence of transactions to the same address.

Note: For AMBA 4 AXI subordinates, Platform Designer allows 256-beat `INCR` bursts. You must ensure that 256-beat narrow-sized `INCR` bursts are shortened to 16-beat narrow-sized INCR bursts for AMBA 3 AXI subordinates.

Avalon memory mapped hosts always issue addresses that are aligned to the size of the transfer. However, when Platform Designer uses a narrow-to-wide width adaptation, the resulting address may be unaligned. For unaligned addresses, the burst adapter issues the maximum sized bursts with appropriate byte enables. This brings the burst-in-progress up to an aligned agent address. Then, it completes the burst on aligned addresses.

The burst adapter supports variable wrap or sequential burst types to accommodate different properties of memory-mapped hosts. Some bursting hosts can issue more than one burst type.

Burst adaptation is available for Avalon to Avalon, Avalon to AXI, and AXI to Avalon, and AXI to AXI connections. For information about AXI-to-AXI adaptation, refer to *AXI Wide-to-Narrow Adaptation*

Note: For AMBA 4 AXI to AMBA 3 AXI connections, Platform Designer follows an AMBA 4 AXI 256 burst length to AMBA 3 AXI 16 burst length.

5.1.9.1. Burst Adapter Implementation Options

Platform Designer automatically inserts burst adapters into your system depending on your host and agent connections, and properties. You can select burst adapter implementation options on the **Interconnect Requirements** tab.

To access the implementation options, you must select the **Burst adapter implementation** setting for the `$system` identifier.

- **Generic converter (slower, lower area)**—Default. Controls all burst conversions with a single converter that can adapt incoming burst types. This results in an adapter that has lower f_{MAX}, but smaller area.
- **Per-burst-type converter (faster, higher area)**—Controls incoming bursts with a specific converter, depending on the burst type. This results in an adapter that has higher f_{MAX}, but higher area. This setting is useful when you have AXI managers or subordinates and you want a higher f_{MAX}.

Note: For more information about the **Interconnect Requirements** tab, refer to *Creating a System with Platform Designer*.

Related Information

Creating a System with Platform Designer on page 11

5.1.9.2. Burst Adaptation: AXI to Avalon

The following entries specify the behavior when converting between AXI and Avalon burst types.

Table 48. Burst Adaptation: AXI to Avalon

Burst Type	Behavior
Incrementing	**Sequential Subordinate** Bursts that exceed `slave_max_burst_length` are converted to multiple sequential bursts of a length less than or equal to the `slave_max_burst_length`. Otherwise, the burst is unconverted. For example, for an Avalon agent with a maximum burst length of 4, an `INCR7` burst is converted to `INCR4 + INCR3`. **Wrapping Subordinate** Bursts that exceed the `slave_max_burst_length` are converted to multiple sequential bursts of length less than or equal to the `slave_max_burst_length`. Bursts that exceed the wrapping boundary are converted to multiple sequential bursts that respect the Avalon agent's wrapping boundary.
Wrapping	**Sequential Subordinate** A WRAP burst is converted to multiple sequential bursts. The sequential bursts are less than or equal to the `max_burst_length` and respect the transaction's wrapping boundary **Wrapping Subordinate** If the WRAP transaction's boundary matches the Avalon agent's boundary, then the burst passes through. Otherwise, the burst is converted to sequential bursts that respect both the transaction and Avalon agent wrap boundaries.
Fixed	Fixed bursts are converted to sequential bursts of length 1 that repeatedly access the same address.
Narrow	All narrow-sized bursts are broken into multiple bursts of length 1.

5.1.9.3. Burst Adaptation: Avalon to AXI

The following entries specify the behavior when converting between Avalon and AXI burst types.

Note: The Platform Designer-generated interconnect that adapts between an Avalon memory-mapped interface host and a connected AXI subordinate does not account for the AXI3 or AXI4 4KB boundary restriction for burst transactions. When connecting an Avalon memory-mapped interface FPGA host to an AXI subordinate in Platform Designer, you must ensure that the bursts do not exceed the AXI3 or AXI4 4KB boundary restriction for burst transactions.

Table 49. Burst Adaptation: Avalon to AXI

Burst Type	Definition
Sequential	Bursts of length greater than 16 are converted to multiple `INCR` bursts of a length less than or equal to 16. Bursts of length less than or equal to 16 are not converted.
Wrapping	Only Avalon hosts with `alwaysBurstMaxBurst = true` are supported. The WRAP burst is passed through if the length is less than or equal to 16. Otherwise, it is converted to two or more `INCR` bursts that respect the transaction's wrap boundary.
`GENERIC_CONVERTER`	Controls all burst conversions with a single converter that adapts all incoming burst types, resulting in an adapter that has smaller area, but lower f_{MAX}.

5.1.10. Waitrequest Allowance Adapter

The Waitrequest Allowance Adapter allows a connection between a host and an agent interface with different `waitrequestAllowance` properties.

The Waitrequest Allowance adapter provides the following features:

- The adapter is used in the memory-mapped domain and operates with signals on the memory-mapped interface.
- Signal widths and all properties other than `waitrequestAllowance` are identical on host and agent interfaces.
- The adapter does not modify any command properties such as data width, burst type, or burst count.
- The adapter is inserted by the Platform Designer interconnect software when a host and agent with different `waitrequestAllowance` property are connected.

When the agent has a `waitrequestAllowance = n` the host must deassert `read` or `write` signals after <*n*> transfers when `waitrequest` is asserted.

Table 50. Interconnect Scenarios Requiring waitrequestAllowance

Host *(m)* / Agent *(n)* waitrequestAllowance	Adaptation Required	Description	Adapter Function
m = *n*	No	The host `waitrequestAllowance` is equal to the agent's `waitrequestAllowance`.	All signals are passed through.
m = 0; *n* > 0	Yes	The host cannot send when `waitrequest=1`, but holds the value on the bus. This would result in the agent receiving multiple copies.	The adapter deasserts `valid` when input `waitrequest` is asserted.

continued...

Host *(m)* / Agent *(n)* waitrequestAllowance	Adaptation Required	Description	Adapter Function
		Requires adaptation to prevent.	
m < *n*; *m* != 0	No	The host can send <*m*> transfers after `waitrequest` is asserted. The agent receives fewer than <*n*> transfers, which is acceptable.	All signals are passed through.
m > *n*; *n* = 0	Yes	The agent cannot accept transfers when `waitrequest` is asserted. Transfers sent when `waitrequest=1` can be lost. Prevention requires adaptation in the form of transfer buffering.	If the input `waitrequest` is asserted, the adapter buffers the input data.
m > *n*; *n* > 0	Yes	The agent cannot accept more than <*n*> transfers after `waitrequest` is asserted, however the host can send up to <*m*> transfers. Transfers (<*m*> – <*n*>) can be lost. Prevention requires adaptation in the form of transfer buffering.	The adapter buffers the input data.

5.1.11. Read and Write Responses

Platform Designer merges write responses if a write is converted (burst adapted) into multiple bursts. Platform Designer requires read response merging for a downsized (wide-to-narrow width adapted) read.

Platform Designer merges responses based on the following precedence rule:

```
DECERR > SLVERR > OKAY > EXOKAY
```

Adaptation between a host with write responses and an agent without write responses can be costly, especially if there are multiple agents, or if the agent supports bursts. To minimize the cost of logic between agents, consider placing the agents that do not have write responses behind a bridge so that the write response adaptation logic cost is only incurred once, at the bridge's agent interface.

The following table describes what happens when there is a mismatch in response support between the host and agent.

Table 51. Response Support for Mismatched Host and Agent

	Agent with Response	Agent Without Response
Host with Response	Interconnect delivers response from the agent to the host. Response merging or duplication may be necessary for bus sizing.	Interconnect delivers an OKAY response to the host
Host without Response	Host ignores responses from the agent	No need for responses. Host, agent and interconnect operate without response support.

Note: If there is a bridge between the host and the endpoint agent, and the responses must come from the endpoint agent, ensure that the bridge passes the appropriate response signals through from the endpoint agent to the host.

If the bridge does not support responses, then the responses are generated by the interconnect at the agent interface of the bridge, and responses from the endpoint agent are ignored.

For the response case where the transaction violates security settings or uses an illegal address, the interconnect routes the transactions to the default agent. For information about Platform Designer system security, refer to Manage System Security. For information about specifying a default agent, refer to *Error Response Agent* in *Platform Designer System Design Components*.

Note: Avalon memory mapped agents without a `response` signal are not able to notify a connected host that a transaction has not completed successfully. As a result, Platform Designer interconnect generates an OKAY response on behalf of the Avalon memory mapped agent.

Related Information

- Avalon Host and AXI Manager Network Interfaces on page 258
- Error Response Slave Intel FPGA IP on page 390
- Error Correction Coding (ECC) in Platform Designer Interconnect on page 331

5.1.12. Platform Designer Address Decoding

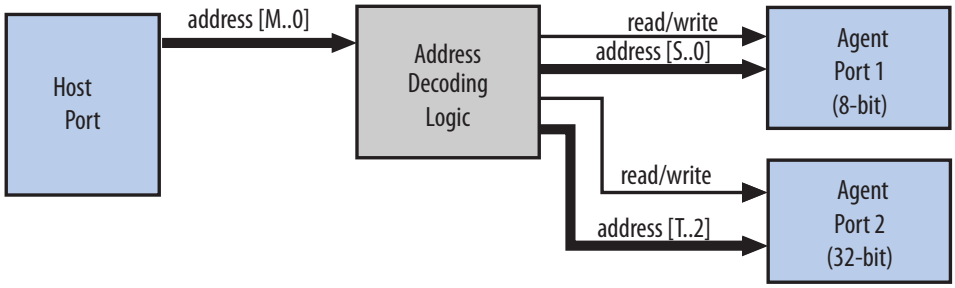
Address decoding logic forwards appropriate addresses to each agent.

Address decoding logic simplifies component design in the following ways:

- The interconnect selects an agent whenever it is being addressed by a host. Agent components do not need to decode the address to determine when they are selected.
- Agent addresses are properly aligned to the agent interface.
- Changing the system memory map does not involve manually editing HDL.

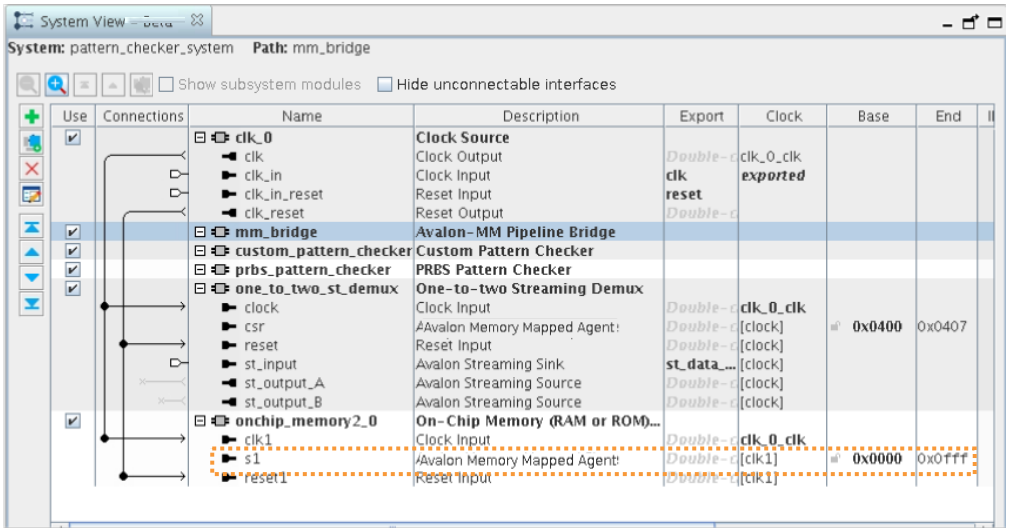
Figure 210. Address Decoding for One Host and Two Agents

In this example, Platform Designer generates separate address decoding logic for each host in a system. The address decoding logic processes the difference between the host address width (<M>) and the individual agent address widths (<S>) and (<T>). The address decoding logic also maps only the necessary host address bits to access words in each agent's address space.



Platform Designer controls the base addresses with the **Base** setting of active components on the **System View** tab. The base address of an agent component must be a multiple of the address span of the component. This restriction is part of the Platform Designer interconnect to allow the address decoding logic to be efficient, and to achieve the best possible f_{MAX}.

Figure 211. Address Decoding Base Settings



5.2. Avalon Streaming Interfaces

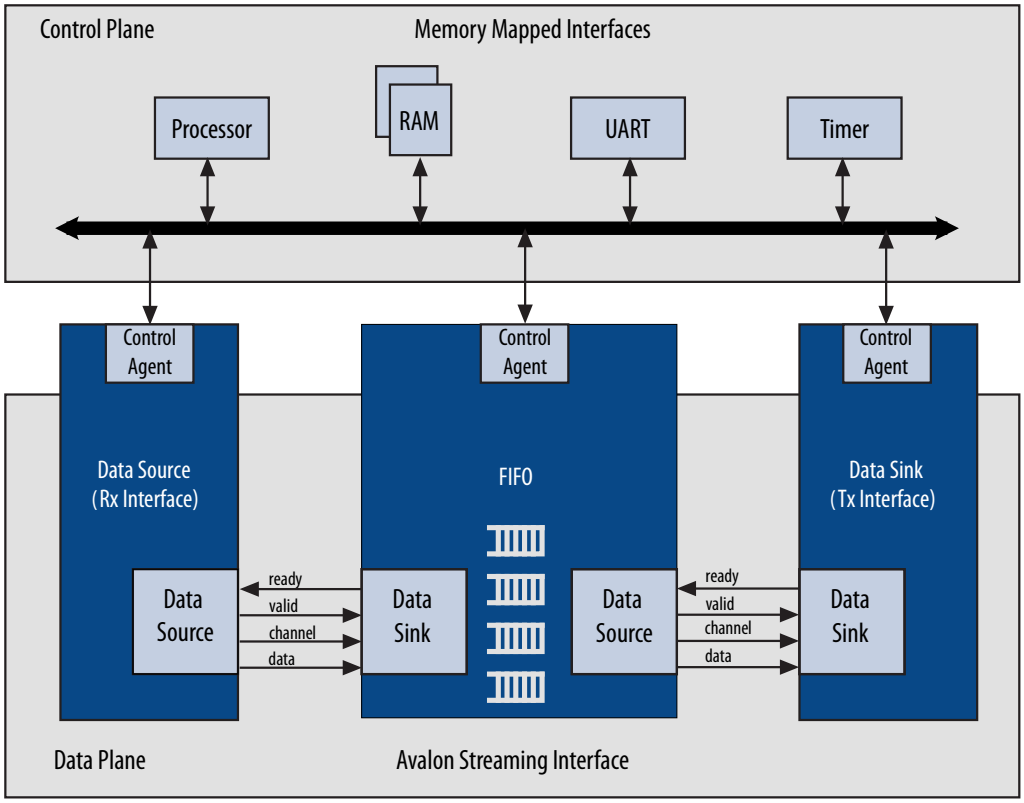
High bandwidth components with streaming data typically use Avalon streaming interfaces for the high throughput datapath. Streaming interfaces can also use memory-mapped connection interfaces to provide an access point for control. In contrast to the memory-mapped interconnect, the Avalon streaming interconnect always creates a point-to-point connection between a single data source and data sink.

Figure 212 on page 279 contains the following connection pairs:

- Data source in the Rx Interface transfers data to the data sink in the FIFO.
- Data source in the FIFO transfers data to the `Tx Interface` data sink.

The memory-mapped interface allows a processor to access the data source, FIFO, or data sink to provide system control. If your source and sink interfaces have different formats, for example, a 32-bit source and an 8-bit sink, Platform Designer automatically inserts the necessary adapters. You can view the adapters on the **System View** tab by clicking **System ➤ Show System with Platform Designer Interconnect**.

Figure 212. Avalon Memory-Mapped and Avalon Streaming Interfaces



The following diagram shows a source-sink pair that includes only the `data` signal. The sink must be able to receive data as soon as the source interface comes out of reset.

Figure 213. Avalon Streaming Connection Between the Source and Sink

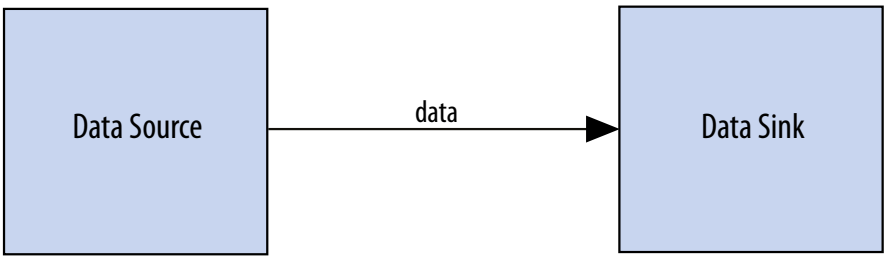
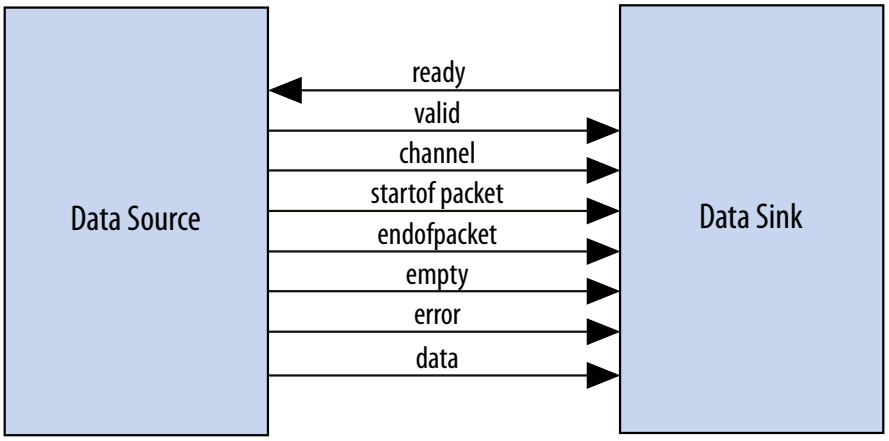


Figure 214. Signals Indicating the Start and End of Packets, Channel Numbers, Error Conditions, and Backpressure

All data transfers using Avalon streaming interconnect occur synchronously on the rising edge of the associated clock interface. Throughput and frequency of a system depends on the components and how they are connected.



The IP Catalog includes Avalon streaming components that you can use to create datapaths, including datapaths whose input and output streams have different properties. Generated systems that include memory-mapped host and agent components may also use these Avalon streaming components because Platform Designer generation creates interconnect with a structure similar to a network topology, as described in *Platform Designer Transformations*. The following sections introduce the Avalon streaming components.

Related Information

Platform Designer Transformations on page 256

5.2.1. Avalon Streaming Adapters

Platform Designer automatically adds Avalon streaming adapters between two components during system generation when it detects mismatched interfaces. If you connect mismatched Avalon streaming sources and sinks, for example, a 32-bit source and an 8-bit sink, Platform Designer inserts the appropriate adapter type to connect the mismatched interfaces.

After generation, you can view the inserted adapters selecting **System ➤ Show System With Platform Designer Interconnect**. For each mismatched source-sink pair, Platform Designer inserts an Avalon streaming adapter. The adapter instantiates the necessary adaptation logic as sub-components. You can review the logic for each adapter instantiation in the Hierarchy view by expanding each adapter's source and sink interface and comparing the relevant ports. For example, to determine why a channel adapter is inserted, expand the channel adapter's sink and source interfaces and review the channel port properties for each interface.

You can turn off the auto-inserted adapters feature by adding the `qsys_enable_avalon_streaming_transform=off` command to the `quartus.ini` file. When you turn off the auto-inserted adapters feature, if mismatched interfaces are detected during system generation, Platform Designer does not insert adapters and reports the mismatched interface with validation error message.

Note: The auto-inserted adapters feature does not work for video IP connections.

5.2.1.1. Avalon Streaming Adapter

The Avalon streaming adapter combines the logic of the channel, error, data format, and timing adapters. The Avalon streaming adapter provides adaptations between interfaces that have mismatched Avalon streaming endpoints. Based on the source and sink interface parameterizations for the Avalon streaming adapter, Platform Designer instantiates the necessary adapter logic (channel, error, data format, or timing) as hierarchal sub-components.

5.2.1.1.1. Avalon Streaming Adapter Parameters Common to Source and Sink Interfaces

Table 52. Avalon Streaming Adapter Parameters Common to Source and Sink Interfaces

Parameter Name	Description
Symbol Width	Width of a single symbol in bits.
Use Packet	Indicates whether the source and sink interfaces connected to the adapter's source and sink interfaces include the `startofpacket` and `endofpacket` signals, and the optional `empty` signal.

5.2.1.1.2. Avalon Streaming Adapter Upstream Source Interface Parameters

Table 53. Avalon Streaming Adapter Upstream Source Interface Parameters

Parameter Name	Description
Source Data Width	Controls the data width of the source interface `data` port.
Source Top Channel	Maximum number of output channels allowed.
Source Channel Port Width	Sets the bit width of the source interface `channel` port. If set to 0, there is no `channel` port on the sink interface.
Source Error Port Width	Sets the bit width of the source interface `error` port. If set to 0, there is no `error` port on the sink interface.
Source Error Descriptors	A list of strings that describe the error conditions for each bit of the source interface `error` signal.
Source Uses Empty Port	Indicates whether the source interface includes the `empty` port, and whether the sink interface should also include the `empty` port.
Source Empty Port Width	Indicates the bit width of the source interface `empty` port, and sets the bit width of the sink interface `empty` port.
Source Uses Valid Port	Indicates whether the source interface connected to the sink interface uses the `valid` port, and if set, configures the sink interface to use the `valid` port.
Source Uses Ready Port	Indicates whether the sink interface uses the `ready` port, and if set, configures the source interface to use the `ready` port.
Source Ready Latency	Specifies what ready latency to expect from the source interface connected to the adapter's sink interface.

5.2.1.1.3. Avalon Streaming Adapter Downstream Sink Interface Parameters

Table 54. Avalon Streaming Adapter Downstream Sink Interface Parameters

Parameter Name	Description
Sink Data Width	Indicates the bit width of the `data` port on the sink interface connected to the source interface.
Sink Top Channel	Maximum number of output channels allowed.
Sink Channel Port Width	Indicates the bit width of the `channel` port on the sink interface connected the source interface.
Sink Error Port Width	Indicates the bit width of the `error` port on the sink interface connected to the adapter's source interface. If set to zero, there is no error port on the source interface.
Sink Error Descriptors	A list of strings that describe the error conditions for each bit of the `error` port on the sink interface connected to the source interface.
Sink Uses Empty Port	Indicates whether the sink interface connected to the source interface uses the `empty` port, and whether the source interface should also use the `empty` port.
Sink Empty Port Width	Indicates the bit width of the `empty` port on the sink interface connected to the source interface, and configures a corresponding `empty` port on the source interface.
Sink Uses Valid Port	Indicates whether the sink interface connected to the source interface uses the `valid` port, and if set, configures the source interface to use the `valid` port.
Sink Uses Ready Port	Indicates whether the `ready` port on the sink interface is connected to the source interface , and if set, configures the sink interface to use the ready port.
Sink Ready Latency	Specifies what ready latency to expect from the source interface connected to the sink interface.

5.2.1.2. Channel Adapter

The channel adapter provides adaptations between interfaces that have different channel signal widths.

Table 55. Channel Adapter Adaptations

Condition	Description of Adapter Logic
The source uses channels, but the sink does not.	Platform Designer gives a warning at generation time. The adapter provides a simulation error and signals an error for data for any channel from the source other than 0.
The sink has channel, but the source does not.	Platform Designer gives a warning at generation time, and the channel inputs to the sink are all tied to a logical 0.
The source and sink both support channels, and the source's maximum channel number is less than the sink's maximum channel number.	The source's channel is connected to the sink's channel unchanged. If the sink's channel signal has more bits, the higher bits are tied to a logical 0.
The source and sink both support channels, but the source's maximum channel number is greater than the sink's maximum channel number.	The source's channel is connected to the sink's channel unchanged. If the source's channel signal has more bits, the higher bits are left unconnected. Platform Designer gives a warning that channel information may be lost. An adapter provides a simulation error message and an error indication if the value of channel from the source is greater than the sink's maximum number of channels. In addition, the `valid` signal to the sink is deasserted so that the sink never sees data for channels that are out of range.

5.2.1.2.1. Avalon Streaming Channel Adapter Input Interface Parameters

Table 56. Avalon Streaming Channel Adapter Input Interface Parameters

Parameter Name	Description
Channel Signal Width (bits)	Width of the input channel signal in bits
Max Channel	Maximum number of input channels allowed.

5.2.1.2.2. Avalon Streaming Channel Adapter Output Interface Parameters

Table 57. Avalon Streaming Channel Adapter Output Interface Parameters

Parameter Name	Description
Channel Signal Width (bits)	Width of the output channel signal in bits.
Max Channel	Maximum number of output channels allowed.

5.2.1.2.3. Avalon Streaming Channel Adapter Common to Input and Output Interface Parameters

Table 58. Avalon Streaming Channel Adapter Common to Input and Output Interface Parameters

Parameter Name	Description
Data Bits Per Symbol	Number of bits for each symbol in a transfer.
Include Packet Support	When the Avalon Streaming Channel adapter supports packets, the `startofpacket`, `endofpacket`, and optional `empty` signals are included on its sink and source interfaces.
Include Empty Signal	Indicates whether an `empty` signal is required.
Data Symbols Per Beat	Number of symbols per transfer.
Support Backpressure with the ready signal	Indicates whether a `ready` signal is required.
Ready Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.
Error Signal Width (bits)	Bit width of the `error` signal.
Error Signal Description	A list of strings that describes what each bit of the `error` signal represents.

5.2.1.3. Data Format Adapter

The data format adapter allows you to connect interfaces that have different values for the parameters defining the `data` signal, or interfaces where the source does not use the `empty` signal, but the sink does use the `empty` signal. One of the most common uses of this adapter is to convert data streams of different widths.

Table 59. Data Format Adapter Adaptations

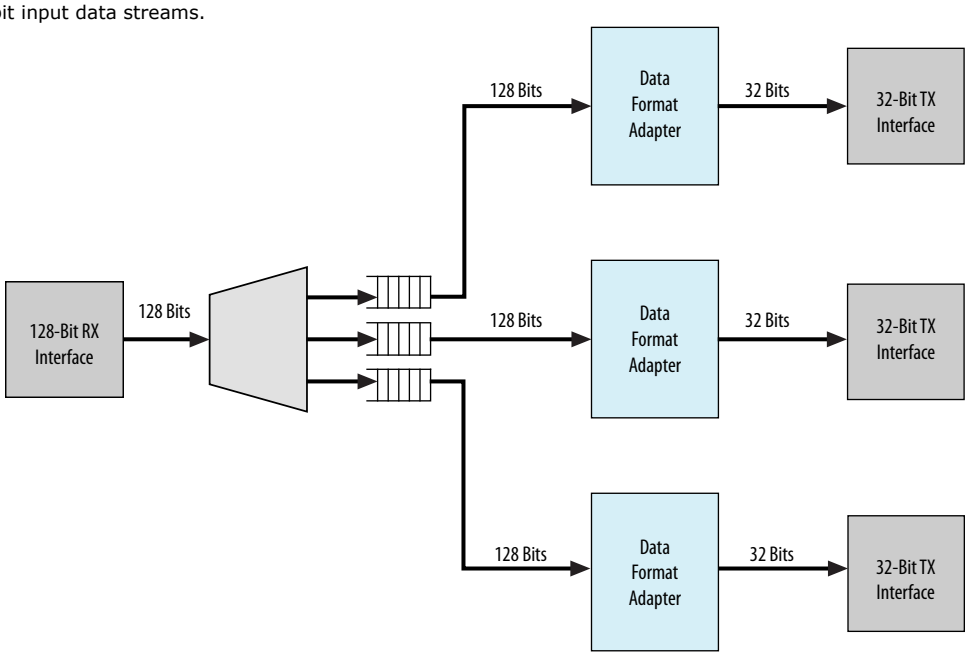
Condition	Description of Adapter Logic
The source and sink's bits per symbol parameters are different.	The connection cannot be made.
The source and sink have a different number of symbols per beat.	The adapter converts the source's width to the sink's width.

continued...

Condition	Description of Adapter Logic
	If the adaptation is from a wider to a narrower interface, a beat of data at the input corresponds to multiple beats of data at the output. If the input `error` signal is asserted for a single beat, it is asserted on output for multiple beats. If the adaptation is from a narrow to a wider interface, multiple input beats are required to fill a single output beat, and the output `error` is the logical OR of the input `error` signal.
The source uses the `empty` signal, but the sink does not use the `empty` signal.	Platform Designer cannot make the connection.

Figure 215. Avalon Streaming Interconnect with Data Format Adapter

In this example, the data format adapter allows a connection between a 128-bit output data stream and three 32-bit input data streams.



5.2.1.3.1. Avalon Streaming Data Format Adapter Input Interface Parameters

Table 60. Avalon Streaming Data Format Adapter Input Interface Parameters

Parameter Name	Description
Data Symbols Per Beat	Number of symbols per transfer.
Include Empty Signal	Indicates whether an `empty` signal is required.

5.2.1.3.2. Avalon Streaming Data Format Adapter Output Interface Parameters

Table 61. Avalon Streaming Data Format Adapter Output Interface Parameters

Parameter Name	Description
Data Symbols Per Beat	Number of symbols per transfer.
Include Empty Signals	Indicates whether an `empty` signal is required.

5.2.1.3.3. Avalon Streaming Data Format Adapter Common to Input and Output Interface Parameters

Table 62. Avalon Streaming Data Format Adapter Common to Input and Output Interface Parameters

Parameter Name	Description
Data Bits Per Symbol	Number of bits for each symbol in a transfer.
Include Packet Support	When the Avalon Streaming Data Format adapter supports packets, Platform Designer uses `startofpacket`, `endofpacket`, and `empty` signals.
Channel Signal Width (bits)	Width of the output channel signal in bits.
Max Channel	Maximum number of channels allowed.
Read Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.
Error Signal Width (bits)	Width of the `error` signal output in bits.
Error Signal Description	A list of strings that describes what each bit of the `error` signal represents.

5.2.1.4. Error Adapter

The error adapter ensures that per-bit-error information provided by the source interface is correctly connected to the sink interface's input error signal. Error conditions that both source and sink can process are connected. If the source has an `error` signal representing an error condition that is not supported by the sink, the signal is left unconnected; the adapter provides a simulation error message and an error indication if the error is asserted. If the sink has an error condition that is not supported by the source, the sink's input error bit corresponding to that condition is set to 0.

Note: The output interface error signal descriptor accepts an error set with an `other` descriptor. Platform Designer assigns the bit-wise `ORing` of all input error bits that are unmatched, to the output interface error bits set with the `other` descriptor.

5.2.1.4.1. Avalon Streaming Error Adapter Input Interface Parameters

Table 63. Avalon Streaming Error Adapter Input Interface Parameters

Parameter Name	Description
Error Signal Width (bits)	The width of the `error` signal. Valid values are 0–256 bits. Type `0` if the `error` signal is not used.
Error Signal Description	The description for each of the error bits. If scripting, separate the description fields by commas. For a successful connection, the description strings of the error bits in the source and sink must match and are case sensitive.

5.2.1.4.2. Avalon Streaming Error Adapter Output Interface Parameters

Table 64. Avalon Streaming Error Adapter Output Interface Parameters

Parameter Name	Description
Error Signal Width (bits)	The width of the `error` signal. Valid values are 0–256 bits. Type `0` if you do not need to send error values.
Error Signal Description	The description for each of the error bits. Separate the description fields by commas. For successful connection, the description of the error bits in the source and sink must match, and are case sensitive.

5.2.1.4.3. Avalon Streaming Error Adapter Common to Input and Output Interface Parameters

Table 65. Avalon Streaming Error Adapter Common to Input and Output Interface Parameters

Parameter Name	Description
Support Backpressure with the ready signal	Turn on this option to add the backpressure functionality to the interface.
Ready Latency	When the `ready` signal is used, the value for `ready_latency` indicates the number of cycles between when the ready signal is asserted and when valid data is driven.
Channel Signal Width (bits)	The width of the `channel` signal. A channel width of 4 allows up to 16 channels. The maximum width of the `channel` signal is eight bits. Set to 0 if channels are not used.
Max Channel	The maximum number of channels that the interface supports. Valid values are 0–255.
Data Bits Per Symbol	Number of bits per symbol.
Data Symbols Per Beat	Number of symbols per active transfer.
Include Packet Support	Turn on this option if the connected interfaces support a packet protocol, including the `startofpacket`, `endofpacket` and `empty` signals.
Include Empty Signal	Turn this option on if the cycle that includes the `endofpacket` signal can include empty symbols. This signal is not necessary if the number of symbols per beat is 1.

5.2.1.5. Timing Adapter

The timing adapter allows you to connect component interfaces that require a different number of cycles before driving or receiving data. This adapter inserts a FIFO buffer between the source and sink to buffer data or pipeline stages to delay the back-pressure signals. You can also use the timing adapter to connect interfaces that support the `ready` signal, and those that do not. The timing adapter treats all signals other than the `ready` and `valid` signals as payload, and simply drives them from the source to the sink.

Table 66. Timing Adapter Adaptations

Condition	Adaptation
The source has `ready`, but the sink does not.	In this case, the source can respond to `backpressure`, but the sink never needs to apply it. The `ready` input to the source interface is connected directly to logical 1.
The source does not have `ready`, but the sink does.	The sink may apply `backpressure`, but the source is unable to respond to it. There is no logic that the adapter can insert that prevents data loss when the source asserts `valid` but the sink is not ready. The adapter provides simulation time error messages if data is lost. The user is presented with a warning, and the connection is allowed.
The source and sink both support backpressure, but the sink's ready latency is greater than the source's.	The source responds to `ready` assertion or deassertion faster than the sink requires it. The number of pipeline stages equal to the difference in ready latency are inserted in the `ready` path from the sink back to the source, causing the source and the sink to see the same cycles as `ready` cycles.
The source and sink both support backpressure, but the sink's ready latency is less than the source's.	The source cannot respond to `ready` assertion or deassertion in time to satisfy the sink. A FIFO whose depth is equal to the difference in ready latency is inserted to compensate for the source's inability to respond in time.

5.2.1.5.1. Avalon Streaming Timing Adapter Input Interface Parameters

Table 67. Avalon Streaming Timing Adapter Input Interface Parameters

Parameter Name	Description
Support Backpressure with the ready signal	Indicates whether a `ready` signal is required.
Read Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.
Include Valid Signal	Indicates whether the sink interface requires a valid signal.

5.2.1.5.2. Avalon Streaming Timing Adapter Output Interface Parameters

Table 68. Avalon Streaming Timing Adapter Output Interface Parameters

Parameter Name	Description
Support Backpressure with the ready signal	Indicates whether a `ready` signal is required.
Read Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.
Include Valid Signal	Indicates whether the sink interface requires a valid signal.

5.2.1.5.3. Avalon Streaming Timing Adapter Common to Input and Output Interface Parameters

Table 69. Avalon Streaming Timing Adapter Common to Input and Output Interface Parameters

Parameter Name	Description
Data Bits Per Symbol	Number of bits for each symbol in a transfer.
Include Packet Support	Turn this option on if the connected interfaces support a packet protocol, including the `startofpacket`, `endofpacket` and `empty` signals.
Include Empty Signal	Turn this option on if the cycle that includes the `endofpacket` signal can include empty symbols. This signal is not necessary if the number of symbols per beat is 1.

continued...

Parameter Name	Description
Data Symbols Per Beat	Number of symbols per active transfer.
Channel Signal Width (bits)	Width of the output channel signal in bits.
Max Channel	Maximum number of output channels allowed.
Error Signal Width (bits)	Width of the output `error` signal in bits.
Error Signal Description	A list of strings that describes errors.

5.3. Avalon Streaming Credit Interfaces

Avalon Streaming Credit interfaces are for use with components that drive high-bandwidth, low-latency, unidirectional data. Typical applications include multiplexed streams, packets, and DSP data. The Avalon Streaming Credit interface signals can describe traditional streaming interfaces supporting a single stream of data, without knowledge of channels or packet boundaries. The interface can also support more complex protocols capable of burst and packet transfers with packets interleaved across multiple channels.

All Avalon Streaming Credit source and sink interfaces are not necessarily interoperable. However, if two interfaces provide compatible functions for the same application space, adapters are available to allow them to interoperate.

You can also connect an Avalon Streaming Credit source to an Avalon Streaming sink via an adapter. Similarly, you can connect an Avalon Streaming source to an Avalon Streaming Credit sink via an adapter.

Avalon Streaming Credit interfaces support datapaths requiring the following features:

- Low-latency, high-throughput point-to-point data transfer
- Multiple channel support with flexible packet interleaving
- Sideband signaling of channel, error, and start and end of packet delineation
- Support for data bursting
- Sideband signals for user-defined functionality

Related Information

- Avalon Streaming Credit Adapters on page 289
- Avalon Streaming Credit Multiplexer on page 300
- Avalon Streaming Credit Demultiplexer on page 302
- Avalon Streaming Credit Pipeline Bridge on page 305

5.3.1. Terms and Concepts

The Avalon Streaming Credit interface protocol defines the following terms and concepts:

- **Avalon Streaming Credit System**—an Avalon Streaming Credit system contains one or more Avalon Streaming Credit connections that transfer data from a source interface to a sink interface.
- **Avalon Streaming Credit Components**—a typical system using Avalon Streaming credit interfaces combines multiple functional modules, called components. The system designer configures the components and connects them together to implement a system.
- **Source and Sink Interfaces and Connections**—when two components are connected, credits flow from the sink to the source; and the data flows from the source interface to the sink interface. The combination of a source interface connected to a sink interface is referred to as a connection.
- **Transfers**—a transfer results in data and control propagation from a source interface to a sink interface. For data interfaces, sources can start data transfers only if they have credits available. Similarly, sinks can accept data only if they have outstanding credits.
- **Symbol**—a symbol is the smallest unit of data as defined for the interface. One or more symbols make up the single unit of data transferred in a clock cycle. The default symbol size is 8 bits.
- **Beat**—a beat is a single cycle transfer between a source and sink interface made up of one or more symbols.
- **Packet**—a packet is an aggregation of data and control signals that is transmitted together. A packet may contain a header to help routers and other network devices direct the packet to the correct destination. The packet format is defined by the application, not this specification. Avalon Streaming packets can be variable in length and can be interleaved across a connection. With an Avalon Streaming Credit interface, the use of packets is optional.

5.3.2. Avalon Streaming Credit Adapters

Avalon Streaming Credit adapters perform data width conversion when the Avalon Streaming Credit source and Avalon Streaming Credit sink have different data widths. For example, if an Avalon Streaming Credit source and Avalon Streaming Credit sink have different data widths, you can insert the Avalon Streaming Credit Wide to Narrow Adapter, or the Avalon Streaming Credit Narrow to Wide Adapter, for data width conversion.

Platform Designer interconnect supports the following data format adapters for Avalon Streaming Credit interfaces:

Related Information

- Avalon Interface Specifications
- Avalon Streaming Credit Wide to Narrow Adapter on page 290
- Avalon Streaming Credit Narrow to Wide Adapter on page 292
- Avalon Streaming Credit Max Credit Adapter on page 294
- Avalon Streaming Ready to Credit Adapter on page 296
- Avalon Streaming Credit to Ready Adapter on page 298

5.3.2.1. Avalon Streaming Credit Wide to Narrow Adapter

The Avalon Streaming Credit Wide to Narrow Adapter has the following blocks for performing primary functions.

Table 70. Avalon Streaming Credit Wide to Narrow Adapter Blocks

Adapter Block	Function
Data Management	Accepts incoming wide data and converts the wide data to narrow data for the sink.
Credit Management	Accepts incoming credits from the sink and sends credits to the source. However, credits received from the sink do not directly transfer to the source. Internally, this adapter implements credit management logic. The adapter sends data out if there are credits and data available. If all the credits from the sink are serviced, then the adapter holds the incoming data from the source side until the adapter receives more credits from the sink.
Packet Management	Accepts the `startofpacket`, `endofpacket`, and `empty` packet management signals and drives them appropriately, while maintaining packet and data integrity, on the output.
User Signal Management	Accepts user-defined signals (such as `channel`, `error`, `user_signal_per_packet`, `user_signal_per_symbol`) and sends the signals to the source side, in accordance with the wide-to-narrow conversion.

The adapter is available for parameterization as the **Avalon Streaming Credit Wide to Narrow Adapter Intel FPGA IP** in the Platform Designer IP Catalog.

Figure 216. Avalon Streaming Credit Wide to Narrow Adapter

5.3.2.1.1. Avalon Streaming Credit Wide to Narrow Adapter Interface Parameters

You can specify the following parameters for the Avalon Streaming Credit Wide to Narrow Adapter by double-clicking **Avalon Streaming Credit Wide to Narrow Adapter Intel FPGA IP** in the Platform Designer IP Catalog:

Note: The source data width must be an integer multiple of the sink data width.

Table 71. Avalon Streaming Credit Wide to Narrow Adapter Interface Parameters

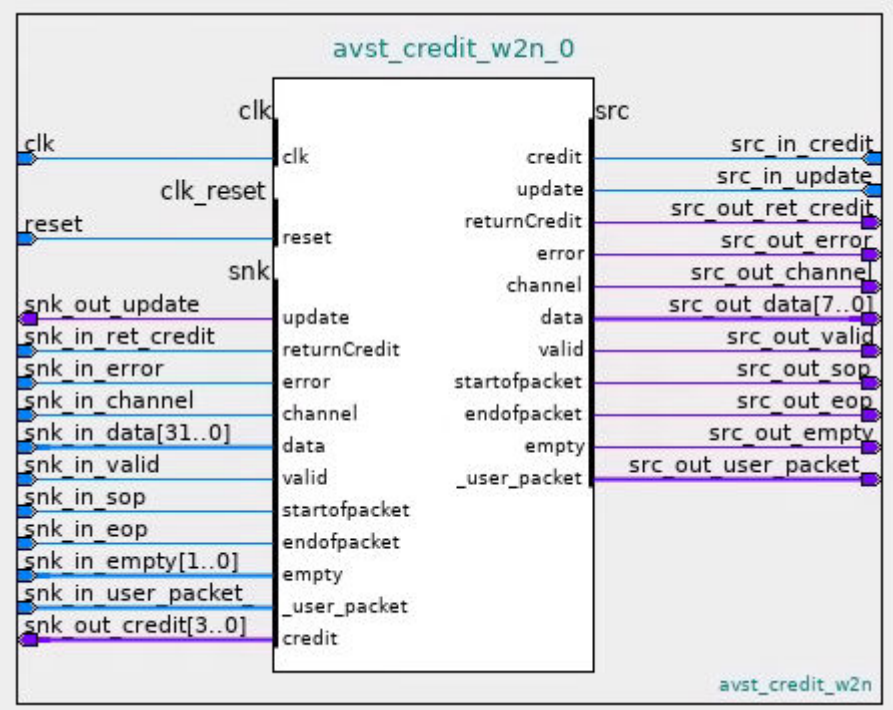
Parameter Name	Description	Legal Values
fifo depth	Specifies the number of words the adapter can save for transformation. Also dictates the maximum number of credits the sink can request from the upstream source side.	**8**
Maximum Credit Allowed (Sink Interface)	Specifies the maximum number of credits allowed by the sink interface. The default value is 8.	**8**

continued...

Parameter Name	Description	Legal Values
Number of symbols	Specifies the maximum number of symbols that can transfer.	**1-8192**
Symbol width	Specifies the number of data bits per symbol.	**1-8192**
Width of Channel Port	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is disabled.	**1-128**
Width of Error Port	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is disabled.	**1-1024**
Width of Empty Port	The width of the `empty` signal on the output interfaces. A value of 0 indicates that the empty signal is not in use. This parameter is disabled when **Use Empty** is disabled.	**1-1024**
Maximum Credit Allowed (Source Interface)	Specifies the maximum number of credits allowed by the source interface. Legal values are from 1 to 256.	**1,2,4,8,16,32,64,128, 256**
Use Packets	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.	**On|Off**
Use Empty	Enables or disables the `empty` signal.	**On|Off**
Use Channel	Enables or disables the `channel` signal.	**On|Off**
Use Error	Enables or disables the `error` signal.	**On|Off**
Enable User Signals	Allows you to specify the name, width, and type of user signals.	**On|Off**

5.3.2.1.2. Avalon Streaming Credit Wide to Narrow Adapter Interface Signals

Figure 217. Avalon Streaming Credit Wide to Narrow Adapter Interface Signals



Related Information

Avalon Streaming Credit Interface Signal Roles on page 355

5.3.2.2. Avalon Streaming Credit Narrow to Wide Adapter

The Avalon Streaming Credit Narrow to Wide Adapter has the following blocks for performing primary functions:

Table 72. Avalon Streaming Credit Narrow to Wide Adapter Blocks

Adapter Block	Function
Data Management	Accepts incoming narrow data and converts the narrow data to wide data for the sink.
Credit Management	Accepts incoming credits from the sink and sends credits to the source. However, credits received from the sink do not directly transfer to the source. Internally, this adapter implements credit management logic. The adapter sends data out if there are credits and data available. If all the credits from the sink are serviced, then the adapter holds the incoming data from the source side until the adapter receives more credits from the sink.
Packet Management	Accepts the `startofpacket`, `endofpacket`, and `empty` package management signals and drives them appropriately, while maintaining packet and data integrity, on the output.
User Signal Management	Accepts user-defined signals (such as `channel`, `error`, `user_signal_per_packet`, `user_signal_per_symbol`) and sends the signal to the source side, in accordance with the narrow-to-wide conversion.

The adaptor is available for parameterization as the **Avalon Streaming Credit Narrow to Wide Adapter Intel FPGA IP** in the Platform Designer IP Catalog.

Figure 218. Avalon Streaming Credit Narrow to Wide Adapter

5.3.2.2.1. Avalon Streaming Credit Narrow to Wide Adapter Interface Parameters

You can specify the following parameters for the Avalon Streaming Credit Narrow to Wide Adapter by double-clicking **Avalon Streaming credit narrow to wide adapter Intel FPGA IP** in the Platform Designer IP Catalog.

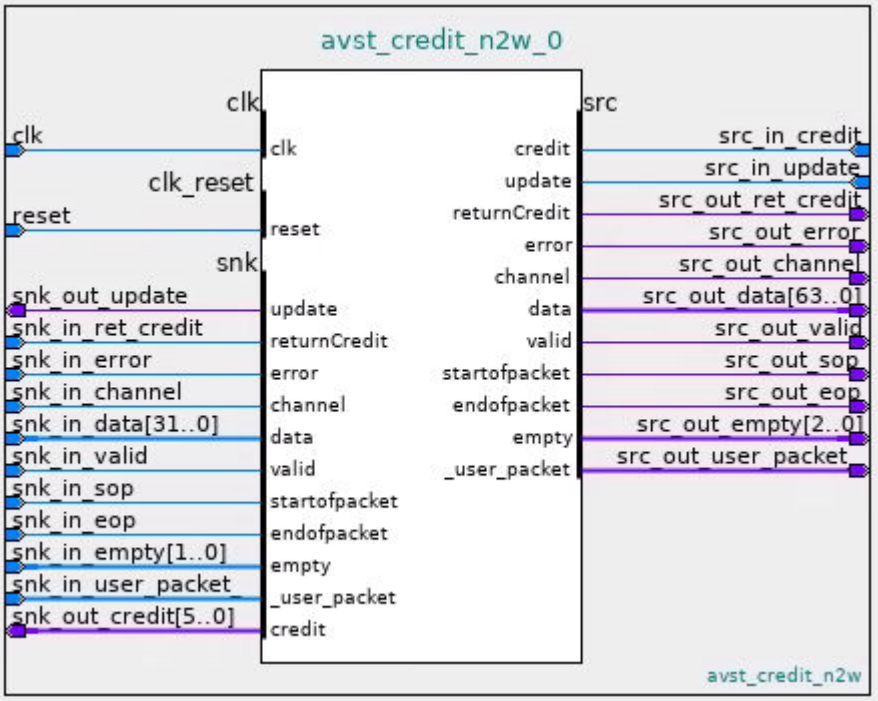
Note: The sink data width must be an integer multiple of the source data width.

Table 73. Avalon Streaming Credit Narrow to Wide Adapter Interface Parameters

Parameter Name	Description	Legal Values
Maximum Credit Allowed (Sink Interface)	Specifies the maximum number of credits allowed by the sink interface. The legal value is 32.	**32**
Number of symbols	Specifies the maximum number of symbols that can transfer.	**1-8192**
Symbol width	Specifies the number of data bits per symbol.	**1-8192**
Width of Channel Port	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is disabled.	**1-128**
Width of Error Port	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is disabled.	**1-1024**
Width of Empty Port	The width of the `empty` signal on the output interfaces. A value of 0 indicates that the empty signal is not in use. This parameter is disabled when **Use Empty** is disabled.	**1-1024**
Maximum Credit Allowed (Source Interface)	Specifies the maximum number of credits allowed by the source interface. Legal values are from 1 to 256.	**1,2,4,8,16,32,64,128, 256**
Use Packets	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.	**On|Off**
Use Empty	Enables or disables the `empty` signal.	**On|Off**
Use Channel	Enables or disables the `channel` signal.	**On|Off**
Use Error	Enables or disables the `error` signal.	**On|Off**
Enable User Signals	Allows you to specify the name, width, and type of user signals.	**On|Off**

5.3.2.2.2. Avalon Streaming Credit Narrow to Wide Adapter Interface Signals

Figure 219. Avalon Streaming Credit Narrow to Wide Adapter Interface Signals



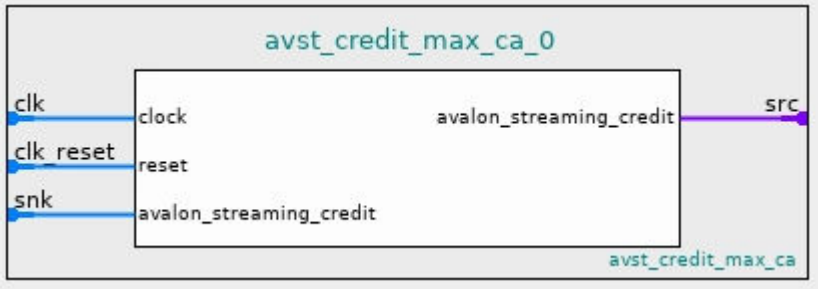
Related Information

Avalon Streaming Credit Interface Signal Roles on page 355

5.3.2.3. Avalon Streaming Credit Max Credit Adapter

The Avalon Streaming Max Credit Adapter facilitates credit flow integrity when the source's **Maximum Credit Allowed** (`maxCredit`) parameter value is less than the sink's **Maximum Credit Allowed** (`maxCredit`) value. The adaptor is available for parameterization as the **Avalon Streaming Credit Max Credit Adapter Intel FPGA IP** from the Platform Designer IP Catalog.

Figure 220. Avalon Streaming Credit Max Credit Adapter



This adapter can accept up to the `maxCredit` of sink credits from the sink, and then supply credits to the source according to the source's `maxCredit` property.

For example, if the source's `maxCredit` is 4 and the sink's `maxCredit` is 8, and the sink provides a value of 8 on the credit bus, the adapter sends 4 credits to the source. When the adapter receives 4 valid beats from the source, the adapter releases the remaining 4 credits to the source. This adapter does not modify data from the source.

5.3.2.3.1. Avalon Streaming Credit Max Credit Adapter Interface Parameters

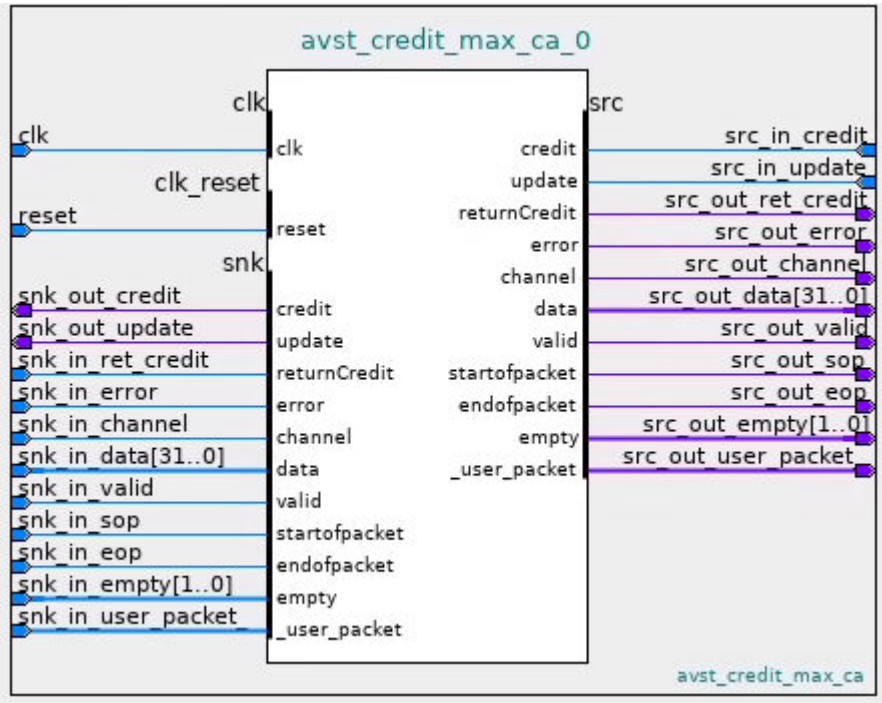
You can specify the following parameters for the Avalon Streaming Credit Max Credit Adapter by double-clicking **Avalon Streaming Credit Max Credit Adapter Intel FPGA IP** in the Platform Designer IP Catalog:

Table 74. Avalon Streaming Credit Max Credit Adapter Interface Parameters

Parameter Name	Description	Legal Values
Maximum Credit Allowed (Sink Interface)	Specifies the maximum number of credits allowed by the sink interface. Legal values are from 1 to 256.	**1,2,4,8,16,32,64,128, 256**
Number of symbols	Specifies the maximum number of symbols that can transfer.	**1-8192**
Symbol width	Specifies the number of data bits per symbol.	**1-8192**
Width of Channel Port	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is disabled.	**1-128**
Width of Error Port	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is disabled.	**1-1024**
Width of Empty Port	The width of the `empty` signal on the output interfaces. A value of 0 indicates that the empty signal is not in use. This parameter is disabled when **Use Empty** is disabled.	**1-1024**
Maximum Credit Allowed (Source Interface)	Specifies the maximum number of credits allowed by the source interface. Legal values are from 1 to 256.	**1,2,4,8,16,32,64,128, 256**
Use Packets	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.	**On|Off**
Use Empty	Enables or disables the `empty` signal.	**On|Off**
Use Channel	Enables or disables the `channel` signal.	**On|Off**
Use Error	Enables or disables the `error` signal.	**On|Off**
Enable User Signals	Allows you to specify the name, width, and type of user signals.	**On|Off**

5.3.2.3.2. Avalon Streaming Credit Max Credit Adapter Interface Signals

Figure 221. Avalon Streaming Credit Max Credit Adapter Interface Signals



Related Information

Avalon Streaming Credit Interface Signal Roles on page 355

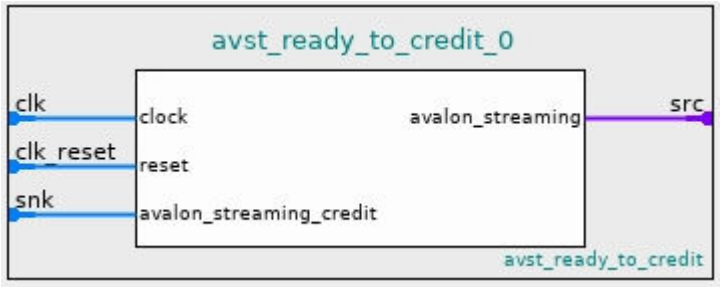
5.3.2.4. Avalon Streaming Ready to Credit Adapter

The Avalon Streaming Ready to Credit Adapter facilitates connectivity between an Avalon Streaming sink and an Avalon Streaming Credit source.

The Avalon Streaming Ready to Credit adapter:

- Outputs credits on the Avalon Streaming Credit sink interface.
- Outputs data on the Avalon Streaming source interface
- Accepts `ready` and asserts `valid` on the Avalon Streaming source interface.
- Supports the `readyLatency` property on the Avalon Streaming sink interface.
- Is available as the **Avalon Streaming Ready to Credit Intel FPGA IP** from the Platform Designer IP Catalog.

Figure 222. Avalon Streaming Ready to Credit Adapter



5.3.2.4.1. Avalon Streaming Ready to Credit Adapter Interface Parameters

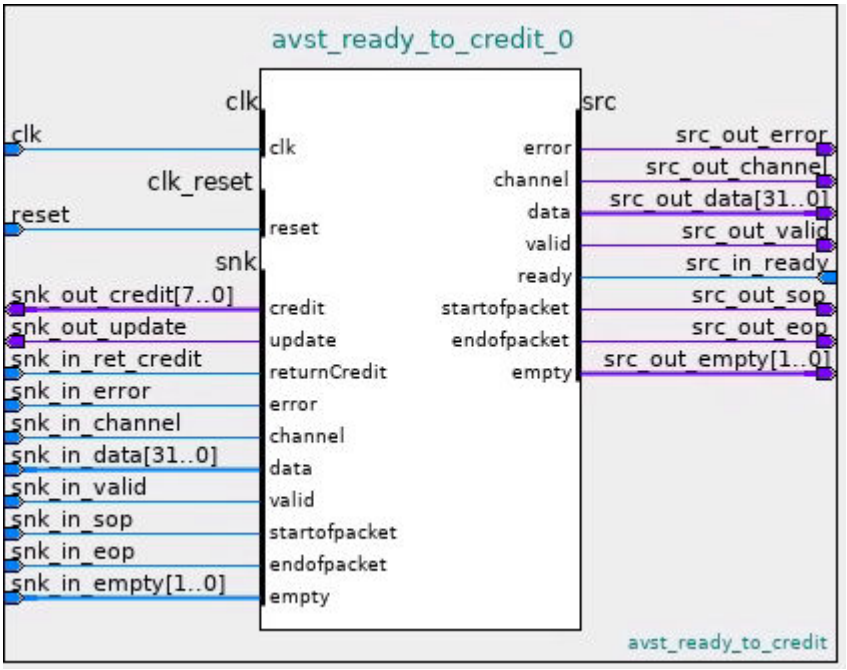
You can specify the following parameters for the Avalon Streaming Ready to Credit Adapter by double-clicking **Avalon Streaming Ready to Credit Intel FPGA IP** in the Platform Designer IP Catalog:

Table 75. Avalon Streaming Ready to Credit Adapter Interface Parameters

Parameter Name	Description	Legal Values
Maximum Credit Allowed (Sink Interface)	Specifies the maximum number of credits allowed by the sink interface. Legal values are from 1 to 256.	**1,2,4,8,16,32,64,128, 256**
Number of symbols	Specifies the maximum number of symbols that can transfer.	**1-8192**
Symbol width	Specifies the number of data bits per symbol.	**1-8192**
Width of Channel Port	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is disabled.	**1-128**
Width of Error Port	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is disabled.	**1-1024**
Width of Empty Port	The width of the `empty` signal on the output interfaces. A value of 0 indicates that the empty signal is not in use. This parameter is disabled when **Use Empty** is disabled.	**1-1024**
Ready Latency	Specifies the ready latency to expect from the sink connected to the module's source interface.	**0-32**
synchronous reset	Specifies that the adapter should have a synchronous reset.	**On|Off**
Use Packets	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.	**On|Off**
Use Empty	Enables or disables the `empty` signal.	**On|Off**
Use Channel	Enables or disables the `channel` signal.	**On|Off**
Use Error	Enables or disables the `error` signal.	**On|Off**

5.3.2.4.2. Avalon Streaming Ready to Credit Adapter Interface Signals

Figure 223. Avalon Streaming Ready to Credit Adapter Interface Signals



Related Information

Avalon Streaming Credit Interface Signal Roles on page 355

5.3.2.5. Avalon Streaming Credit to Ready Adapter

The Avalon Streaming Credit to Ready Adapter facilitates connectivity between an Avalon Streaming Credit sink and an Avalon Streaming source.

The Avalon Streaming Credit to Ready adapter:

- Receives credits on the Avalon Streaming Credit source interface.
- Asserts `ready` and accepts `valid` on the Avalon Streaming sink interface.
- Supports the **Ready Latency** (`readyLatency`) parameter on the Avalon Streaming sink interface.
- Is available as the **Avalon Streaming Credit to Ready Intel FPGA IP** from the Platform Designer IP Catalog.

Figure 224. Avalon Streaming Credit to Ready Adapter

5.3.2.5.1. Avalon Streaming Credit to Ready Adapter Interface Parameters

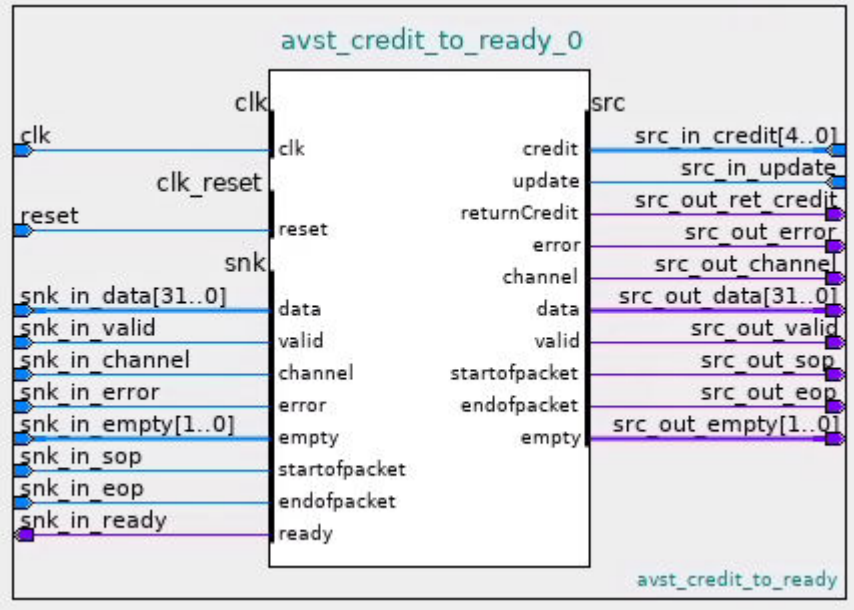
You can specify the following parameters for the Avalon Streaming Credit to Ready adapter by double-clicking **Avalon Streaming Credit to Ready Intel FPGA IP** in the Platform Designer IP Catalog:

Table 76. Avalon Streaming Credit to Ready Adapter Interface Parameters

Parameter Name	Description	Legal Values
Maximum Credit Allowed (Source Interface)	Specifies the maximum number of credits allowed by the source interface. The legal value is 16.	**16**
Number of symbols	Specifies the maximum number of symbols that can transfer.	**1-8192**
Symbol width	Specifies the number of data bits per symbol.	**1-8192**
Width of Channel Port	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is disabled.	**1-128**
Width of Error Port	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is disabled.	**1-1024**
Width of Empty Port	The width of the `empty` signal on the output interfaces. A value of 0 indicates that the empty signal is not in use. This parameter is disabled when **Use Empty** is disabled.	**1-1024**
Ready Latency	Specifies the readyLatency of the source connected to the adapter sink interface.	**0-32**
synchronous reset	Specifies that the adapter should have a synchronous reset.	**On|Off**
Use Packets	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.	**On|Off**
Use Empty	Enables or disables the `empty` signal.	**On|Off**
Use Channel	Enables or disables the `channel` signal.	**On|Off**
Use Error	Enables or disables the `error` signal.	**On|Off**

5.3.2.5.2. Avalon Streaming Credit to Ready Adapter Interface Signals

Figure 225. Avalon Streaming Credit to Ready Adapter Interface Signals



Related Information

Avalon Streaming Credit Interface Signal Roles on page 355

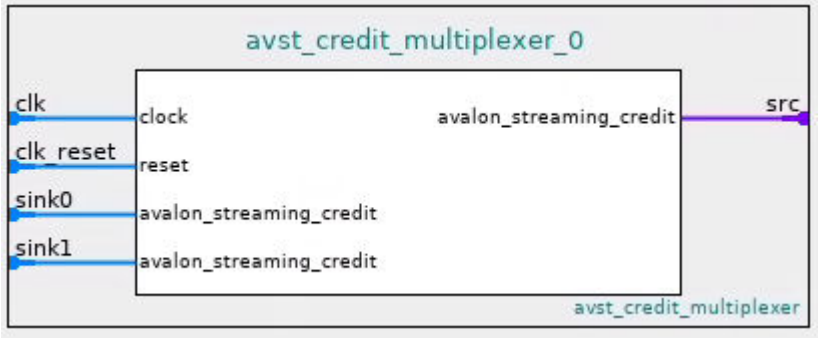
5.3.3. Avalon Streaming Credit Multiplexer

The Avalon Streaming Credit Multiplexer allows multiple sources to access a single sink.

The Avalon Streaming Credit Multiplexer:

- Supports round-robin arbitration.
- Receives credits from the sink, and ensures credits supplied to sources are independent.
- Implements separate credit management logic on both source and sink interfaces.
- Maintains packet integrity from a source.
- Is available as the **Avalon Streaming Credit Multiplexer Intel FPGA IP** from the Platform Designer IP Catalog.

Figure 226. Avalon Streaming Credit Multiplexer



5.3.3.1. Avalon Streaming Credit Multiplexer Parameters

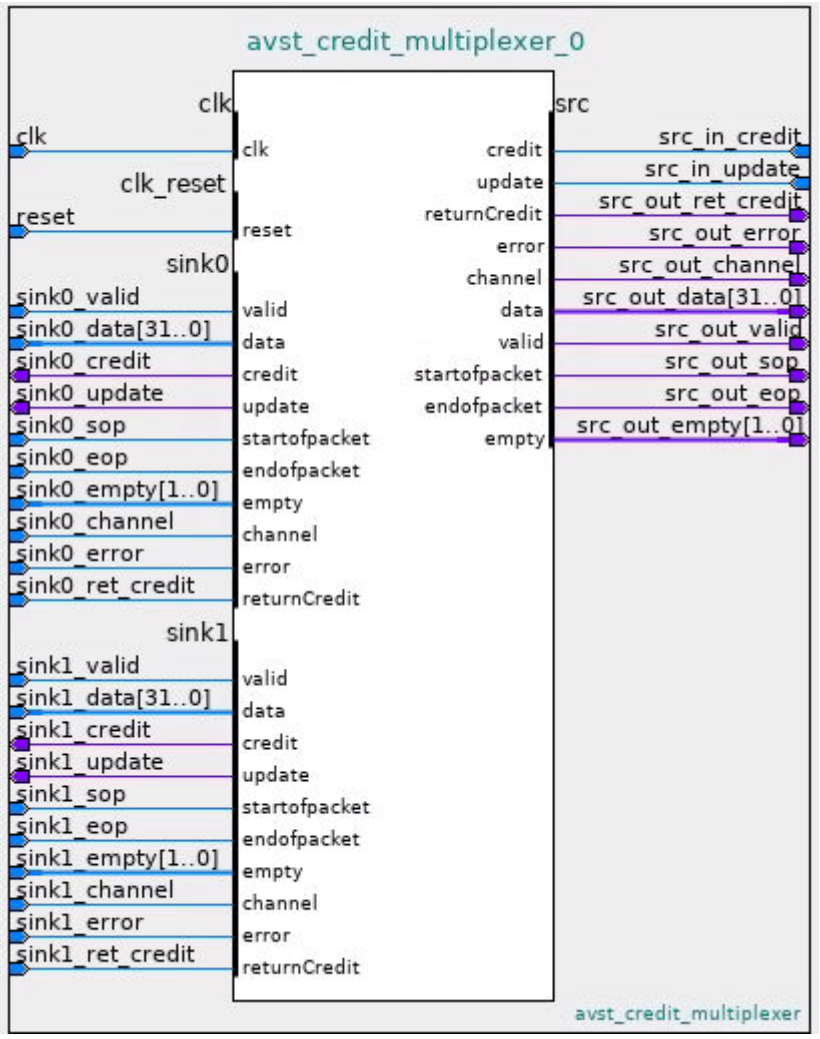
You can specify the following parameters for the Avalon Streaming Credit Multiplexer by double-clicking **Avalon Streaming Credit Multiplexer Intel FPGA IP** in the Platform Designer IP Catalog:

Table 77. Avalon Streaming Credit Multiplexer Parameters

Parameter Name	Description	Legal Values
Number of mux inputs	Specifies the number of inputs to the multiplexer.	N/A
Use Packets	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.	**On|Off**
Use Empty	Enables or disables the `empty` signal.	**On|Off**
Use Channel	Enables or disables the `channel` signal.	**On|Off**
Use Error	Enables or disables the `error` signal.	**On|Off**
Maximum Credit Allowed (Source Interface)	Specifies the maximum number of credits allowed by the source interface. Legal values are from 1 to 256.	**1,2,4,8,16,32,64,128,256**
Number of symbols	Specifies the maximum number of symbols that can transfer.	**1-8192**
Symbol width	Specifies the number of data bits per symbol.	**1-8192**
Width of Channel Port	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is disabled.	**1-128**
Width of Error Port	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is disabled.	**1-1024**
Width of Empty Port	The width of the `empty` signal on the output interfaces. A value of 0 indicates that the empty signal is not in use. This parameter is disabled when **Use Empty** is disabled.	**1-1024**
Use channel on Sink interface	Enables or disables the `channel` signal on the sink interlace.	**On|Off**
Sink channel width	Specifies the width of the sink channel when **Use channel on Sink interface** is **On**.	**1-128**

5.3.3.2. Avalon Streaming Credit Multiplexer Interface Signals

Figure 227. Avalon Streaming Credit Multiplexer Interface Signals



Related Information

Avalon Streaming Credit Interface Signal Roles on page 355

5.3.4. Avalon Streaming Credit Demultiplexer

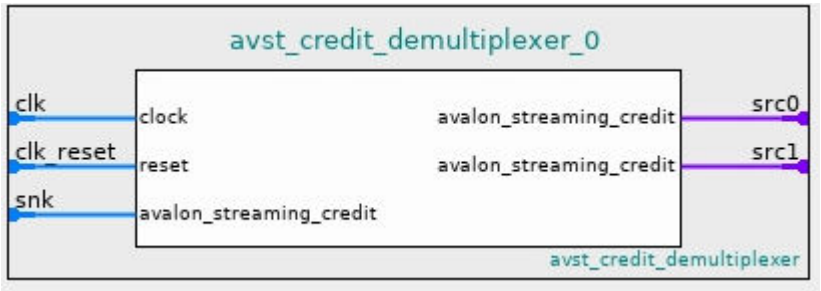
The Avalon Streaming Credit Demultiplexer allows one source to access multiple sinks.

The Avalon Streaming Credit Demultiplexer:

- Must have a channel signal from the source as an input.
- The channel input from the source must be at least `ceil(log2 (NUM_SINKS))` wide, where `NUM_SINKS` is the number of sinks.

  For example, if the source connects to 6 sinks, the channel signal from the source must be at least 3 bits. The least significant `ceil(log2(NUM_SINKS))` bits route data from the source to the appropriate sink. For example, if the demultiplexer connects to 5 sinks (`sink0`, `sink1`, `sink2`, `sink3`, `sink4`), and the channel signal is 8 bits wide, with a value on the channel bus of `00010010`, data routes to `sink2`.
- The source is responsible for maintaining packet integrity to any sink.
- During a packet transmission, the source must not change the lower order channel bits. Otherwise, the sinks may receive incomplete or invalid packets.
- Is available as the **Avalon Streaming Credit Demultiplexer Intel FPGA IP** from the Platform Designer IP Catalog.

Figure 228. Avalon Streaming Credit Demultiplexer



5.3.4.1. Avalon Streaming Credit Demultiplexer Parameters

You can specify the following parameters for the Avalon Streaming Credit Demultiplexer by double-clicking **Avalon Streaming Credit Demultiplexer Intel FPGA IP** in the Platform Designer IP Catalog:

Table 78. Avalon Streaming Credit Demultiplexer Parameters

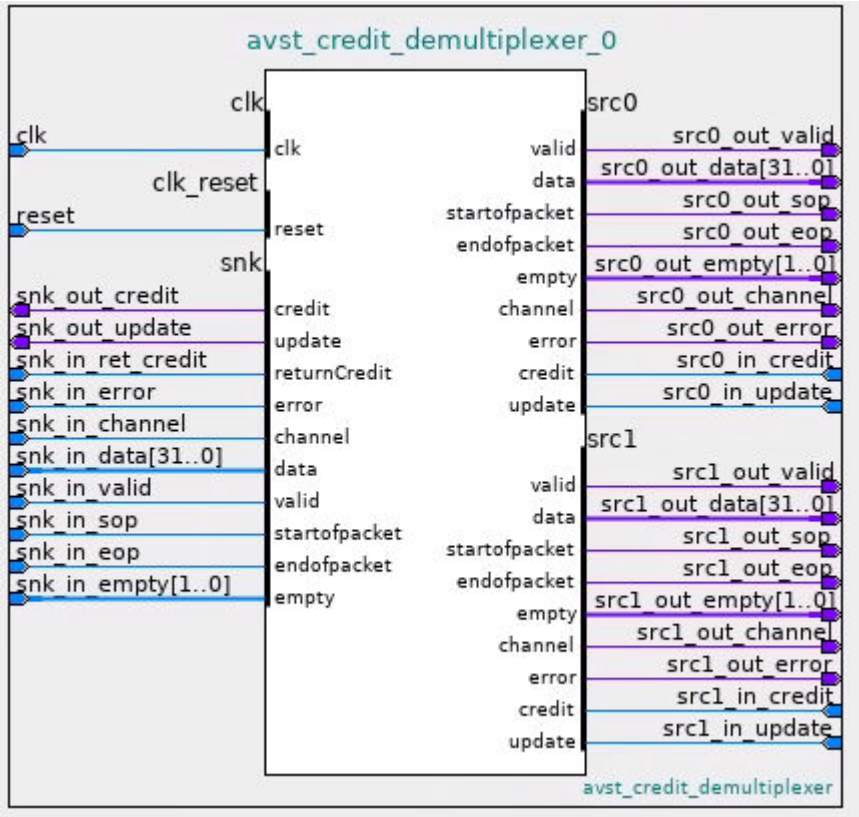
Parameter Name	Description	Legal Values
Number of demux outputs	Specifies the number of outputs from the demultiplexer.	N/A
Maximum Credit Allowed (Sink Interface)	Specifies the maximum number of credits allowed by the sink interface. Legal values are from 1 to 256.	**1,2,4,8,16,32,64,128,256**
Number of symbols	Specifies the maximum number of symbols that can transfer.	**1-8192**
Symbol width	Specifies the number of data bits per symbol.	**1-8192**
Width of Error Port	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is disabled.	**1-1024**

continued...

Parameter Name	Description	Legal Values
Width of Empty Port	The width of the `empty` signal on the output interfaces. A value of 0 indicates that the empty signal is not in use. This parameter is disabled when **Use Empty** is disabled.	**1-1024**
Use Channel	Enables the `channel` signal. (always on)	**On**
Width of Channel Port	The width of the `channel` signal on the sink interface based on the number of demux outputs.	**1-128**
Use Packets	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.	**On|Off**
Use Empty	Enables or disables the `empty` signal.	**On|Off**
Use Error	Enables or disables the `error` signal.	**On|Off**
Enable User Signals	Allows you to specify the name, width, and type of user signals.	**On|Off**
Use channel on Source interface	Enables or disables the `channel` signal on the source interface.	**On|Off**
Source channel width	Specifies the width of the source channel when **Use channel on Source interface** is **On**.	**1-128**

5.3.4.2. Avalon Streaming Credit Demultiplexer Interface Signals

Figure 229. Avalon Streaming Credit Demultiplexer Interface Signals



Related Information

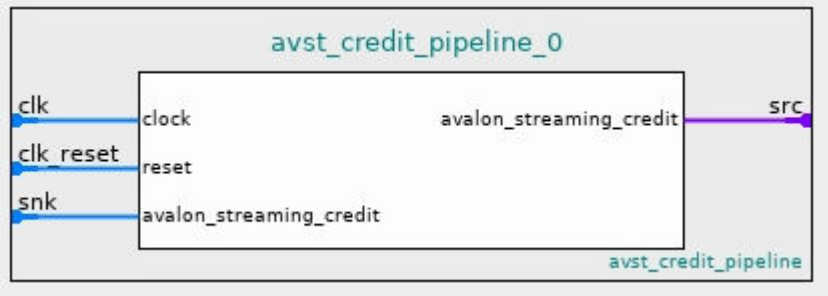
Avalon Streaming Credit Interface Signal Roles on page 355

5.3.5. Avalon Streaming Credit Pipeline Bridge

The Avalon Streaming Credit Pipeline Bridge allows you to insert an arbitrary number of pipeline stages on both the credit and data paths. The number of pipeline stages on the credit path, from sink to source, need not be equal to the number of pipeline stages on the data path, from source to sink.

This bridge is available as the **Avalon Streaming Credit Pipeline Bridge Intel FPGA IP** from the Platform Designer IP Catalog.

Figure 230. Avalon Streaming Credit Pipeline Bridge



5.3.5.1. Avalon Streaming Credit Pipeline Bridge Parameters

You can specify the following parameters for the Avalon Streaming Credit Pipeline Bridge by double-clicking **Avalon Streaming Credit Pipeline Bridge Intel FPGA IP** in the Platform Designer IP Catalog:

Table 79. Avalon Streaming Credit Pipeline Bridge Parameters

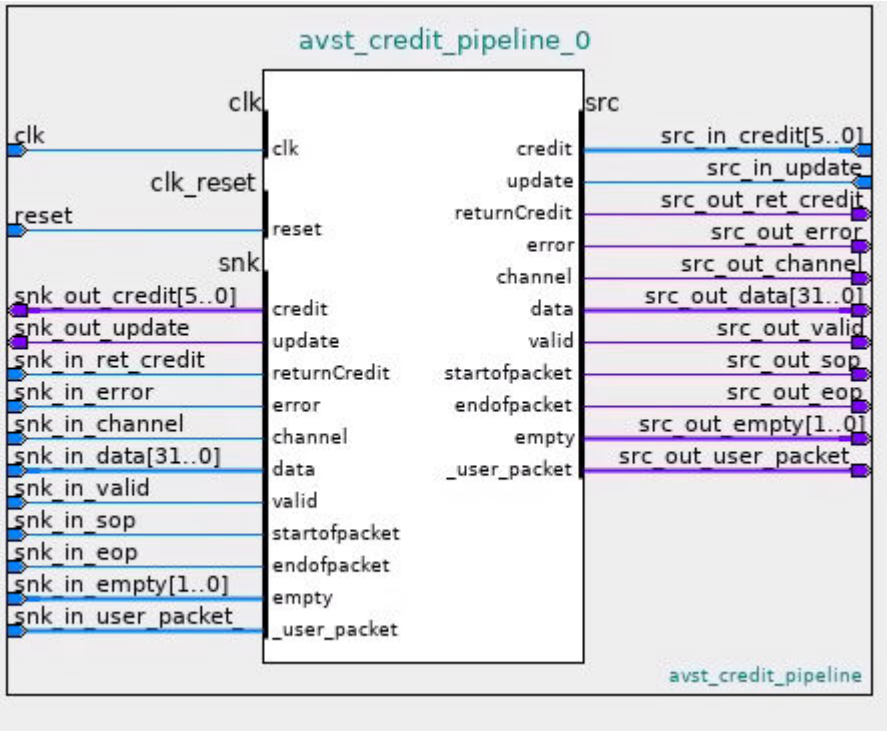
Parameter Name	Description	Legal Values
Data Pipeline Depth	Specifies the depth (size) of the data pipeline.	**1-16**
Credit Pipeline Depth	Specifies the depth (size) of the credit pipeline.	**1-16**
Maximum Credit Allowed (Source Interface)	Specifies the maximum number of credits allowed by the source interface. Legal values are from 1 to 256.	**1,2,4,8,16,32,64,128, 256**
Number of symbols	Specifies the maximum number of symbols that can transfer.	**1-8192**
Symbol width	Specifies the number of data bits per symbol.	**1-8192**
Width of Channel Port	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is disabled.	**1-128**
Width of Error Port	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is disabled.	**1-1024**
Width of Empty Port	The width of the `empty` signal on the output interfaces. A value of 0 indicates that the empty signal is not in use. This parameter is disabled when **Use Empty** is disabled.	**1-1024**

continued...

Parameter Name	Description	Legal Values
Use Packets	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.	**On|Off**
Use Empty	Enables or disables the `empty` signal.	**On|Off**
Use Channel	Enables or disables the `channel` signal.	**On|Off**
Use Error	Enables or disables the `error` signal.	**On|Off**
Enable User Signals	Allows you to specify the name, width, and type of user signals.	**On|Off**

5.3.5.2. Avalon Streaming Credit Pipeline Bridge Interface Signals

Figure 231. Avalon Streaming Credit Pipeline Bridge Interface Signals



Related Information

Avalon Streaming Credit Interface Signal Roles on page 355

5.4. Interrupt Interfaces

Using individual requests, the interrupt logic can process up to 2048 IRQ inputs connected to each interrupt receiver. With this logic, the interrupt sender connected to interrupt `receiver0` is the highest priority with sequential receivers being successively lower priority. You can redefine the priority of interrupt senders by instantiating the IRQ mapper component. For more information, refer to IRQ Mapper on page 309.

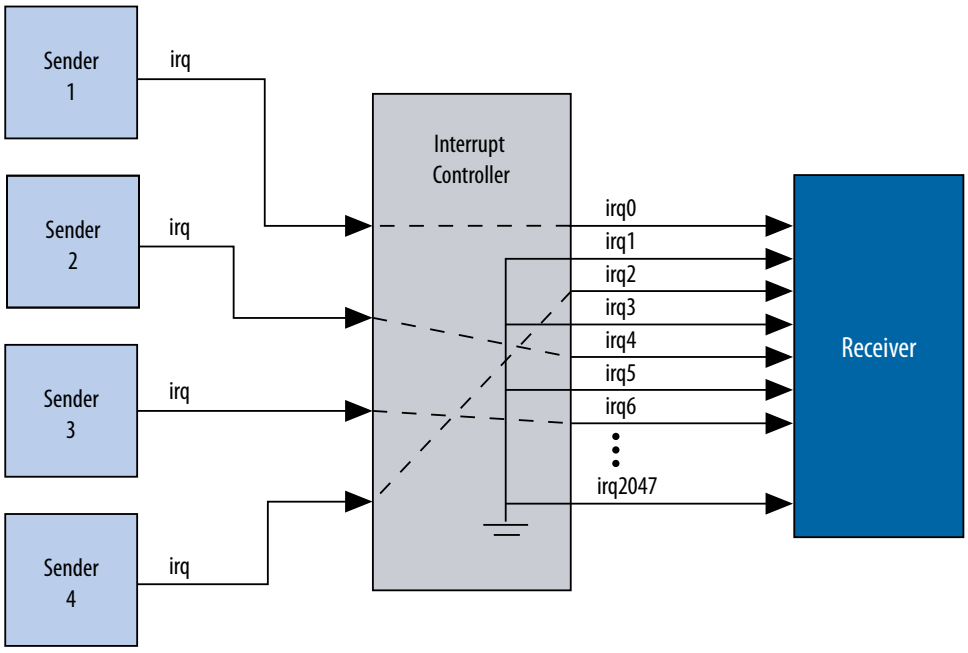
You can define the interrupt sender interface as asynchronous with no associated clock or reset interfaces. You can also define the interrupt receiver interface as asynchronous with no associated clock or reset interfaces. As a result, the receiver does its own synchronization internally. Platform Designer does not insert interrupt synchronizers for such receivers.

For clock crossing adaptation on interrupts, Platform Designer inserts a synchronizer. This synchronizer is clocked with the interrupt end point interface clock when the corresponding starting point interrupt interface has no clock or a different clock (than the end point). Platform Designer inserts the adapter if there is any kind of mismatch between the start and end points. Platform Designer does not insert the adapter if the interrupt receiver does not have an associated clock.

5.4.1. Individual Requests IRQ Scheme

In the individual requests IRQ scheme, Platform Designer interconnect passes IRQs directly from the sender to the receiver, without making assumptions about IRQ priority. If multiple senders assert their IRQs simultaneously, the receiver logic determines which IRQ has highest priority, and then responds appropriately.

Figure 232. Interrupt Controller Mapping IRQs



Using individual requests, the interrupt controller can process up to 2048 IRQ inputs. The interrupt controller generates a 2048-bit signal `irq[2047:0]` to the receiver, and maps agent IRQ signals to the bits of `irq[2047:0]`. Any unassigned bits of `irq[2047:0]` are disabled.

5.4.2. Assigning IRQs in Platform Designer

You assign IRQ connections on the **System View** tab of Platform Designer. After adding all components to the system, you connect interrupt senders and receivers.

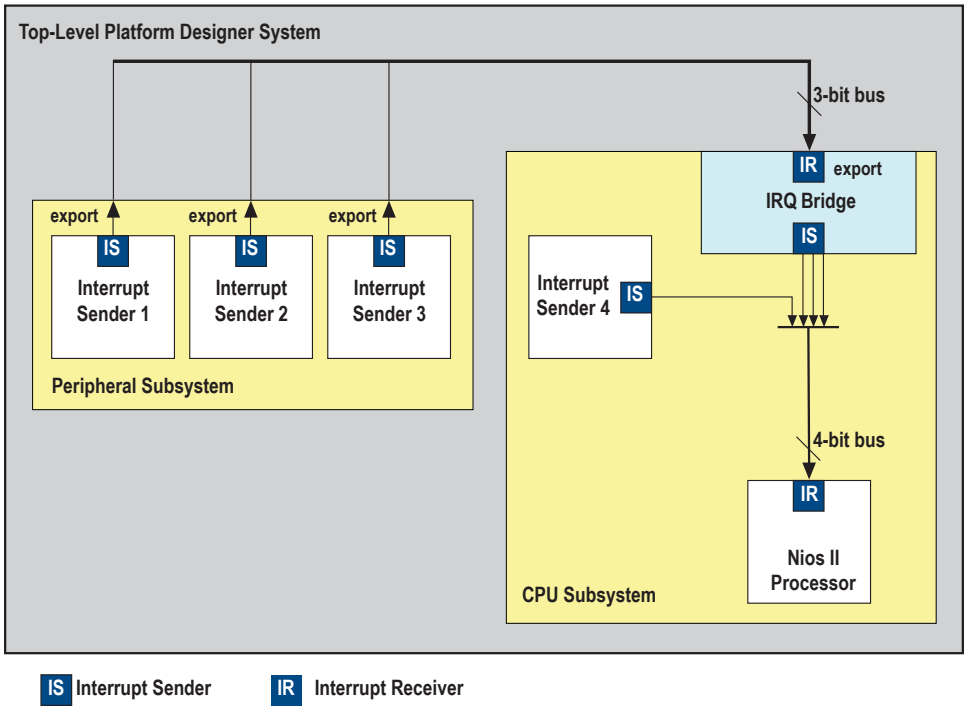
You can use the **IRQ** column to specify an IRQ number with respect to each receiver, or to specify a receiver's IRQ as unconnected. Platform Designer uses the following components to implement interrupt handling: IRQ Bridge, IRQ Mapper, IRQ Clock Crosser, and IRQ Fanout.

5.4.2.1. IRQ Bridge

The IRQ Bridge Intel FPGA IP allows you to route interrupt wires between Platform Designer subsystems.

Figure 233. Platform Designer IRQ Bridge Application

The peripheral subsystem example below has three interrupt senders that are exported to the top-level of the subsystem. The interrupts are then routed to the CPU subsystem using the IRQ bridge.



Note: Nios II BSP tools support the IRQ Bridge. Interrupts connected via an IRQ Bridge appear in the generated `system.h` file. You can use the following properties with the IRQ Bridge, which do not effect Platform Designer interconnect generation. Platform Designer uses these properties to generate the correct IRQ information for downstream tools:

- `set_interface_property <sender port> bridgesToReceiver <receiver port>`— The <*sender port*> of the IP generates a signal that is received on the IP's <*receiver port*>. Sender ports are single bits. Receive ports can be multiple bits. Platform Designer requires the `bridgedReceiverOffset` property to identify the <*receiver port*> bit that the <*sender port*> sends.
- `set_interface_property <sender port> bridgedReceiverOffset <port number>`— Indicates the <*port number*> of the receiver port that the <*sender port*> sends.

5.4.2.2. IRQ Mapper

Platform Designer inserts the IRQ Mapper Intel FPGA IP automatically during generation. The IRQ Mapper converts individual interrupt wires to a bus, and then maps the appropriate IRQ priority number onto the bus.

You can specify the following parameter values:

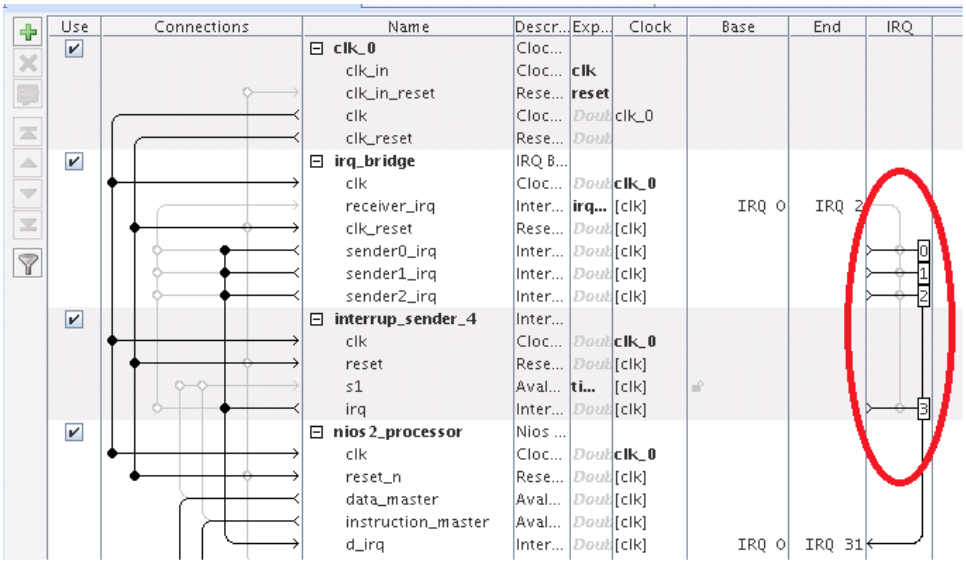
Table 80. IRQ Mapper Parameters

Parameter	Values	Description
Number of receivers	1-2048	Specifies the number of Avalon receiver signals on the bus.
Sender interrupt width	1-2048	Specifies the width of the Avalon sender signal bus.
Use synchronous resets	On|Off	Resets signal synchronously. This option is Off by default.
Remove Clock and Reset Ports	On|Off	Removes the clock and reset ports from the component. If both **Use synchronous resets** and **Remove Clock and Reset Ports** are on, Platform Designer displays an error.

By default, the interrupt sender connected to the `receiver0` interface of the IRQ mapper is the highest priority, and sequential receivers are successively lower priority. You can modify the interrupt priority of each IRQ wire by modifying the IRQ priority number in Platform Designer under the **IRQ** column. The modified priority is reflected in the **IRQ_MAP** parameter for the auto-inserted IRQ Mapper.

Figure 234. IRQ Column in Platform Designer

Circled in the **IRQ** column are the default interrupt priorities allocated for the CPU subsystem.



5.4.2.3. IRQ Clock Crosser

The IRQ Clock Crosser Intel FPGA IP synchronizes interrupt senders and receivers that are in different clock domains. To use this component, connect the clocks for both the interrupt sender and receiver, and for both the interrupt sender and receiver interfaces. Platform Designer automatically inserts this component when it is required.

5.4.2.4. IRQ Fanout

The IRQ Fanout Intel FPGA IP allows you to specify a value for the **Number of senders** parameter that controls the number of interrupt senders exported to the top level of the system. Platform Designer automatically inserts this component when the system requires.

5.5. Clock Interfaces

Clock interfaces define the clocks used by a component. Components can have clock inputs, clock outputs, or both. To update the clock frequency of the component, use the **Parameters** tab for the clock source.

The **Clock Source** parameters allows you to set the following options:

- **Clock frequency**—the frequency of the output clock from this clock source.
- **Clock frequency is known**— when turned on, the clock frequency is known. When turned off, the frequency is set from outside the system. Turning off this option is useful for cases when a subsystem receives a clock from a higher level system.

  Note: If turned off, system generation may fail because the components do not receive the necessary clock information. For best results, turn this option on before system generation.

For more information about synchronous design practices, refer to *Quartus Prime Pro Edition User Guide: Design Recommendations*.

5.5.1. (High Speed Serial Interface) HSSI Clock Interfaces

You can use HSSI Serial Clock and HSSI Bonded Clock interfaces in Platform Designer to enable high speed serial connectivity between clocks that are used by certain IP protocols.

5.5.1.1. HSSI Serial Clock Interface

You can connect the HSSI Serial Clock interface with only similar type of interfaces, for example, you can connect a HSSI Serial Clock Source interface to a HSSI Serial Clock Sink interface.

5.5.1.1.1. HSSI Serial Clock Source

The HSSI Serial Clock interface includes a source in the **Start** direction.

You can instantiate the HSSI Serial Clock Source interface in the **_hw.tcl** file as:

```
add_interface <name> hssi_serial_clock start
```

You can connect the HSSI Serial Clock Source to multiple HSSI Serial Clock Sinks because the HSSI Serial Clock Source supports multiple fan-outs. This Interface has a single **clk** port role limited to a 1 bit width, and a **clockRate** parameter, which is the frequency of the clock driven by the HSSI Serial Clock Source interface.

An unconnected and unexported HSSI Serial Source is valid and does not generate error messages.

Table 81. HSSI Serial Clock Source Port Roles

Name	Direction	Width	Description
clk	Output	1 bit	A single bit wide port role, which provides synchronization for internal logic.

Table 82. HSSI Serial Clock Source Parameters

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven byte HSSI Serial Clock Source interface.

5.5.1.1.2. HSSI Serial Clock Sink

The HSSI Serial Clock interface includes a sink in the **End** direction.

You can instantiate the HSSI Serial Clock Sink interface in the **_hw.tcl** file as:

```
add_interface <name> hssi_serial_clock end
```

You can connect the HSSI Serial Clock Sink interface to a single HSSI Serial Clock Source interface; you cannot connect it to multiple sources. This Interface has a single **clk** port role limited to a 1 bit width, and a **clockRate** parameter, which is the frequency of the clock driven by the HSSI Serial Clock Source interface.

An unconnected and unexported HSSI Serial Sink is invalid and generates error messages.

Table 83. HSSI Serial Clock Sink Port Roles

Name	Direction	Width	Description
clk	Output	1	A single bit wide port role, which provides synchronization for internal logic

Table 84. HSSI Serial Clock Sink Parameters

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven by the HSSI Serial Clock Source interface. When you specify a **clockRate** greater than 0, then this interface can be driven only at that rate.

5.5.1.1.3. HSSI Serial Clock Connection

The HSSI Serial Clock Connection defines a connection between a HSSI Serial Clock Source connection point, and a HSSI Serial Clock Sink connection point.

A valid HSSI Serial Clock Connection exists when all the following criteria are satisfied. If the following criteria are not satisfied, Platform Designer generates error messages and the connection is prohibited.

- The starting connection point is an HSSI Serial Clock Source with a single port role **clk** and maximum 1 bit in width. The direction of the starting port is **Output**.
- The ending connection point is an HSSI Serial Clock Sink with a single port role **clk**, and maximum 1 bit in width. The direction of the ending port is **Input**.
- If the parameter, **clockRate** of the HSSI Serial Clock Sink is greater than 0, the connection is only valid if the **clockRate** of the HSSI Serial Clock Source is the same as the **clockRate** of the HSSI Serial Clock Sink.

5.5.1.1.4. HSSI Serial Clock Example

Example 23. HSSI Serial Clock Interface Example

You can make connections to declare the HSSI Serial Clock interfaces in the **_hw.tcl**.

```
package require -exact qsys 14.0

set_module_property name hssi_serial_component
set_module_property ELABORATION_CALLBACK elaborate

add_fileset QUARTUS_SYNTH QUARTUS_SYNTH generate
add_fileset SIM_VERILOG SIM_VERILOG generate
add_fileset SIM_VHDL SIM_VHDL generate

set_fileset_property QUARTUS_SYNTH TOP_LEVEL \
"hssi_serial_component"

set_fileset_property SIM_VERILOG TOP_LEVEL "hssi_serial_component"
set_fileset_property SIM_VHDL TOP_LEVEL "hssi_serial_component"

proc elaborate {} {
    # declaring HSSI Serial Clock Source
    add_interface my_clock_start hssi_serial_clock start
    set_interface_property my_clock_start  ENABLED true

    add_interface_port my_clock_start  hssi_serial_clock_port_out \
 clk Output 1

    # declaring HSSI Serial Clock Sink
    add_interface my_clock_end hssi_serial_clock end
    set_interface_property my_clock_end  ENABLED true
```

```
    add_interface_port my_clock_end  hssi_serial_clock_port_in clk \
 Input 1
}

proc generate { output_name } {

    add_fileset_file hssi_serial_component.v VERILOG PATH \
 "hssi_serial_component.v"
}
```

Example 24. HSSI Serial Clock Instantiated in a Composed Component

If you use the components in a hierarchy, for example, instantiated in a composed component, you can declare the connections as illustrated in this example.

```
add_instance myinst1 hssi_serial_component
add_instance myinst2 hssi_serial_component
# add connection from source of myinst1 to sink of myinst2

add_connection myinst1.my_clock_start myinst2.my_clock_end \
hssi_serial_clock

# adding connection from source of myinst2 to sink of myinst1

add_connection myinst2.my_clock_start myinst2.my_clock_end \
hssi_serial_clock
```

5.5.1.2. HSSI Bonded Clock Interface

You can connect the HSSI Bonded Clock interface only with similar type of interfaces, for example, you can connect a HSSI Bonded Clock Source interface to a HSSI Bonded Clock Sink interface.

5.5.1.2.1. HSSI Bonded Clock Source

The HSSI Bonded Clock interface includes a source in the **Start** direction.

You can instantiate the HSSI Bonded Clock Source interface in the **_hw.tcl** file as:

```
add_interface <name> hssi_bonded_clock start
```

You can connect the HSSI Bonded Clock Source to multiple HSSI Bonded Clock Sinks because the HSSI Serial Clock Source supports multiple fanouts. This Interface has a single **clk** port role limited to a width range of 1 to 1024 bits. The HSSI Bonded Clock Source interface has two parameters: **clockRate** and **serializationFactor**. **clockRate** is the frequency of the clock driven by the HSSI Bonded Clock Source interface, and the **serializationFactor** is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the required frequency and phases of the individual clocks within the HSSI Bonded Clock interface

An unconnected and unexported HSSI Bonded Source is valid, and does not generate error messages.

Table 85. HSSI Bonded Clock Source Port Roles

Name	Direction	Width	Description
clk	Output	1 to 24 bits	A multiple bit wide port role which provides synchronization for internal logic.

Table 86. HSSI Bonded Clock Source Parameters

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven byte HSSI Serial Clock Source interface.
serialization	long	0	No	The serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the necessary frequency and phases of the individual clocks within the HSSI Bonded Clock interface.

5.5.1.2.2. HSSI Bonded Clock Sink

The HSSI Bonded Clock interface includes a sink in the **End** direction.

You can instantiate the HSSI Bonded Clock Sink interface in the **_hw.tcl** file as:

```
add_interface <name> hssi_bonded_clock end
```

You can connect the HSSI Bonded Clock Sink interface to a single HSSI Bonded Clock Source interface; you cannot connect it to multiple sources. This Interface has a single **clk** port role limited to a width range of 1 to 1024 bits. The HSSI Bonded Clock Source interface has two parameters: **clockRate** and **serializationFactor**. **clockRate** is the frequency of the clock driven by the HSSI Bonded Clock Source interface, and the serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the required frequency and phases of the individual clocks within the HSSI Bonded Clock interface

An unconnected and unexported HSSI Bonded Sink is invalid and generates error messages.

Table 87. HSSI Bonded Clock Source Port Roles

Name	Direction	Width	Description
clk	Output	1 to 24 bits	A multiple bit wide port role which provides synchronization for internal logic.

Table 88. HSSI Bonded Clock Source Parameters

Name	Type	Default	Derived	Description
clockRate	long	0	No	The frequency of the clock driven byte HSSI Serial Clock Source interface.
serialization	long	0	No	The serialization factor is the parallel data width that operates the HSSI TX serializer. The serialization factor determines the necessary frequency and phases of the individual clocks within the HSSI Bonded Clock interface.

5.5.1.2.3. HSSI Bonded Clock Connection

The HSSI Bonded Clock Connection defines a connection between a HSSI Bonded Clock Source connection point, and a HSSI Bonded Clock Sink connection point.

A valid HSSI Bonded Clock Connection exists when all the following criteria are satisfied. If the following criteria are not satisfied, Platform Designer generates error messages and the connection is prohibited.

- The starting connection point is an HSSI Bonded Clock Source with a single port role **clk** with a width range of 1 to 24 bits. The direction of the starting port is **Output**.
- The ending connection point is an HSSI Bonded Clock Sink with a single port role **clk** with a width range of 1 to 24 bits. The direction of the ending port is **Input**.
- The width of the starting connection point **clk** must be the same as the width of the ending connection point.
- If the parameter, **clockRate** of the HSSI Bonded Clock Sink greater than 0, then the connection is only valid if the **clockRate** of the HSSI Bonded Clock Source is same as the **clockRate** of the HSSI Bonded Clock Sink.
- If the parameter, **serializationFactor** of the HSSI Bonded Clock Sink is greater than 0, Platform Designer generates a warning if the **serializationFactor** of HSSI Bonded Clock Source is not same as the **serializationFactor** of the HSSI Bonded Clock Sink.

5.5.1.2.4. HSSI Bonded Clock Example

Example 25. HSSI Bonded Clock Interface Example

You can make connections to declare the HSSI Bonded Clock interfaces in the **_hw.tcl** file.

```
package require -exact qsys 14.0

set_module_property name hssi_bonded_component
set_module_property ELABORATION_CALLBACK elaborate

add_fileset synthesis QUARTUS_SYNTH generate
add_fileset verilog_simulation SIM_VERILOG generate

set_fileset_property synthesis TOP_LEVEL "hssi_bonded_component"

set_fileset_property verilog_simulation TOP_LEVEL \
"hssi_bonded_component"

proc elaborate {} {
    add_interface my_clock_start hssi_bonded_clock start
    set_interface_property my_clock_start  ENABLED true

    add_interface_port my_clock_start  hssi_bonded_clock_port_out \
 clk Output 1024

    add_interface my_clock_end hssi_bonded_clock end
    set_interface_property my_clock_end  ENABLED true

    add_interface_port my_clock_end  hssi_bonded_clock_port_in \
 clk Input 1024
}

proc generate { output_name } {
    add_fileset_file hssi_bonded_component.v VERILOG PATH \
 "hssi_bonded_component.v"}
```

If you use the components in a hierarchy, for example, instantiated in a composed component, you can declare the connections as illustrated in this example.

Example 26. HSII Bonded Clock Instantiated in a Composed Component

```
add_instance myinst1 hssi_bonded_component
add_instance myinst2 hssi_bonded_component
# add connection from source of myinst1 to sink of myinst2

add_connection myinst1.my_clock_start myinst2.my_clock_end \
hssi_bonded_clock

# adding connection from source of myinst2 to sink of myinst1

add_connection myinst2.my_clock_start myinst2.my_clock_end \
hssi_bonded_clock
```

5.6. Reset Interfaces

Reset interfaces provide both soft and hard reset functionality. Soft reset logic typically re-initializes registers and memories without powering down the device. Hard reset logic initializes the device after power-on.

You can define separate reset sources for each clock domain, a single reset source for all clocks, or any combination in between. You can choose to create a single global reset domain by clicking **System ➤ Create Global Reset Network**. If your design requires more than one reset domain, you can implement your own reset logic and connectivity. The IP Catalog includes a reset controller, reset sequencer, and a reset bridge to implement the reset functionality. You can also design your own reset logic.

Platform Designer interconnect now supports synchronous reset of registers in the interconnect. Use of synchronous reset can result in higher performance for Stratix 10 designs because although Stratix 10 Hyper-Registers lack a reset signal, they can make use of the synchronous reset from an adjacent LAB. If a register in your Stratix 10 design uses asynchronous reset, the Compiler cannot implement the register as a Hyper-Register, potentially reducing performance.

When **Use synchronous reset** is set to **True** in the **Domains** tab, all registers in the interconnect use synchronous reset. The **Use synchronous reset** option is enabled by default for Stratix 10 designs, but is disabled by default for all other designs.

Note: If you design your own reset circuitry, you must carefully consider situations which may result in system lockup. For example, if an Avalon memory mapped agent is reset in the middle of a transaction, the Avalon memory mapped host may lockup.

Related Information

Specifying Interconnect Parameters on page 71

5.6.1. Single Global Reset Signal Implemented by Platform Designer

When you select **System ➤ Create Global Reset Network**, the Platform Designer interconnect creates a global reset bus. All the reset requests are ORed together, synchronized to each clock domain, and fed to the reset inputs. The duration of the reset signal is at least one clock period.

The Platform Designer interconnect inserts the system-wide reset under the following conditions:

- The global reset input to the Platform Designer system is asserted.
- Any component asserts its `resetrequest` signal.

5.6.2. Reset Controller

Platform Designer automatically inserts a reset controller block if the input reset source does not have a reset request, but the connected reset sink requires a reset request.

The Reset Controller has the following parameters that you can specify to customize its behavior:

- **Number of inputs**— Indicates the number of individual reset interfaces the controller ORs to create a signal reset output.
- **Output reset synchronous edges**—Specifies the level of synchronization. You can select one the following options:
  - **None**—The reset is asserted and deasserted asynchronously. You can use this setting if you have designed internal synchronization circuitry that matches the reset style required for the IP in the system.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously and asserted asynchronously.
- **Synchronization depth**—Specifies the number of register stages the synchronizer uses to eliminate the propagation of metastable events.
- **Reset request**—Enables reset request generation, which is an early signal that is asserted before reset assertion. The reset request is used by blocks that require protection from asynchronous inputs, for example, M20K blocks.

Platform Designer automatically inserts reset synchronizers under the following conditions:

- More than one reset source is connected to a reset sink
- There is a mismatch between the reset source's synchronous edges and the reset sinks' synchronous edges

5.6.3. Reset Bridge

The Reset Bridge allows you to use a reset signal in two or more subsystems of your Platform Designer system. You can connect one reset source to local components, and export one or more to other subsystems, as required.

The Reset Bridge parameters are used to describe the incoming reset and include the following options:

- **Active low reset**—When turned on, reset is asserted low.
- **Synchronous edges**—Specifies the level of synchronization and includes the following options:
  - **None**—The reset is asserted and deasserted asynchronously. Use this setting if you have internal synchronization circuitry.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously, and asserted asynchronously.
- **Number of reset outputs**—The number of reset interfaces that are exported.

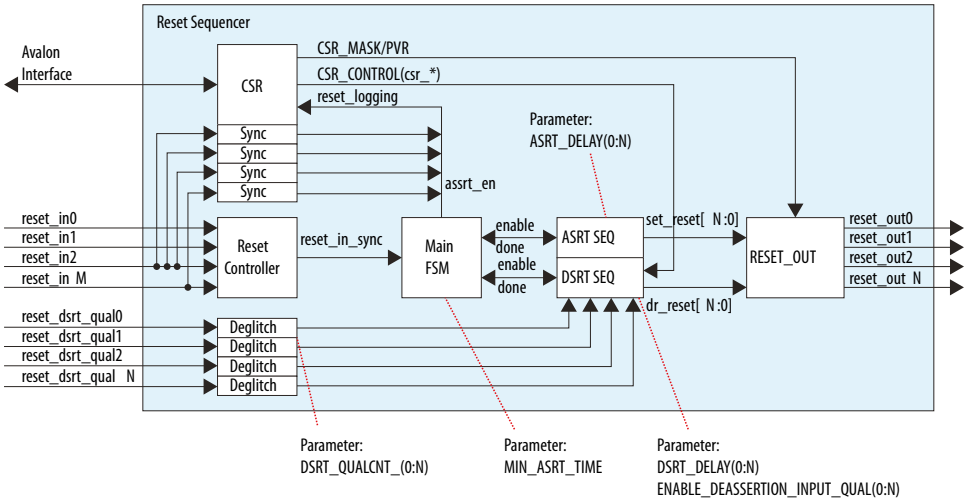
Note: Platform Designer supports multiple reset sink connections to a single reset source interface. However, there are situations in composed systems where an internally generated reset must be exported from the composed system in addition to being used to connect internal components. In this situation, you must declare one reset output interface as an export, and use another reset output to connect internal components.

5.6.4. Reset Sequencer

The Reset Sequencer allows you to control the assertion and deassertion sequence for Platform Designer system resets.

The Parameter Editor displays the expected assertion and deassertion sequences based on the current settings. You can connect multiple reset sources to the reset sequencer, and then connect the outputs of the Reset Sequencer to components in the system.

Figure 235. Elements and Flow of a Reset Sequencer



- Reset Controller—Reused reset controller block. It synchronizes the reset inputs into one and feeds into the main FSM of the sequencer block.
- Sync—Synchronization block (double flipflop).
- Deglitch—Deglitch block. This block waits for a signal to be at a level for X clocks before propagating the input to the output.
- CSR—This block contains the CSR Avalon interface and related CSR register and control block in the sequencer.
- Main FSM—Main sequencer. This block determines when assertion/deassertion and assertion hold timing occurs.
- [A/D]SRT SEQ—Generic sequencer block that sequences out assertion/deassertion of reset from 0:N. The block has multiple counters that saturate upon reaching count.
- RESET_OUT—Controls the end output via:
  - Set/clear from the ASRT_SEQ/DSRT_SEQ.
  - Masking/forcing from CSR controls.
  - Remap of numbering (parameterization).

5.6.4.1. Reset Sequencer Parameters

Table 89. Reset Sequencer Parameters

Parameter	Description
Number of reset outputs	Sets the number of output resets to be sequenced, which is the number of output reset signals defined in the component with a range of 2 to 10.
Number of reset inputs	Sets the number of input reset signals to be sequenced, which is the number of input reset signals defined in the component with a range of 1 to 10.
Minimum reset assertion time	Specifies the minimum assertion cycles between the assertion of the last sequenced reset, and the deassertion of the first sequenced reset. The range is 0 to 1023.
Enable Reset Sequencer CSR	Enables CSR functionality of the Reset Sequencer through an Avalon interface.
reset_out#	Lists the reset output signals. Set the parameters in the other columns for each reset signal in the table.
ASRT Seq#	Determines the order of reset assertion. Enter the values 1, 2, 3, etc. to specify the required non-overlapping assertion order. This value determines the ASRT_REMAP value in the component HDL.
ASRT Cycle#	Number of cycles to wait before assertion of the reset. The value set here corresponds to the ASRT_DELAY value in the component HDL. The range is 0 to 1023.
DSRT Seq#	Determines the reset order of reset deassertion. Enter the values 1, 2, 3, etc. to specify the required non-overlapping deassertion order. This value determines the DSRT_REMAP value in the component HDL.
DSRT Cycle#/Deglitch#	Number of cycles to wait before deasserting or deglitching the reset. If the **USE_DRST_QUAL** parameter is set to 0, specifies the number of cycles to wait before deasserting the reset. If **USE_DSRT_QUAL** is set to1, specifies the number of cycles to deglitch the input `reset_dsrt_qual` signal. This value determines either the DSRT_DELAY, or the DSRT_QUALCNT value in the component HDL, depending on the **USE_DSRT_QUAL** parameter setting. The range is 0 to 1023.
USE_DSRT_QUAL	If you set **USE_DSRT_QUAL** to 1, the deassertion sequence waits for an external input signal for sequence qualification instead of waiting for a fixed delay count. To use a fixed delay count for deassertion, set this parameter to 0.

5.6.4.2. Reset Sequencer Timing Diagrams

Figure 236. Basic Sequencing

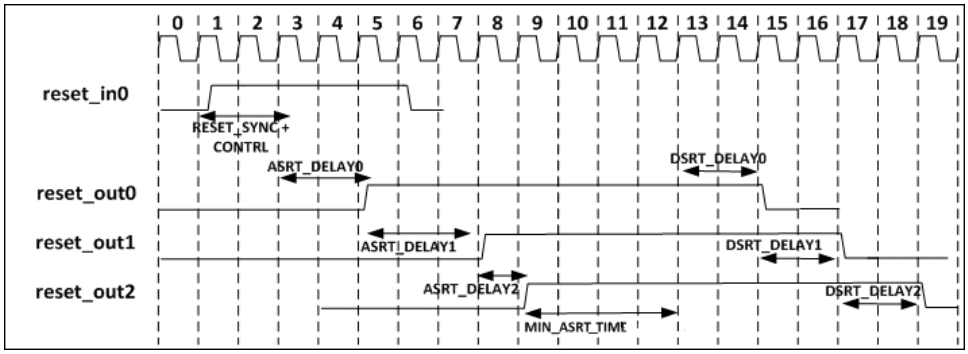
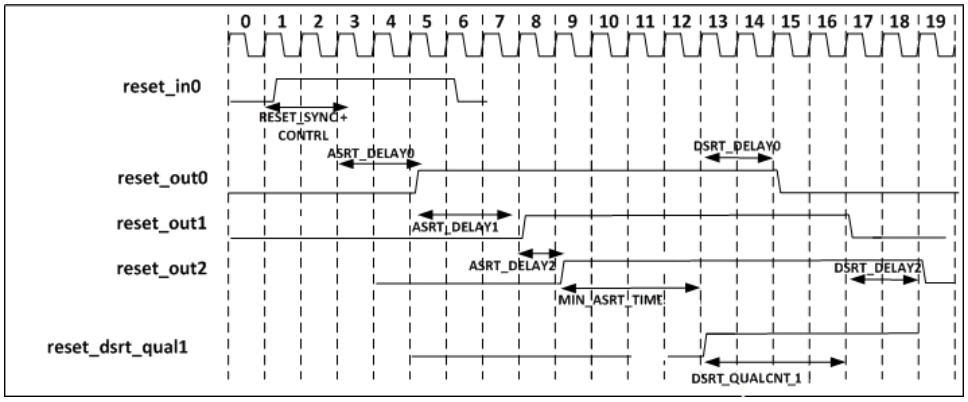


Figure 237. Sequencing with USE_DSRT_QUAL Set



5.6.4.3. Reset Sequencer CSR Registers

The Reset Sequencer's CSR registers provide the following functionality:

- **Support reset logging**
  - Ability to identify which reset is asserted.
  - Ability to determine whether any reset is currently active.
- **Support software triggered resets**
  - Ability to generate reset by writing to the register.
  - Ability to disable assertion or deassertion sequence.
- **Support software sequenced reset**
  - Ability for the software to fully control the assertion/deassertion sequence by writing to registers and stepping through the sequence.
- **Support reset override**
  - Ability to assert a specific component reset through software.

Table 90. Reset Sequencer CSR Register Map

Register	Offset	Width	Reset Value	Description
`Status Register`	`0x00`	32	`0x0`	The `Status` register indicates which sources are allowed to cause a reset.
`Interrupt Enable Register`	`0x04`	32	`0x0`	The `Interrupt Enable` register bits enable events triggering the IRQ of the reset sequencer.
`Control Register`	`0x08`	32	`0x0`	The `Control` register allows you to control the Reset Sequencer.
`Software Sequenced Reset Assert Control Register`	`0x0C`	32	`0x3FF`	You can program the `Software Sequenced Reset Assert` control register to control the reset assertion sequence.

continued...

Register	Offset	Width	Reset Value	Description
Software Sequenced Reset Deassert Control Register	0x10	32	0x3FF	You can program the Software Sequenced Reset Deassert register to control the reset deassertion sequence.
Software Direct Controlled Resets	0x14	32	0X0	You can write a bit to 1 to assert the reset_outN signal, and to 0 to deassert the reset_outN signal.
Software Reset Masking	0x18	32	0x0	Masking off (writing 1) to a reset_outN "Reset Mask Enable" signal prevents the corresponding reset from being asserted. Writing a bit to 0 to a reset mask enable signal allows assertion of reset_outN.

5.6.4.3.1. Reset Sequencer Status Register

The Status register indicates which sources are allowed to cause a reset.

You can clear bits by writing 1 to the bit location. The Reset Sequencer ignores attempts to write bits with a value of 0. If the sequencer is reset (power-on-reset), all bits are cleared, except the power-on-reset bit.

Table 91. Values for the Status Register at Offset 0x00

Bit	Attribute	Default	Description
31	RO	0	Reset Active—Indicates that the sequencer is currently active in reset sequence (assertion or deassertion).
30	RW1C	0	Reset Asserted and waiting for SW to proceed—Set when there is an active reset assertion, and the next sequence is waiting for the software to proceed. Only valid when the Enable SW sequenced reset assert option is turned on.
29	RW1C	0	Reset Deasserted and waiting for SW to proceed—Set when there is an active reset deassertion, and the next sequence is waiting for the software to proceed. Only valid when the Enable SW sequenced reset deassert option is turned on.
28:26	Reserved.		
25:16	RW1C	0	Reset deassertion input qualification signal reset_dsrt_qual [9:0] status—Indicates that the reset deassertion's input signal qualification signal is set. This bit is set on the detection of assertion of the signal.
15:12	Reserved.		
11	RW1C	0	reset_in9 was triggered—Indicates that reset_in9 triggered the reset. Software clears this bits by writing 1 to this location.
10	RW1C	0	reset_in8 was triggered—Indicates that reset_in8 triggered the reset. Software clears this bit by writing 1 to this location.
9	RW1C	0	reset_in7 was triggered—Indicates that reset_in7 triggered the reset. Software clears this bit by writing 1 to this location.
8	RW1C	0	reset_in6 was triggered—Indicates that reset_in6 triggered the reset. Software clears this bit by writing 1 to this location.

continued...

Bit	Attribute	Default	Description
7	RW1C	0	reset_in5 was triggered—Indicates that reset_in5 triggered the reset. Software clears this bit by writing 1 to this location.
6	RW1C	0	reset_in4 was triggered—Indicates that reset_in4 triggered the reset. Software clears this bit by writing 1 to this location.
5	RW1C	0	reset_in3 was triggered—Indicates that reset_in3 triggered the reset. Software clears this bit by writing 1 to this location.
4	RW1C	0	reset_in2 was triggered—Indicates that reset_in2 triggered the reset. Software clears this bit by writing 1 to this location.
3	RW1C	0	reset_in1 was triggered—Indicates that reset_in1 triggered the reset. Software clears this bit by writing 1 to this location.
2	RW1C	0	reset_in0 was triggered—Indicates that reset_in0 triggered. Software clears this bit by writing 1 to this location.
1	RW1C	0	Software-triggered reset—Indicates that the software-triggered reset is set by the software, and triggering a reset.
0	RW1C	0	Power-on-reset was triggered—Asserted whenever the reset to the sequencer is triggered. This bit is NOT reset when sequencer is reset. Software clears this bit by writing 1 to this location.

5.6.4.3.2. Reset Sequencer Interrupt Enable Register

The Interrupt Enable register bits enable events triggering the IRQ of the reset sequencer.

Table 92. Values for the Interrupt Enable Register at Offset 0x04

Bit	Attribute	Default	Description
31	Reserved.		
30	RW	0	Interrupt on Reset Asserted and waiting for SW to proceed enable. When set, the IRQ is set when the sequencer is waiting for the software to proceed in an assertion sequence.
29	RW	0	Interrupt on Reset Deasserted and waiting for SW to proceed enable. When set, the IRQ is set when the sequencer is waiting for the software to proceed in a deassertion sequence.
28:26	Reserved.		
25:16	RW	0	Interrupt on Reset deassertion input qualification signal reset_dsrt_qual_[9:0] status— When set, the IRQ is set when the reset_dsrt_qual[9:0] status bit (per bit enable) is set.
15:12	Reserved.		
11	RW	0	Interrupt on reset_in9 Enable—When set, the IRQ is set when the reset_in9 trigger status bit is set.
10	RW	0	Interrupt on reset_in8 Enable—When set, the IRQ is set when the reset_in8 trigger status bit is set.
9	RW	0	Interrupt on reset_in7 Enable—When set, the IRQ is set when the reset_in7 trigger status bit is set.
8	RW	0	Interrupt on reset_in6 Enable—When set, the IRQ is set when the reset_in6 trigger status bit is set.

continued...

Bit	Attribute	Default	Description
7	RW	0	Interrupt on reset_in5 Enable—When set, the IRQ is set when the reset_in5 trigger status bit is set.
6	RW	0	Interrupt on reset_in4 Enable—When set, the IRQ is set when the reset_in4 trigger status bit is set.
5	RW	0	Interrupt on reset_in3 Enable—When set, the IRQ is set when the reset_in3 trigger status bit is set.
4	RW	0	Interrupt on reset_in2 Enable—When set, the IRQ is set when the reset_in2 trigger status bit is set.
3	RW	0	Interrupt on reset_in1 Enable—When set, the IRQ is set when the reset_in1 trigger status bit is set.
2	RW	0	Interrupt on reset_in0 Enable—When set, the IRQ is set when the reset_in0 trigger status bit is set.
1	RW	0	Interrupt on Software triggered reset Enable—When set, the IRQ is set when the software triggered reset status bit is set.
0	RW	0	Interrupt on Power-On-Reset Enable—When set, the IRQ is set when the power-on-reset status bit is set.

5.6.4.3.3. Reset Sequencer Control Register

The Control register allows you to control the Reset Sequencer.

Table 93. Values for the Control Register at Offset 0x08

Bit	Attribute	Default	Description
31:3	Reserved.		
2	RW	0	Enable SW sequenced reset assert—Enable a software sequenced reset assert sequence. Timer delays and input qualification are ignored, and only the software can sequence the assert.
1	RW	0	Enable SW sequenced reset deassert—Enable a software sequenced reset deassert sequence. Timer delays and input qualification are ignored, and only the software can sequence the deassert.
0	WO	0	Initiate Reset Sequence—To trigger the hardware sequenced warm reset, the Reset Sequencer writes this bit to 1 a single time. The Reset Sequencer verifies that Reset Active is 0 before setting this bit, and always reads the value 0. To monitor this sequence, verify that Reset Active is asserted, and then subsequently deasserted.

5.6.4.3.4. Reset Sequencer Software Sequenced Reset Assert Control Register

You can program the Software Sequenced Reset Assert control register to control the reset assertion sequence.

When the corresponding enable bit is set, the sequencer stops when the desired reset asserts, and then sets the Reset Asserted and waiting for SW to proceed bit. The Reset Sequencer proceeds only after the Reset Asserted and waiting for SW to proceed bit is cleared.

Table 94. Values for the Reset Sequencer Software Sequenced Reset Assert Control Register at Offset 0x0C

Bit	Attribute	Default	Description
31:10	Reserved.		
9:0	RW	0x3FF	Per-reset SW sequenced reset assert enable—This is a per-bit enable for SW sequenced reset assert. If the register's bitN is set, the sequencer sets the bit30 of the status register when a resetN is asserted. It then waits for the bit30 of the status register to clear before proceeding with the sequence. By default, all bits are enabled (fully SW sequenced).

5.6.4.3.5. Reset Sequencer Software Sequenced Reset Deassert Control Register

You can program the Software Sequenced Reset Deassert register to control the reset deassertion sequence.

When the corresponding enable bit is set, the sequencer stops when the desired reset asserts, and then sets the Reset Deasserted and waiting for SW to proceed bit. The Reset Sequencer proceeds only after the Reset Deasserted and waiting for SW to proceed bit is cleared.

Table 95. Values for the Reset Sequencer Software Sequenced Reset Deassert Control Register at Offset 0x10

Bit	Attribute	Default	Description
31:10	Reserved.		
9:0	RW	0x3FF	Per-reset SW sequenced reset deassert enable—This is a per-bit enable for SW-sequenced reset deassert. If bitN of this register is set, the sequencer sets bit29 of the Status Register when a resetN is asserted. It then waits for the bit29 of the status register to clear before proceeding with the sequence. By default, all bits are enabled (fully SW sequenced).

5.6.4.3.6. Reset Sequencer Software Direct Controlled Resets

You can write a bit to 1 to assert the reset_outN signal, and to 0 to deassert the reset_outN signal.

Table 96. Values for the Software Direct Controlled Resets at Offset 0x14

Bit	Attribute	Default	Description
31:26	Reserved.		
25:16	WO	0	Reset Overwrite Trigger Enable—This is a per-bit control trigger bit for the overwrite value to take effect.
15:10	Reserved.		
9:0	WO	0	reset_outN Reset Overwrite Value—This is a per-bit control of the reset_out bit. The Reset Sequencer can use this to forcefully drive the reset to a specific value. A value of 1 sets the reset_out. A value of 0 clears the reset_out. A write to this register only takes effect if the corresponding trigger bit in this register is set.

5.6.4.3.7. Reset Sequencer Software Reset Masking

Masking off (writing 1) to a `reset_outN "Reset Mask Enable"` signal prevents the corresponding reset from being asserted. Writing a bit to `0` to a reset mask enable signal allows assertion of `reset_outN`.

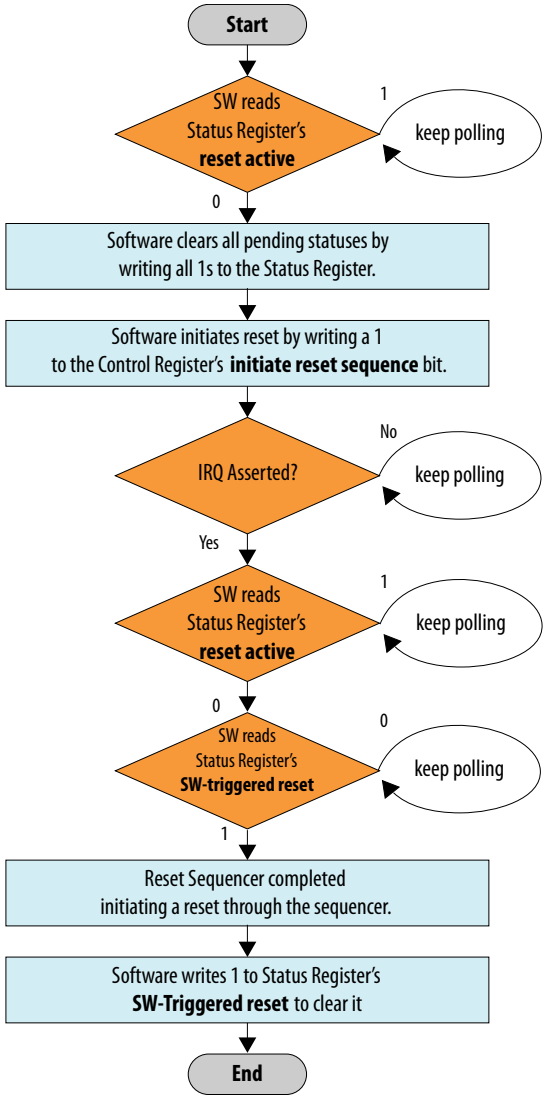
Table 97. Values for the Reset Sequencer Software Reset Masking at Offset 0x18

Bit	Attribute	Default	Description
31:10	Reserved.		
9:0	RW	0	`reset_outN "Reset Mask Enable"`—This is a per-bit control to mask off the `reset_outN` bit. Software Reset Masking prevents the reset bit from being asserted during a reset assertion sequence. If `reset_out` is already asserted, it does not deassert the reset.

5.6.4.4. Reset Sequencer Software Flows

5.6.4.4.1. Reset Sequencer (Software-Triggered) Flow

Figure 238. Reset Sequencer (Software-Triggered) Flow Diagram



Related Information

- Reset Sequencer Status Register on page 321
- Reset Sequencer Control Register on page 323

5.6.4.4.2. Reset Assert Flow

The following flow sequence occurs for a Reset Assert Flow:

- A reset is triggered either by the software, or when input resets to the Reset Sequencer are asserted.
- The IRQ is asserted, if the IRQ is enabled.
- Software reads the Status register to determine which reset was triggered.

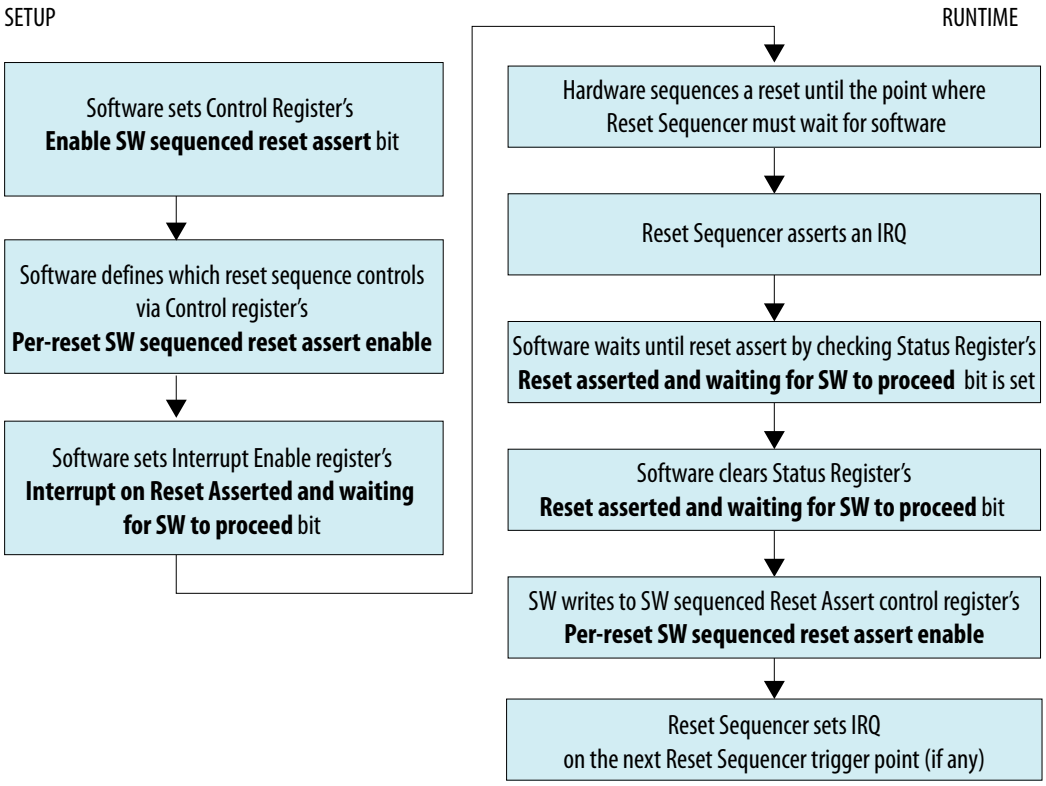
5.6.4.4.3. Reset Deassert Flow

The following flow sequence occurs for a Reset Deassert Flow:

- When a reset source is deasserted, or when the reset assert sequence has completed without pending resets asserted, the deassertion flow is initiated.
- The IRQ is asserted, if the IRQ is enabled.
- Software reads the Status Register to determine which reset was triggered.

5.6.4.4.4. Reset Assert (Software Sequenced) Flow

Figure 239. Reset Assert (Software Sequenced) Flow



Related Information

- Reset Sequencer Control Register on page 323
- Reset Sequencer Software Sequenced Reset Assert Control Register on page 323
- Reset Sequencer Interrupt Enable Register on page 322

- Reset Sequencer Status Register on page 321

5.6.4.4.5. Reset Deassert (Software Sequenced) Flow

The sequence and flow is similar to the `Reset Assert (SW Sequenced)` flow, though, this flow uses the `reset deassert` registers/bits instead of the `reset assert` registers/bits.

Related Information

Reset Assert (Software Sequenced) Flow on page 327

5.7. Conduits

You can use the conduit interface type for interfaces that do not fit any of the other interface types, and to group any arbitrary collection of signals. Like other interface types, you can export or connect conduit interfaces.

The PCI Express-to-Ethernet example in *Creating a System with Platform Designer* is an example of using a conduit interface for export. You can declare an associated clock interface for conduit interfaces in the same way as memory-mapped interfaces with the `associatedClock`.

To connect two conduit interfaces inside Platform Designer, the following conditions must be met:

- The interfaces must match exactly with the same signal roles and widths.
- The interfaces must be the opposite directions.
- Clocked conduit connections must have matching `associatedClocks` on each of their endpoint interfaces.

Note: To connect a conduit output to more than one input conduit interface, you can create a custom component. The custom component could have one input that connects to two outputs, and you can use this component between other conduits that you want to connect. For information about the Avalon Conduit interface, refer to the *Avalon Interface Specifications*

Related Information

- Avalon Interface Specifications
- Creating a System with Platform Designer on page 11

5.8. Interconnect Pipelining

You can use pipeline stages within the interconnect to increase a design's f_{MAX}. Insertion of pipeline stages reduces the combinational logic depth, while incurring additional latency and logic use.[13]

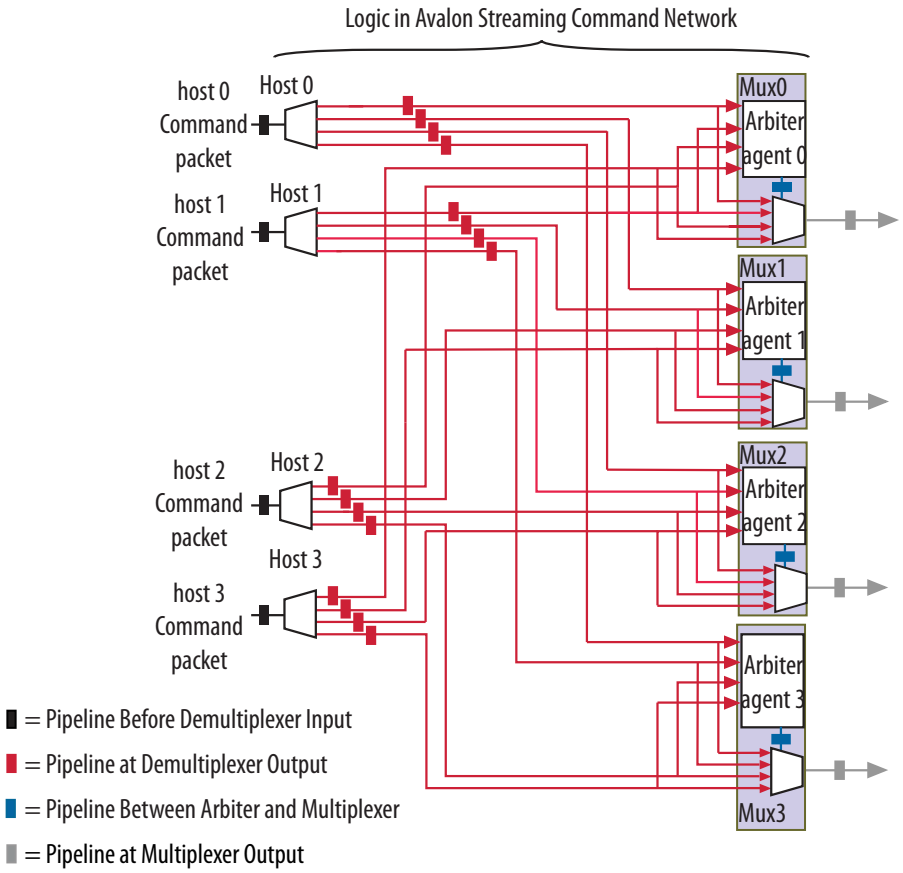
[13] Each pipeline stage requires two registers and some control logic to store write data, address, and control signals on the command path, as well as response, data, and control signals on the response path.

The **Limit interconnect pipeline stages to** option on the **Domains** tab allows you to specify the maximum number of fixed location Avalon pipeline stages that Platform Designer can automatically insert in the command and response path, as Figure 240 on page 329 illustrates. You can specify between 0 to 4 pipeline stages, where 0 means that the interconnect has a combinational datapath. Choosing 3 or 4 pipeline stages can significantly increase the logic utilization of the system. **Limit interconnect pipeline stages to** is specific to each Platform Designer system or subsystem.

Note: Enabling **Limit interconnect pipeline stages to** only *allows* insertion up to the limit you specify. However, the actual insertion of pipelines depends upon the existence of certain interconnect components. For example, single-agent systems do not have multiplexers; therefore, multiplexer pipelining does not occur. In an extreme case of a single-host to single-agent system, no pipelining occurs, regardless of the value of the **Limit interconnect pipeline stages to** option.

Figure 240. Pipeline Placement in Arbitration Logic for Command Path

The following example shows the location of up to four potential pipeline stages in the interconnect command path. Platform Designer can potentially place pipeline stages before the input to the demultiplexer, at the output of the demultiplexer, between the arbiter and the multiplexer, and at the output of the multiplexer. The response path has one less potential pipeline location than the command path.



If the **Limit interconnect pipeline stages to** setting does not provide enough fine control over the placement of pipelines, you can explicitly adjust the number of pipeline stages by clicking **Show System with Interconnect** on the **Domains** tab, as Add Pipeline Stages to the Interconnect Schematic on page 330 describes.

Related Information

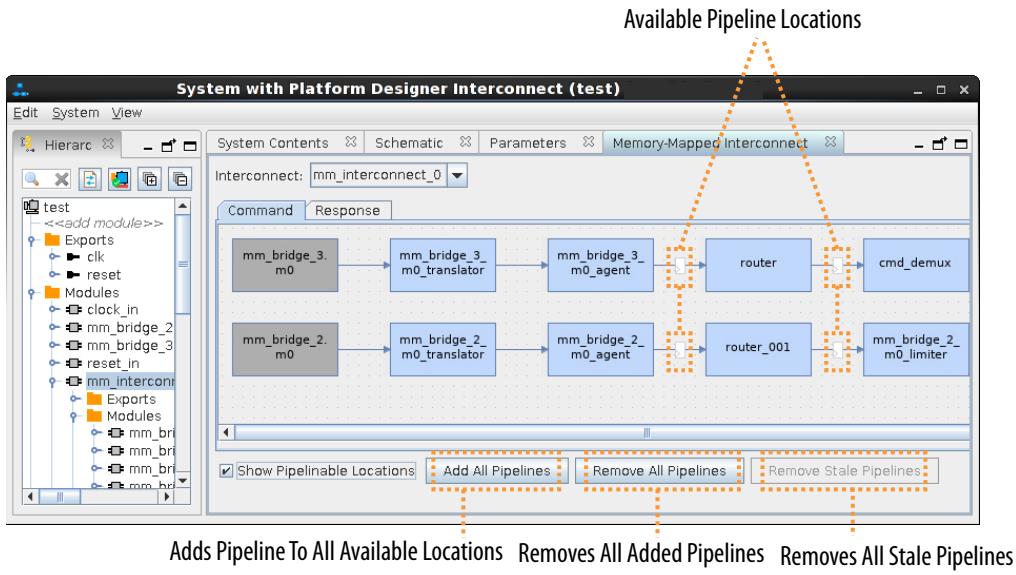
- Previewing the System Interconnect on page 74
- Inserting Pipeline Stages to Increase System Frequency on page 214

5.8.1. Add Pipeline Stages to the Interconnect Schematic

You can view and enable pipelinable locations in a graphical schematic of the Platform Designer interconnect. The **Memory-Mapped Interconnect** tab allows you to adjust pipeline connections in the Platform Designer command and response interconnect.

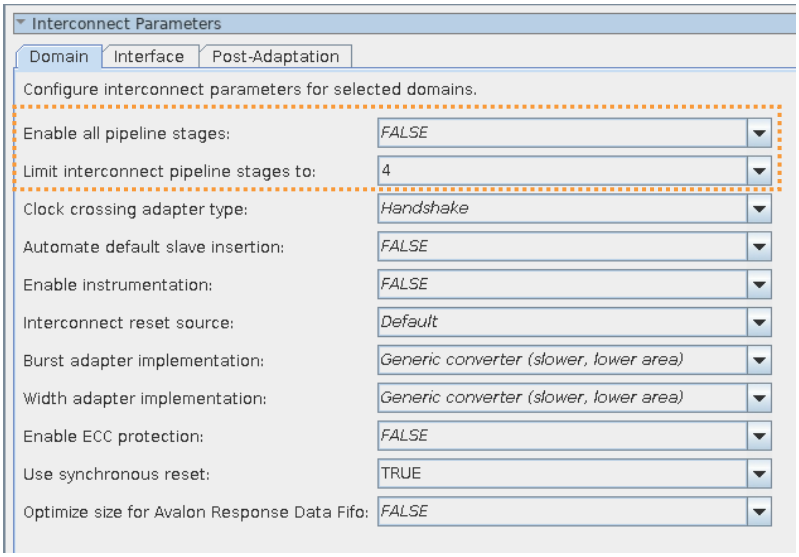
Note: You can specify the **Limit interconnect pipeline stages to** option to automatically insert from zero to a maximum of four pipeline stages, rather than adding pipeline stages explicitly in the schematic. Use the schematic if you require finer control than the option provides. Add pipelines to the interconnect schematic only for complete systems. When **Enable all pipeline stages** is set to **TRUE**, the **Limit interconnect pipeline stages to** option is disabled, and you cannot edit pipelines in the schematic.

Figure 241. Available Pipeline Locations in Memory Mapped Interconnect Schematic



1. In the Platform Designer software, open a system that includes two Avalon memory mapped pipeline bridge instances, with one instance functioning as the host, and the other instance functioning as the agent.
2. Click **View ➤ Domains**. The **Domains** tab displays the memory mapped domains in the system.
3. Under **Interconnect Parameters**, ensure that **Enable all pipeline stages** is set to **False**. When set to **True**, this option disables manual pipeline insertion.

Figure 242. Domains Tab



4. In the **Domains** tab, click **Show System With Interconnect**. The System with Platform Designer Interconnect window displays the available pipeline locations on the **Command** and **Response** tabs.
5. Specify the placement of pipeline stages in the interconnect schematic:
   - To view all locations that can accept a pipeline stage, enable **Show Pipelinable Locations**.
   - To add a pipeline to an available location, right-click the location and enable **Pipelined**.
   - To add pipelines at all available locations, click **Add All Pipelines**.
   - To remove all pipelines you add, click **Remove All Pipelines**.
   - To remove pipelines that have become stale due to changes since enabling a pipeline, click **Remove Stale Pipelines**. If you make changes to the original system's connectivity after manually pipelining an interconnect, the inserted pipelines may become invalid. Platform Designer displays warning messages when you generate the system if invalid pipeline stages are detected. You can remove invalid pipeline stages with the **Remove Stale Pipelines** option in the **Memory-Mapped Interconnect** tab. Do not make changes to the system's connectivity after manual pipeline insertion.

Note: Review manually-inserted pipelines when upgrading to newer versions of Platform Designer. Manually-inserted pipelines in one version of Platform Designer may not be valid in a future version.

5.9. Error Correction Coding (ECC) in Platform Designer Interconnect

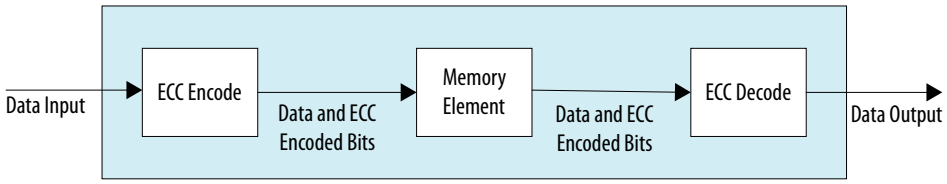
Error Correction Coding (ECC) logic allows the Platform Designer interconnect to detect and correct errors. Enabling ECC improves data integrity in memory blocks. Platform Designer supports ECC protection for Read Data FIFO (`rdata_FIFO`) instances only.

As transistors become smaller, computer hardware is more susceptible to data corruption. Data corruption causes Single Event Upsets (SEUs), and increases the probability of Failures in Time (FIT) rates in computer systems. SEU events without error notification can cause the system to be stuck in an unknown response status, and increase the FIT rate.

Before writing data to the memory device, the ECC logic encodes the data bus with a Hamming code. Then, the ECC logic decodes and performs error checking on the data output.

Platform Designer interconnect sends uncorrectable errors arising from memory as DECODEERROR (DECERR) on the Avalon response bus.

Figure 243. High-Level Implementation of rdata_FIFO with ECC Enabled



Note: Enabling ECC logic may increase logic utilization and cause lower f_{MAX}.

Related Information

Read and Write Responses on page 276

5.10. AMBA 3 AXI Protocol Specification Support (version 1.0)

Platform Designer allows memory-mapped connections between AMBA 3 AXI components, AMBA 3 AXI and AMBA 4 AXI components, and AMBA 3 AXI and Avalon interfaces with unique or exceptional support.

Refer to the *AMBA 3 Protocol Specifications* on the ARM website for more information. (14)

Related Information

- Arm AMBA Protocol Specifications
- Avalon Agent and AXI Subordinate Network Interfaces on page 261

5.10.1. Channels

Platform Designer has the following support and restrictions for AMBA 3 AXI channels.

(14) This document now refers to the AXI "manager" and "subordinate" to replace the former terms, as the latest AMBA® AXI and ACE Protocol Specification describes.

5.10.1.1. Read and Write Address Channels

Most signals are allowed. However, the following limitations are present in Platform Designer 14.0:

- Supports 64-bit addressing.
- ID width limited to 18-bits.
- HPS-FPGA host interface has a 12-bit ID.

5.10.1.2. Write Data, Write Response, and Read Data Channels

Most signals are allowed. However, the following limitations are present in Platform Designer 14.0:

- Data widths limited to a maximum of 1024-bits
- Limited to a fixed byte width of 8-bits

5.10.1.3. Low Power Channel

Low power extensions are not supported in Platform Designer, version 14.0.

5.10.2. Cache Support

AWCACHE and ARCACHE are passed to an AXI subordinate unmodified. (15)

5.10.2.1. Bufferable

Platform Designer interconnect treats AXI transactions as non-bufferable. All responses must come from the terminal subordinate.

When connecting to Avalon memory mapped agents, since they do not have write responses, the following exceptions apply:

- For Avalon memory mapped agents, the write response is generated by the AXI subordinate agent once the write transaction is accepted by the subordinate. The following limitation exists for an Avalon bridge:
- For an Avalon bridge, the response is generated before the write reaches the endpoint. Be aware of this limitation and avoid multiple paths past the bridge to any endpoint subordinate, or only perform bufferable transactions to an Avalon bridge.

5.10.2.2. Cacheable (Modifiable)

Platform Designer interconnect acknowledges the cacheable (modifiable) attribute of AXI transactions.

(15) This document now refers to the AXI "manager" and "subordinate" to replace the former outmoded terms. Refer to the *AMBA AXI and ACE Protocol Specification*.

It does not change the address, burst length, or burst size of non-modifiable transactions, with the following exceptions:

- Platform Designer considers a wide transaction to a narrow agent as modifiable because the size requires reduction.
- Platform Designer may consider AXI read and write transactions as modifiable when the destination is an Avalon agent. The AXI transaction may be split into multiple Avalon transactions if the agent is unable to accept the transaction. This may occur because of burst lengths, narrow sizes, or burst types.

Platform Designer ignores all other bits, for example, read allocate or write allocate because the interconnect does not perform caching. By default, Platform Designer considers Avalon host transactions as non-bufferable and non-cacheable, with the allocate bits tied low.

5.10.3. Security Support

TrustZone refers to the security extension of the ARM architecture, which includes the concept of "secure" and "non-secure" transactions, and a protocol for processing between the designations.

The interconnect passes the AWPROT and ARPROT signals to the endpoint subordinate without modification. It does not use or modify the PROT bits.

Refer to *Manage System Security* in *Creating a System with Platform Designer* for more information about secure systems and the TrustZone feature.

Related Information

Configuring Platform Designer System Security on page 79

5.10.4. Atomic Accesses

Exclusive accesses are supported for AXI subordinates by passing the lock, transaction ID, and response signals from manager to subordinate, with the limitation that subordinates that do not reorder responses. Avalon agents do not support exclusive accesses, and always return OKAY as a response. Locked accesses are also not supported.

5.10.5. Response Signaling

Full response signaling is supported. Avalon agents always return OKAY as a response.

5.10.6. Ordering Model

Platform Designer interconnect provides responses in the same order as the commands are issued.

To prevent reordering, for subordinates that accept reordering depths greater than 1, Platform Designer does not transfer the transaction ID from the manager, but provides a constant transaction ID of 0. For subordinates that do not reorder, Platform Designer allows the transaction ID to be transferred to the subordinate. To avoid cyclic dependencies, Platform Designer supports a single outstanding subordinate scheme for both reads and writes. Changing the targeted subordinate before all responses have returned stalls the manager, regardless of transaction ID.

5.10.6.1. AXI and Avalon Ordering

There is a potential read-after-write risk when Avalon hosts transact to AXI subordinates.

According to the *AMBA Protocol Specifications*, there is no ordering requirement between reads and writes. However, Avalon has an implicit ordering model that requires transactions from a host to the same AXI subordinate to be in order. The Avalon interconnect always processes the transactions in order. The interconnect blocks transactions if required. The interconnect prevents writing to the AXI subordinate when read is pending.

5.10.7. Data Buses

Narrow bus transfers are supported. AXI write strobes can have any pattern that is compatible with the address and size information. Intel recommends that transactions to Avalon agents follow Avalon `byteenable` limitations for maximum compatibility.

Note: Byte `0` is always bits `[7:0]` in the interconnect, following AXI's and Avalon's byte (address) invariance scheme.

5.10.8. Unaligned Address Commands

Unaligned address commands are commands with addresses that do not conform to the data width of a subordinate. Since Avalon memory mapped subordinates accept only aligned addresses, Platform Designer modifies unaligned commands from AXI managers to the correct data width. Platform Designer must preserve commands issued by AXI managers when passing the commands to AXI subordinates.

Note: Unaligned transfers are aligned if downsizing occurs. For example, when downsizing to a bus width narrower than that required by the transaction size, `AWSIZE` or `ARSIZE`, the transaction must be modified.

5.10.9. Avalon and AXI Transaction Support

Platform Designer supports transactions between Avalon and AXI interfaces with the following limitations in this section.

Related Information

Avalon Interface Specifications

5.10.9.1. Transaction Cannot Cross 4KB Boundaries

When an Avalon host issues a transaction to an AXI subordinate, the transaction cannot cross 4KB boundaries. Non-bursting Avalon hosts already follow this boundary restriction.

When connecting an Avalon memory-mapped interface FPGA host to an AXI subordinate in Platform Designer, you must ensure that the bursts do not exceed the AXI3 or AXI4 4KB boundary restriction for burst transactions.

5.10.9.2. Adjacent Bytelanes with Partial Width Transactions

The following limitations apply to Avalon to AXI partial width transactions with use of adjacent bytelanes:

- Avalon interfaces only support adjacent bytelanes if the interface requires more than one byte enable. For example: `1100`, `0011`.
- AXI fully supports use of bytelanes that are not adjacent. For example: `1010`, `0101`.

Related Information

Avalon Interface Specifications

5.10.9.3. Handling Read Side Effects

Read side effects can occur when more bytes than necessary are read from the subordinate, and the unwanted data that are read are later inaccessible on subsequent reads. For write commands, the correct byteenable paths are asserted based on the size of the transactions. For read commands, narrow-sized bursts are broken up into multiple non-bursting commands, and each command with the correct byteenable paths asserted.

Platform Designer always assumes that the byteenable is asserted based on the size of the command, not the address of the command. The following scenarios are examples:

- For a 32-bit AXI manager that issues a read command with an unaligned address starting at address `0x01`, and a burstcount of 2 to a 32-bit Avalon agent, the starting address is: `0x00`.
- For a 32-bit AXI manager that issues a read command with an unaligned address starting at address `0x01`, with 4-bytes to an 8-bit AXI subordinate, the starting address is: `0x00`.

5.11. AMBA 3 APB Protocol Specification Support (version 1.0)

APB (Advanced Peripheral Bus) interface is optimized for minimal power consumption and reduced interface complexity. You can use APB to interface to peripherals which are low-bandwidth and do not require the high performance of a pipelined bus interface. Signal transitions are sampled at the rising edge of the clock to enable the integration of APB peripherals easily into any design flow.

Platform Designer allows connections between APB components, and AMBA 3 AXI, AMBA 4 AXI, and Avalon memory-mapped interfaces. The following sections describe unique or exceptional APB support in the Platform Designer software. (16)

Related Information

Arm AMBA Protocol Specifications

5.11.1. Bridges

With APB, you cannot use bridge components that use multiple PSELx in Platform Designer. As a workaround, you can group PSELx, and then send the packet to the subordinate directly.

(16) This document now refers to the AXI "manager" and "subordinate" to replace the former terms, as the latest AMBA® AXI and ACE Protocol Specification describes.

Intel recommends as an alternative that you instantiate the APB bridge and all the APB subordinates in Platform Designer. You should then connect the subordinate side of the bridge to any high speed interface and connect the manager side of the bridge to the APB subordinates. Platform Designer creates the interconnect on either side of the APB bridge and creates only one PSEL signal.

Alternatively, you can connect a bridge to the APB bus outside of Platform Designer. Use an AXI bridge to export the AXI manager to the top-level, and then connect this AXI interface to the subordinate side of the APB bridge. Alternatively, instantiate the APB bridge in Platform Designer and export APB manager to the top- level, and from there connect to APB bus outside of Platform Designer.

5.11.2. Burst Adaptation

APB is a non-bursting interface. Therefore, for any AXI manager or Avalon host with bursting support, a burst adapter is inserted before the agent or subordinate interface, and the burst transaction is translated into a series of non-bursting transactions before reaching the APB subordinate or agent.

5.11.3. Width Adaptation

Platform Designer allows different data width connections with APB. When connecting a wider manager to a narrower APB subordinate, the width adapter converts the wider transactions to a narrower transaction to fit the APB subordinate data width. APB does not support Write Strobe. Therefore, when you connect a narrower transaction to a wider APB subordinate, the subordinate cannot determine which byte lane to write. In this case, the subordinate data may be overwritten or corrupted.

5.11.4. Error Response

Error responses are returned to the manager. Platform Designer performs error mapping if the manager is an AMBA 3 AXI or AMBA 4 AXI manager, for example, RRESP/BRESP= SLVERR. For the case when the subordinate does not use SLVERR signal, an OKAY response is sent back to manager by default.

5.12. AMBA 4 AXI Memory-Mapped Interface Support (version 2.0)

Platform Designer allows memory-mapped connections between AMBA 4 AXI components, AMBA 4 AXI and AMBA 3 AXI components, and AMBA 4 AXI and Avalon interfaces with unique or exceptional support. [17]

5.12.1. Burst Support

Platform Designer supports `INCR` bursts up to 256 beats. Platform Designer converts long bursts to multiple bursts in a packet with each burst having a length less than or equal to `MAX_BURST` when going to AMBA 3 AXI or Avalon agents.

For narrow-sized transfers, bursts with Avalon agents as destinations are shortened to multiple non-bursting transactions in order to transmit the correct address to the agents, since Avalon agents always perform full-sized `datawidth` transactions.

[17] This document now refers to the AXI "manager" and "subordinate" to replace the former terms, as the latest AMBA® AXI and ACE Protocol Specification describes.

Bursts with AMBA 3 AXI subordinates as destinations are shortened to multiple bursts, with each burst length less than or equal to 16. Bursts with AMBA 4 AXI subordinates as destinations are not shortened.

5.12.2. QoS

Platform Designer routes 4-bit QoS signals (Quality of Service Signaling) on the read and write address channels directly from the AXI manager or Avalon host to the AXI subordinate or Avalon agent, respectively.

Transactions from AXI managers and Avalon hosts have a default value of `4'b0000`, which indicates that the transactions are not part of the QoS flow. QoS values are not used for subordinates or agents that do not support QoS.

There are no programmable QoS registers or compile-time QoS options for a manager or host that overrides its real or default value.

5.12.3. Regions

AMBA 4 AXI subordinates with `AXREGION` signals are allowed. `AXREGION` signals are driven with the default value of `0x0`, and are limited to one entry in a manager's address map.

5.12.4. Write Response Dependency

Write response dependency as specified in the *Arm AMBA Protocol Specifications* for AMBA 4 AXI is not supported.

Related Information

Arm AMBA Protocol Specifications

5.12.5. AWCACHE and ARCACHE

For AMBA 4 AXI, Platform Designer meets the requirement for modifiable and non-modifiable transactions. The modifiable bit refers to `ARCACHE[1]`and `AWCACHE[1]`.

5.12.6. Width Adaptation and Data Packing in Platform Designer

Data packing applies only to systems where the data width of Avalon hosts or AXI managers is less than the data width of Avalon agents or AXI subordinates, respectively.

The following rules apply:

- Data packing is supported when hosts and agents are Avalon memory mapped.
- Data packing is not supported when any manager or subordinate is an AMBA 3 AXI, AMBA 4 AXI, or APB component.

For example, for a read/write command with a 32-bit host connected to a 64-bit agent, and a transaction of 2 burstcounts, Platform Designer sends 2 separate read/write commands to access the 64-bit data width of the agent.

5.12.7. Ordering Model

Platform Designer does not support out of order command response. Platform Designer processes AXI subordinates as device non-bufferable memory types.

The following describes the required behavior for the device non-bufferable memory type:

- Write response must be obtained from the final destination.
- Read data must be obtained from the final destination
- Transaction characteristics must not be modified.
- Reads must not be pre-fetched. Writes must not be merged.
- Non-modifiable read and write transactions.

(`AWCACHE[1] = 0 or ARCACHE[1] = 0`) from the same ID to the same subordinate must remain ordered. The interconnect always provides responses in the same order as the commands issued. subordinates that support reordering provide a constant transaction ID to prevent reordering. AXI subordinates that do not reorder are provided with transaction IDs, which allows exclusive accesses for such subordinates.

5.12.8. Read and Write Allocate

Read and write allocate does not apply to Platform Designer interconnect, which does not have caching features, and always receives responses from an endpoint.

5.12.9. Locked Transactions

Locked transactions are not supported for Platform Designer, version 14.0.

5.12.10. Memory Types

For AMBA 4 AXI, Platform Designer processes transactions as though the endpoint is a device memory type. For device memory types, using non-bufferable transactions to force previous bufferable transactions to finish is irrelevant, because Platform Designer interconnect always identifies transactions as being non-bufferable.

5.12.11. Mismatched Attributes

There are rules for how multiple managers issue cache values to a shared memory region. The interconnect meets requirements if signals are not modified.

5.12.12. Signals

Platform Designer supports up to 64-bits for the BUSER, WUSER and RUSER sideband signals. AMBA 4 AXI allows some signals to be omitted from interfaces by aligning them with the default values as defined in the *AMBA Protocol Specifications* on the ARM website.

Related Information

Arm AMBA Protocol Specifications

5.13. AMBA 4 AXI Streaming Interface Support (version 1.0)

5.13.1. Connection Points

Platform Designer allows you to connect an AMBA 4 AXI-Stream interface to another AMBA 4 AXI-Stream interface.

The connection is point-to-point without adaptation and must be between an `axi4stream_manager` and `axi4stream_subordinate`. Connected interfaces must have the same port roles and widths.

Non matching manager to subordinate connections, and multiple managers to multiple subordinates connections are not supported. (18)

5.13.1.1. AMBA 4 AXI Streaming Connection Point Parameters

Table 98. AMBA 4 AXI Streaming Connection Point Parameters

Name	Type	Description
`associatedClock`	string	Name of associated clock interface.
`associatedReset`	string	Name of associated reset interface

5.13.1.2. AMBA 4 AXI Streaming Connection Point Signals

Table 99. AMBA 4 AXI-Stream Connection Point Signals

Port Role	Width	Manager Direction	Subordinate Direction	Required
`tvalid`	1	Output	Input	Yes
`tready`	1	Input	Output	No
`tdata`(19)	8:4096	Output	Input	No
`tstrb`	1:512	Output	Input	No
`tkeep`	1:512	Output	Input	No
`tid`(20)	1:8	Output	Input	No

continued...

(18) This document refers to the new AXI "manager" and AXI "subordinate" inclusive terms to replace outmoded terms, as the latest version of the AMBA® AXI and ACE Protocol Specification describes.

(19) integer in multiple of bytes

(20) maximum 8-bits

Port Role	Width	Manager Direction	Subordinate Direction	Required
tdest[21]	1:4	Output	Input	No
tuser[22]	1:4096	Output	Input	No
tlast	1	Output	Input	No

5.13.2. Adaptation

AMBA 4 AXI-Stream adaptation support is not available. AMBA 4 AXI-Stream manager and subordinate interface signals and widths must match.

5.14. AMBA 4 AXI-Lite Protocol Specification Support (version 2.0)

AMBA 4 AXI-Lite is a sub-set of AMBA 4 AXI. It is suitable for simpler control register-style interfaces that do not require the full functionality of AMBA 4 AXI. [23]

Platform Designer supports the following AMBA 4 AXI-Lite features:

- Transactions with a burst length of 1.
- Data accesses use the full width of a data bus (32- bit or 64-bit) for data accesses, and no narrow-size transactions.
- Non-modifiable and non-bufferable accesses.
- No exclusive accesses.

5.14.1. AMBA 4 AXI-Lite Signals

Platform Designer supports all AMBA 4 AXI-Lite interface signals. Signals that are optional in AMBA 4 are also optional in AMBA 4 AXI-Lite.

Table 100. AMBA 4 AXI-Lite Signals

Global	Write Address Channel	Write Data Channel	Write Response Channel	Read Address Channel	Read Data Channel
ACLK	AWVALID	WVALID	BVALID	ARVALID	RVALID
ARESETn	AWREADY	WREADY	BREADY	ARREADY	RREADY
-	AWADDR	WDATA	BRESP	ARADDR	RDATA
-	AWPROT	WSTRB	-	ARPROT	RRESP

(21) maximum 4-bits

(22) number of bits in multiple of the number of bytes of tdata

(23) This document refers to the new AXI "manager" and AXI "subordinate" inclusive terms to replace outmoded terms, as the latest version of the AMBA® AXI and ACE Protocol Specification describes.

5.14.2. AMBA 4 AXI-Lite Optional Port Support and Interconnect

Platform Designer permits you to omit optional ports on the AMBA 4 AXI-Lite interface. However, the Platform Designer interconnect does not support the optional AMBA 4 AXI-Lite signals. The interconnect-facing AMBA 4 AXI-Lite interface must include all signals in AMBA 4 AXI-Lite Signals.

5.14.3. AMBA 4 AXI-Lite Bus Width

AMBA 4 AXI-Lite managers or subordinates must have either 32-bit or 64-bit bus widths. Platform Designer interconnect inserts a width adapter if a manager and subordinate pair have different widths.

5.14.4. AMBA 4 AXI-Lite Outstanding Transactions

AXI-Lite supports outstanding transactions. The options to control outstanding transactions is set in the parameter editor for the selected component.

5.14.5. AMBA 4 AXI-Lite IDs

AMBA 4 AXI-Lite does not support IDs. Platform Designer performs ID reflection inside the AXI subordinate agent.

5.14.6. Connections Between AMBA 3 AXI,AMBA 4 AXI and AMBA 4 AXI-Lite

5.14.6.1. AMBA 4 AXI-Lite Subordinate Requirements

For an AMBA 4 AXI-Lite subordinate side, the host or manager can be any interface type, such as an Avalon (with bursting), AMBA 3 AXI, or AMBA 4 AXI. Platform Designer allows the following connections and inserts adapters, if needed.

- **Burst adapter**—Avalon and AMBA 3 AXI and AMBA 4 AXI bursting host or manager require a burst adapter to shorten the burst length to 1 before sending a transaction to an AMBA 4 AXI-Lite subordinate.
- Platform Designer interconnect uses a width adapter for mismatched data widths.
- Platform Designer interconnect performs ID reflection inside the subordinate agent.
- An AMBA 4 AXI-Lite subordinate must have an address width of at least 12-bits.
- AMBA 4 AXI-Lite does not have the AXSIZE parameter. Narrow manager to a wide AMBA 4 AXI-Lite subordinate is not supported. For managers that support narrow-sized bursts, for example, AMBA 3 AXI and AMBA 4 AXI, a burst to an AMBA 4 AXI-Lite subordinate must have a burst size equal to or greater than the subordinate's burst size.

5.14.6.2. AMBA 4 AXI-Lite Data Packing

Platform Designer interconnect does not support AMBA 4 AXI-Lite data packing.

5.14.7. AMBA 4 AXI-Lite Response Merging

When Platform Designer interconnect merges SLVERR and DECERR, the error responses are not sticky. The response is based on priority and the manager always sees a DECERR. When SLVERR and DECERR are merged, it is based on their priorities, not stickiness. DECERR receives priority in this case, even if SLVERR returns first.

5.15. Port Roles (Interface Signal Types)

Each interface defines signal roles and their behavior. Many signal roles are optional, allowing IP component designers the flexibility to select only the signal roles necessary to implement the required functionality.

5.15.1. AXI Manager Interface Signal Types

Table 101. AXI Manager Interface Signal Types

Name	Direction	Width
araddr	output	1 - 64
arburst	output	2
arcache	output	4
arid	output	1 - 18
arlen	output	4
arlock	output	2
arprot	output	3
arready	input	1
arsize	output	3
aruser	output	1 - 64
arvalid	output	1
awaddr	output	1 - 64
awburst	output	2
awcache	output	4
awid	output	1 - 18
awlen	output	4
awlock	output	2
awprot	output	3
awready	input	1
awsize	output	3
awuser	output	1 - 64
awvalid	output	1
bid	input	1 - 18

continued...

Name	Direction	Width
bready	output	1
bresp	input	2
bvalid	input	1
rdata	input	8, 16, 32, 64, 128, 256, 512, 1024
rid	input	1 - 18
rlast	input	1
rready	output	1
rresp	input	2
rvalid	input	1
wdata	output	8, 16, 32, 64, 128, 256, 512, 1024
wid	output	1 - 18
wlast	output	1
wready	input	1
wstrb	output	1, 2, 4, 8, 16, 32, 64, 128
wvalid	output	1

5.15.2. AXI Subordinate Interface Signal Types

Table 102. AXI Subordinate Interface Signal Types

Name	Direction	Width
araddr	input	1 - 64
arburst	input	2
arcache	input	4
arid	input	1 - 18
arlen	input	4
arlock	input	2
arprot	input	3
arready	output	1
arsize	input	3
aruser	input	1 - 64
arvalid	input	1
awaddr	input	1 - 64
awburst	input	2
awcache	input	4
awid	input	1 - 18

continued...

Name	Direction	Width
awlen	input	4
awlock	input	2
awprot	input	3
awready	output	1
awsize	input	3
awuser	input	1 - 64
awvalid	input	1
bid	output	1 - 18
bready	input	1
bresp	output	2
bvalid	output	1
rdata	output	8, 16, 32, 64, 128, 256, 512, 1024
rid	output	1 - 18
rlast	output	1
rready	input	1
rresp	output	2
rvalid	output	1
wdata	input	8, 16, 32, 64, 128, 256, 512, 1024
wid	input	1 - 18
wlast	input	1
wready	output	1
wstrb	input	1, 2, 4, 8, 16, 32, 64, 128
wvalid	input	1

5.15.3. AMBA 4 AXI Manager Interface Signal Types

Table 103. AMBA 4 AXI manager Interface Signal Types

Name	Direction	Width
araddr	output	1 - 64
arburst	output	2
arcache	output	4
arid	output	1 - 18
arlen	output	8
arlock	output	1
arprot	output	3

continued...

Name	Direction	Width
arready	input	1
arregion	output	1 - 4
arsize	output	3
aruser	output	1 - 64
arvalid	output	1
awaddr	output	1 - 64
awburst	output	2
awcache	output	4
awid	output	1 - 18
awlen	output	8
awlock	output	1
awprot	output	3
awqos	output	1 - 4
awready	input	1
awregion	output	1 - 4
awsize	output	3
awuser	output	1 - 64
awvalid	output	1
bid	input	1 - 18
bready	output	1
bresp	input	2
buser	input	1 - 64
bvalid	input	1
rdata	input	8, 16, 32, 64, 128, 256, 512, 1024
rid	input	1 - 18
rlast	input	1
rready	output	1
rresp	input	2
ruser	input	1 - 64
rvalid	input	1
wdata	output	8, 16, 32, 64, 128, 256, 512, 1024
wid	output	1 - 18
wlast	output	1
wready	input	1

continued...

Name	Direction	Width
wstrb	output	1, 2, 4, 8, 16, 32, 64, 128
wuser	output	1 - 64
wvalid	output	1

5.15.4. AMBA 4 AXI Subordinate Interface Signal Types

Table 104. AMBA 4 AXI Subordinate Interface Signal Types

Name	Direction	Width
araddr	input	1 - 64
arburst	input	2
arcache	input	4
arid	input	1 - 18
arlen	input	8
arlock	input	1
arprot	input	3
arqos	input	1 - 4
arready	output	1
arregion	input	1 - 4
arsize	input	3
aruser	input	1 - 64
arvalid	input	1
awaddr	input	1 - 64
awburst	input	2
awcache	input	4
awid	input	1 - 18
awlen	input	8
awlock	input	1
awprot	input	3
awqos	input	1 - 4
awready	output	1
awregion	input	1 - 4
awsize	input	3
awuser	input	1 - 64
awvalid	input	1
bid	output	1 - 18

continued...

Name	Direction	Width
bready	input	1
bresp	output	2
bvalid	output	1
rdata	output	8, 16, 32, 64, 128, 256, 512, 1024
rid	output	1 - 18
rlast	output	1
rready	input	1
rresp	output	2
ruser	output	1 - 64
rvalid	output	1
wdata	input	8, 16, 32, 64, 128, 256, 512, 1024
wlast	input	1
wready	output	1
wstrb	input	1, 2, 4, 8, 16, 32, 64, 128
wuser	input	1 - 64
wvalid	input	1

5.15.5. AMBA 4 AXI-Stream Manager and Subordinate Interface Signal Types

Table 105. AMBA 4 AXI-Stream Manager and Subordinate Interface Signal Types

Name	Width	Manager Direction	Subordinate Direction	Required
tvalid	1	Output	Input	Yes
tready	1	Input	Output	No
tdata	8:4096	Output	Input	No
tstrb	1:512	Output	Input	No
tkeep	1:512	Output	Input	No
tid	1:8	Output	Input	No
tdest	1:4	Output	Input	No
tuser	1	Output	Input	No
tlast	1:4096	Output	Input	No

5.15.6. AMBA 4 AXI-Lite Signal Support and Limitations

ACE-Lite is a sub-set of AMBA 4 AXI that consists of an AMBA 4 AXI interface with additional signals on the read address and write address channels. ACE-Lite signals indicate transactions meant for cache coherence, cache maintenance, and other functions.

Table 106. ACE-Lite Interface Signal Roles

Name	Width	Manager Direction	Subordinate Direction	Required
arsnoop	4 bits	Output	Input	Yes
ardomain	2 bits	Output	Input	Yes
arbar	2 bits	Output	Input	Yes
awsnoop	3 bits	Output	Input	Yes
awdomain	2 bits	Output	Input	Yes
awbar	2 bits	Output	Input	Yes
awunique	1 bit	Output	Input	Yes

Note: Platform Designer's ACE-Lite interface does comprise of all the signals specified in the AMBA 4 AXI specification. The following sections describe the Platform Designer support and limitations for ACE-Lite transactions.

5.15.6.1. ACE-Lite Transaction Support and Limitations

Platform Designer's ACE-Lite interface does currently comprise of all the signals specified in the AMBA 4 AXI specification. Therefore, there are limitations on the extent of transactional support that Platform Designer interconnect provides for ACE-Lite transactions. The signals are present for point-to-point (1-manager, 1-subordinate) and non-point-to-point Interconnects, when the manager and subordinate ACE-Lite interfaces are not identical and require adaptation.

Table 107. Supported ACE-Lite Transactions

Transaction Type	Supported Transactions
Non-snoop transactions	WriteNoSnoop, ReadNoSnoop
Coherent transactions	WriteUnique, ReadOnce

Table 108. Unsupported ACE-Lite Transactions

Transaction Type	Unsupported Transactions
Barrier transactions	ReadBarrier, WriteBarrier
Cache Maintenance transactions	CleanShared, CleanInvalid, MakeInvalid
Coherent transactions	WriteLineUnique

Note: Unsupported ACE-Lite transaction interconnect behavior is non-deterministic. Use of unsupported transactions can cause a system failure or data corruption.

The following describes the ACE-Lite signal connectivity in various manager-subordinate combinations for the Platform Designer interconnect:

Table 109. ACE-Lite Signal Connectivity in Various Manager-Subordinate Combinations

Manager/Subordinate Combination	ACE-Lite Signal Connectivity
ACE-Lite manager to Avalon memory mapped agent.	ACE-Lite interface signals drop from the interconnect.
AXI manager to ACE-Lite subordinate	The ACE-Lite subordinate receives default values for ACE-Lite interface signals from the manager agent, by virtue of ACE-Lite interface being unsupported on the manager. The values are as follows: • `ARDOMAIN/AWDOMAIN = 2'b11` • `ARSNOOP/AWSNOOP = 0` • `ARBAR/AWBAR = 0` The above transactions indicate non-snooping `ReadNoSnoop/WriteNoSnoop` transactions. `AWUNIQUE = 0` indicates that `AWUNIQUE` is unsupported.
ACE-Lite manager to AXI subordinate	The subordinate agent sends out default values for ACE-Lite interface signals because the subordinate does not support the interface. The values are as follows: • `ARDOMAIN/AWDOMAIN = 2'b11` • `ARSNOOP/AWSNOOP = 0` • `ARBAR/AWBAR = 0` The above transactions indicate non-snooping `ReadNoSnoop/WriteNoSnoop` transactions. `AWUNIQUE = 0` indicates that `AWUNIQUE` is unsupported.
ACE-Lite manager to ACE-Lite subordinate	The ACE-Lite interface signals are sent through the interconnect as pass through.

5.15.7. APB Interface Signal Types

Table 110. APB Interface Signal Types

Name	Width	Direction APB Manager	Direction APB Subordinate	Required
`paddr`	[1:32]	output	input	yes
`psel`	[1:16]	output	input	yes
`penable`	1	output	input	yes
`pwrite`	1	output	input	yes
`pwdata`	[1:32]	output	input	yes
`prdata`	[1:32]	input	output	yes
`pslverr`	1	input	output	no
`pready`	1	input	output	yes
`paddr31`	1	output	input	no

5.15.8. Avalon Memory Mapped Interface Signal Roles

Signal roles define the signal types that Avalon memory mapped host and agent ports allow.

This specification does not require all signals to exist in an Avalon memory mapped interface. There is no one signal that is always required. The minimum requirements for an Avalon memory mapped interface are `readdata` for a read-only interface, or `writedata` and `write` for a write-only interface.

The following table lists signal roles for the Avalon memory mapped interface:

Table 111. Avalon Memory Mapped Signal Roles

Some Avalon memory mapped signals can be active high or active low. When active low, the signal name ends with _n.

Signal Role	Width	Direction	Required	Description
Fundamental Signals				
`address`	1 - 64	Host → Agent	No	Hosts: By default, the `address` signal represents a byte address. The value of the address must align to the data width. To write to specific bytes within a data word, the host must use the `byteenable` signal. Refer to the `addressUnits` interface property for word addressing. Agents: By default, the interconnect translates the byte address into a word address in the agent's address space. From the perspective of the agent, each agent access is for a word of data. For example, `address = 0` selects the first word of the agent. `address = 1` selects the second word of the agent. Refer to the `addressUnits` interface property for byte addressing.
`byteenable` `byteenable_n`	2, 4, 8, 16, 32, 64, 128	Host → Agent	No	Enables one or more specific byte lanes during transfers on interfaces of width greater than 8 bits. Each bit in `byteenable` corresponds to a byte in `writedata` and `readdata`. The host bit <n> of `byteenable` indicates whether byte <n> is being written to. During writes, `byteenables` specify which bytes are being written to. Other bytes should be ignored by the agent. During reads, `byteenables` indicate which bytes the host is reading. Agents that simply return `readdata` with no side effects are free to ignore `byteenables` during reads. If an interface does not have a `byteenable` signal, the transfer proceeds as if all `byteenables` are asserted. When more than one bit of the `byteenable` signal is asserted, all asserted lanes are adjacent.
`debugaccess`	1	Host → Agent	No	When asserted, allows the Nios II processor to write on-chip memories configured as ROMs.
`read` `read_n`	1	Host → Agent	No	Asserted to indicate a `read` transfer. If present, `readdata` is required.
`readdata`	8, 16, 32, 64, 128, 256, 512, 1024	Agent → Host	No	The `readdata` driven from the agent to the host in response to a `read` transfer. Required for interfaces that support reads.
`response [1:0]`	2	Agent → Host	No	The `response` signal is an optional signal that carries the response status. *Note:* Because the signal is shared, an interface cannot issue or accept a write response and a read response in the same clock cycle.

continued...

Signal Role	Width	Direction	Required	Description
				• 00: OKAY—Successful response for a transaction. • 01: RESERVED—Encoding is reserved. • 10: SLVERR—Error from an endpoint agent. Indicates an unsuccessful transaction. • 11: DECODEERROR—Indicates attempted access to an undefined location. For read responses: • One response is sent with each readdata. A read burst length of N results in N responses. Fewer responses are not valid, even in the event of an error. The response signal value may be different for each readdata in the burst. • The interface must have read control signals. Pipeline support is possible with the readdatavalid signal. • On read errors, the corresponding readdata is "don't care". For write responses: • One write response must be sent for each write command. A write burst results in only one response, which must be sent after the final write transfer in the burst is accepted. • If writeresponsevalid is present, all write commands must be completed with write responses.
write write_n	1	Host → Agent	No	Asserted to indicate a write transfer. If present, writedata is required.
writedata	8, 16, 32, 64, 128, 256, 512, 1024	Host → Agent	No	Data for write transfers. The width must be the same as the width of readdata if both are present. Required for interfaces that support writes.
Wait-State Signals				
lock	1	Host → Agent	No	lock ensures that once a host wins arbitration, the winning host maintains access to the agent for multiple transactions. Lock asserts coincident with the first read or write of a locked sequence of transactions. Lock deasserts on the final transaction of a locked sequence of transactions. lock assertion does not guarantee that arbitration is won. After the lock-asserting host has been granted, that host retains grant until lock is deasserted. A host equipped with lock cannot be a burst host. Arbitration priority values for lock-equipped hosts are ignored. lock is particularly useful for read-modify-write (RMW) operations. The typical read-modify-write operation includes the following steps: 1. Host A asserts lock and reads 32-bit data that has multiple bit fields. 2. Host A deasserts lock, changes one bit field, and writes the 32-bit data back. lock prevents host B from performing a write between Host A's read and write.
waitrequest waitrequest_n	1	Agent → Host	No	An agent asserts waitrequest when unable to respond to a read or write request. Forces the host to wait until the interconnect is ready to proceed with the transfer. At the start of all transfers, a host initiates the transfer and waits until waitrequest is deasserted. A host must make no assumption about the assertion state of waitrequest when the host is idle: waitrequest may be high or low, depending on system properties.

continued...

Signal Role	Width	Direction	Required	Description
				When `waitrequest` is asserted, host control signals to the agent must remain constant except for `beginbursttransfer`. For a timing diagram illustrating the `beginbursttransfer` signal, refer to the figure in *Read Bursts*. An Avalon memory mapped agent may assert `waitrequest` during idle cycles. An Avalon memory mapped host may initiate a transaction when `waitrequest` is asserted and wait for that signal to be deasserted. To avoid system lockup, an agent device should assert `waitrequest` when in reset.
Pipeline Signals				
`readdatavalid` `readdatavalid_n`	1	Agent → Host	No	Used for variable-latency, pipelined `read` transfers. When asserted, indicates that the `readdata` signal contains valid data. For a read burst with burstcount value <*n*>, the `readdatavalid` signal must be asserted <*n*> times, once for each readdata item. There must be at least one cycle of latency between acceptance of the `read` and assertion of `readdatavalid`. For a timing diagram illustrating the `readdatavalid` signal, refer to *Pipelined Read Transfer with Variable Latency*. An agent may assert `readdatavalid` to transfer data to the host independently of whether the agent is stalling a new command with `waitrequest`. Required if the host supports pipelined reads. Bursting hosts with read functionality must include the `readdatavalid` signal.
`writeresponsevalid`	1	Agent → Host	No	An optional signal. If present, the interface issues write responses for write commands. When asserted, the value on the response signal is a valid write response. `Writeresponsevalid` is only asserted one clock cycle or more after the write command is accepted. There is at least a one clock cycle latency from command acceptance to assertion of `writeresponsevalid`. A write command is considered accepted when the last beat of the burst is issued to the agent and `waitrequest` is low. `writeresponsevalid` can be asserted one or more clock cycles after the last beat of the burst has been issued.
Burst Signals				

continued...

Signal Role	Width	Direction	Required	Description
burstcount	1 – 11	Host → Agent	No	Used by bursting hosts to indicate the number of transfers in each burst. The value of the maximum burstcount parameter must be a power of 2. A burstcount interface of width <n> can encode a max burst of size $2^{(<n>-1)}$. For example, a 4-bit burstcount signal can support a maximum burst count of 8. The minimum burstcount is 1. The constantBurstBehavior property controls the timing of the burstcount signal. Bursting hosts with read functionality must include the readdatavalid signal. For bursting hosts and agents using byte addresses, the following restriction applies to the width of the address: *<address_w> >= <burstcount_w> + log2(<symbols_per_word_of_interface>)* For bursting hosts and agents using word addresses, the $\log_2$ term above is omitted.
beginbursttransfer	1	Interconnect → Agent	No	Asserted for the first cycle of a burst to indicate when a burst transfer is starting. This signal is deasserted after one cycle regardless of the value of waitrequest. For a timing diagram illustrating beginbursttransfer, refer to the figure in *Read Bursts*. beginbursttransfer is optional. An agent can always internally calculate the start of the next write burst transaction by counting data transfers. ***Warning:*** *do not* use this signal. This signal exists to support legacy memory controllers.

5.15.9. Avalon Streaming Interface Signal Roles

Each signal in an Avalon streaming source or sink interface corresponds to one Avalon streaming signal role. An Avalon streaming interface may contain only one instance of each signal role. All Avalon streaming signal roles apply to both sources and sinks and have the same meaning for both.

Table 112. Avalon Streaming Interface Signals

In the following table, all signal roles are active high.

Signal Role	Width	Direction	Required	Description
Fundamental Signals				
channel	1 – 128	Source → Sink	No	The channel number for data being transferred on the current cycle. If an interface supports the channel signal, the interface must also define the maxChannel parameter.
data	1 – 8,192	Source → Sink	No	The data signal from the source to the sink, typically carries the bulk of the information being transferred. Parameters further define the contents and format of the data signal.
error	1 – 256	Source → Sink	No	A bit mask to mark errors affecting the data being transferred in the current cycle. A single bit of the error signal masks each of the errors the component recognizes. The errorDescriptor defines the error signal properties.

continued...

Signal Role	Width	Direction	Required	Description
`ready`	1	Sink → Source	No	Asserts high to indicate that the sink can accept data. `ready` is asserted by the sink on cycle <n> to mark cycle <*n* + `readyLatency`> as a ready cycle. The source may only assert `valid` and transfer data during `ready` cycles. Sources without a `ready` input do not support backpressure. Sinks without a `ready` output never need to backpressure.
`valid`	1	Source → Sink	No	The source asserts this signal to qualify all other source to sink signals. The sink samples data and other source-to-sink signals on ready cycles where `valid` is asserted. All other cycles are ignored. Sources without a `valid` output implicitly provide valid data on every cycle that a sink is not asserting backpressure. Sinks without a `valid` input expect valid data on every cycle that they are not backpressuring.
Packet Transfer Signals				
`empty`	1 – 10	Source → Sink	No	Indicates the number of symbols that are empty, that is, do not represent valid data. The `empty` signal is not necessary on interfaces where there is one symbol per beat.
`endofpacket`	1	Source → Sink	No	Asserted by the source to mark the end of a packet.
`startofpacket`	1	Source → Sink	No	Asserted by the source to mark the beginning of a packet.

5.15.10. Avalon Streaming Credit Interface Signal Roles

Each signal in an Avalon Streaming Credit source or sink interface corresponds to one Avalon Streaming Credit signal role. An Avalon Streaming Credit interface may contain only one instance of each signal role. All Avalon Streaming Credit signal roles apply to both sources and sinks and have the same meaning for both.

Table 113. Avalon Streaming Credit Interface Signals

Signal Name	Direction	Width	Optional / Required	Description
`update`	Sink to source	1	Required	Sink sends `update` and source updates the available credit counter. Sink sends `update` to source when a transaction is popped from its buffer. Credit counter in source is increased by the value on the credit bus from sink to source.
`credit`	Sink to source	1-9	Required	Indicates additional credit available at sink when update is asserted. This bus carries a value as specified by the sink. Width of the `credit` bus is $ceilog_2$(MAX_CREDIT + 1). Sink sends available credit value on this bus which indicates the number of transactions it can accept. Source captures **`credit`** value only if **`update`** signal is asserted.

continued...

Signal Name	Direction	Width	Optional / Required	Description
return_credit	Source to sink	1	Required	Asserted by source to return 1 credit back to sink. *Note:* For more details, refer to Section 6.2.3 *Returning the Credits*.
data	Source to sink	1-8192	Required	Data is divided into symbols as per existing Avalon Streaming definition.
valid	Source to sink	1	Required	Asserted by the source to qualify all other source to sink signals. Source can assert `valid` only when the credit available to it is greater than 0.
error	Source to sink	1-256	Optional	A bit mask used to mark errors affecting the data being transferred in the current cycle. A single bit in error is used for each of the errors recognized by the component, as defined by the `errorDescriptor` property.
channel	Source to sink	1-128	Optional	The channel number for data being transferred on the current cycle. If an interface supports the `channel` signal, it must also define the `maxChannel` parameter.
Packet Transfer Signals				
startofpacket	Source to sink	1	Optional	Asserted by the source to mark the start of a packet.
endofpacket	Source to sink	1	Optional	Asserted by the source to mark the end of a packet.
empty	Source to sink	ceil(log2(NUM_SYMBOLS))	Optional	Indicates the number of symbols that are empty, that is, do not represent valid data. The `empty` signal is not used on interfaces where there is one symbol per beat.
User Signals				
<Per-Packet User Signals>	Source to sink	1-8192	Optional	Any number of per-packet user signals can be present on source and sink interfaces. Source sets value of this signal when `startofpacket` is asserted. Source should not change the value of this signal until start of new packet. More details are in the User Signal section.
<Per-Symbol User Signals>	Source to sink	1-8192	Optional	Any number of per-symbol user signals can be present on source and sink. More details are in the User Signal section.

5.15.10.1. Synchronous Interface

All transfers of an Avalon Streaming connection occur synchronous to the rising edge of the associated clock signal. All outputs from a source interface to a sink interface, including the `data`, `channel`, and `error` signals, must be registered on the rising edge of clock. Inputs to a sink interface do not have to be registered. Registering signals at the source facilitates high-frequency operation.

Table 114. Avalon Streaming Credit Interface Properties

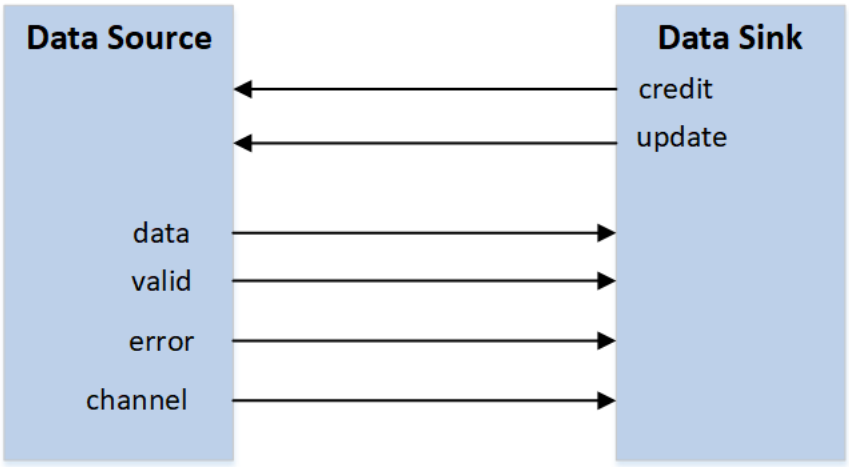
Property Name	Default Value	Legal Value	Description
associatedClock	1	Clock interface	The name of the Avalon Clock interface to which this Avalon Streaming interface is synchronous.
associatedReset	1	Reset interface	The name of the Avalon Reset interface to which this Avalon Streaming interface is synchronous.
dataBitsPerSymbol	8	1 – 8192	Defines the number of bits per symbol. For example, byte-oriented interfaces have 8-bit symbols. This value is not restricted to be a power of 2.
symbolsPerBeat	1	1 – 8192	The number of symbols that are transferred on every valid cycle.
maxCredit	256	1-256	The maximum number of credits that a data interface can support.
errorDescriptor	0	List of strings	A list of words that describe the error associated with each bit of the error signal. The length of the list must be the same as the number of bits in the error signal. The first word in the list applies to the highest order bit. For example, "crc, overflow" means that bit[1] of error indicates a CRC error. Bit[0] indicates an overflow error.
firstSymbolInHighOrderBits	true	true, false	When true, the first-order symbol is driven to the most significant bits of the data interface. The highest-order symbol is labeled D0 in this specification. When this property is set to false, the first symbol appears on the low bits. D0 appears at data[7:0]. For a 32-bit bus, if true, D0 appears on bits[31:24].
maxChannel	0	0	The maximum number of channels that a data interface can support.

5.15.10.2. Typical Data Transfers

This section defines the transfer of data from a source interface to a sink interface. In all cases, the data source and the data sink must comply with the specification. It is not the responsibility of the data sink to detect source protocol errors.

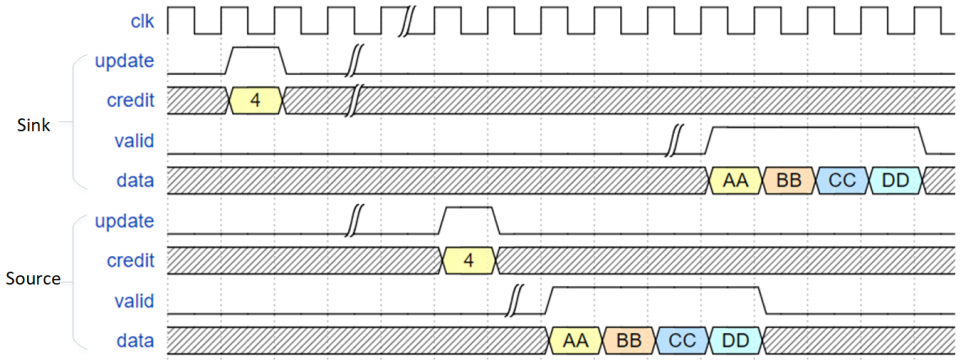
The below figure shows the signals that are typically used in an Avalon Streaming Credit interface.

Figure 244. Typical Avalon Streaming Credit Signals



As this figure indicates, a typical Avalon Streaming Credit source interface drives the `valid`, `data`, `error`, and `channel` signals to the sink. The sink drives `update` and `credit` signals.

Figure 245. Typical Credit and Data Transfer



The above figure shows a typical credit and data transfer between source and sink. There can be an arbitrary delay between the sink asserting `update` and source receiving the `update`. Similarly, there can be an arbitrary delay between source asserting `valid` for data and sink receiving that data. Delay on credit path from sink to source and data path from source to sink need not be equal. These delays can be 0 cycle as well, i.e. when the sink asserts `update`, it is seen by the source in the same cycle. Conversely, when the source asserts `valid`, it is seen by the sink in the same cycle. If source has zero credits, it cannot assert `valid`. Transferred credits are cumulative. If sink has transferred credits equal to its maxCredit property, and has not received any data, it cannot assert `update` until it receives at least 1 data or has received a `return_credit` pulse from the source.

Sink cannot backpressure data from source if sink has provided credits to the source, i.e. sink must accept data from source if there are outstanding credits. Source cannot assert `valid` if it has not received any credit or exhausted the credits received, i.e. already sent the data in lieu of credits received.

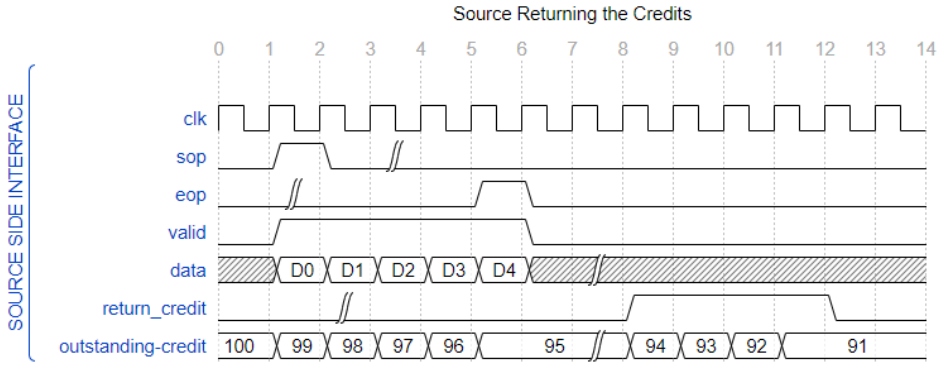
If source has zero credits, source cannot start the data transfer in the same cycle it receives credits. Similarly, if sink has transferred credits equal to its maxCredit property and it receives data, sink cannot send an update in the same cycle as it received data. These restrictions have been put in place to avoid combinational loops in the implementation.

5.15.10.3. Returning the Credits

Avalon Streaming Credit protocol supports a `return_credit` signal. This is used by source to return the credits back to sink. Every cycle this signal is asserted, it indicates source is giving back 1 credit. If source wants to return multiple credits, this signal needs to be asserted for multiple cycles. For example, if source wants to return 10 outstanding credits, it asserts `return_credit` signal for 10 cycles. Sink should account for returned credits in its internal credit maintenance counters. Credits can be returned by source at any point in time as long as it has credits greater than 0.

The below figure exemplifies source returning credits. As shown in the figure, *outstanding_credit* is an internal counter for the source. When source returns credits, this counter is decremented.

Figure 246. Source Returning Credits



Note: Although the diagram above shows the returning of credits when `valid` is deasserted, `return_credit` can also be asserted while `valid` is asserted. In this case, source effectively spends 2 credits: one for `valid`, and one for `return_credit`.

5.15.11. Avalon Streaming Credit User Signals

User signals are optional sideband signals which flow along with data. They are considered valid only when data is valid. Given that user signals do not have any defined meaning or purpose, caution must be used while using these signals. It is the responsibility of the system designer to make sure that two IPs connected to each other agree on the roles of the user signals.

Two types of user signals are being proposed: per-symbol user signals and per-packet user signals.

5.15.11.1. Per-Symbol User Signal

As the name suggests, the data defines a per-symbol user signal (`symbol_user`) per symbol. Each symbol in the data can have a user signal. For example, if the number of symbols in the data is 8, and `symbol_user` width is 2 bits, the total width of the `symbol_user` signal is 16 bits.

`Symbol_user` is valid only when data is valid. Source can change this signal every cycle when data is valid. Sink can disregard the value of `symbol_user` bits for empty symbols.

If a source which has this signal is connected to a sink which does not have this signal on its interface, the signal from source remains dangling in the generated interconnect.

If a source which does not have this signal is connected to a sink which has this signal on its interface, the sink's input user signal ties to 0.

If both source and sink have equal number of symbols in the data, then the user signals for both must have equal widths. Otherwise, they cannot be connected.

If a wide source is connected to a narrow sink, and both have per-symbol user signals, then both must have equal bits of user signal associated with each symbol. For example, if a 16-symbol source has 2 bits of user signal associated with each symbol (for a total of 32 bits of user signal), then a 4-symbol sink must have an 8-bit wide user signal (2 bits associated with each symbol). A data format adapter can convert the 16-symbol source data to 4-symbol sink data, and 32-bit user signal to 8-bit user signal. The data format adapter maintains the association of symbols with corresponding user signal bits.

Similarly, if a narrow source is connected to a wide sink, and both have per-symbol user signals, then both must have equal bits of user signal associated with each symbol. For example, if a 4-symbol source has 2 bits of user signal associated with each symbol (for a total of 8 bits of user signal), then a 16-symbol sink must have a 32-bit wide user signal (2 bits associated with each symbol). A data format adapter can convert the 4-symbol source data to 16-symbol sink data, and 8-bit user signal to 32-bit user signal. The data format adapter maintains the association of symbols with corresponding user signal bits. If the packet is smaller than the ratio of data widths, the data format adapter sets the value of empty accordingly. Sink should disregard the value of user bits associated with empty symbols.

5.15.11.2. Per-Packet User Signal

In addition to `symbol_user`, per-packet user signals (`packet_user`) can also be declared on the interface. `Packet_user` can be of arbitrary width. Unlike `symbol_user`, `packet_user` must remain constant throughout the packet, i.e. its value should be set at the start of the packet and must remain the same until the end of the packet. This restriction makes the implementation of the data format adapter simpler as it eliminates the option to replicate or chop (wide source, narrow sink) or concatenate (narrow source, wide sink) `packet_user`.

If a source has `packet_user` and sink does not, the `packet_user` from source remains dangling. In such a case, the system designer must be careful and not transmit any critical control information on this signal as it is completely or partially ignored.

If a source does not have `packet_user` and the sink does, the `packet_user` to sink is tied to 0.

5.15.12. Avalon Clock Source Signal Roles

An Avalon Clock source interface drives a clock signal out of a component.

Table 115. Clock Source Signal Roles

Signal Role	Width	Direction	Required	Description
`clk`	1	Output	Yes	An output clock signal.

5.15.13. Avalon Clock Sink Signal Roles

A clock sink provides a timing reference for other interfaces and internal logic.

Table 116. Clock Sink Signal Roles

Signal Role	Width	Direction	Required	Description
`clk`	1	Input	Yes	A clock signal. Provides synchronization for internal logic and for other interfaces.

5.15.14. Avalon Conduit Signal Roles

Table 117. Conduit Signal Roles

Signal Role	Width	Direction	Description
<any>	<*n*>	In, out, or bidirectional	A conduit interface consists of one or more input, output, or bidirectional signals of arbitrary width. Conduits can have any user-specified role. You can connect compatible Conduit interfaces inside a Platform Designer system provided the roles and widths match and the directions are opposite.

5.15.15. Avalon Tristate Conduit Signal Roles

The following table lists the signal defined for the Avalon Tristate Conduit interface. All Avalon-TC signals apply to both hosts and agents and have the same meaning for both.

Table 118. Tristate Conduit Interface Signal Roles

Signal Role	Width	Direction	Required	Description
`request`	1	Host → Agent	Yes	The meaning of `request` depends on the state of the `grant` signal, as the following rules dictate. When `request` is asserted and `grant` is deasserted, `request` is requesting access for the current cycle. When `request` is asserted and `grant` is asserted, `request` is requesting access for the next cycle. Consequently, `request` should be deasserted on the final cycle of an access. The `request` signal deasserts in the last cycle of a bus access. The `request` signal can reassert immediately following the final cycle of a transfer.

continued...

Signal Role	Width	Direction	Required	Description
				This protocol makes both rearbitration and continuous bus access possible if no other hosts are requesting access. Once asserted, `request` must remain asserted until granted. Consequently, the shortest bus access is 2 cycles. Refer to *Tristate Conduit Arbitration Timing* for an example of arbitration timing.
`grant`	1	Agent → Host	Yes	When asserted, indicates that a tristate conduit host has access to perform transactions. The `grant` signal asserts in response to the `request` signal. The `grant` signal remains asserted until 1 cycle following the deassertion of `request`.
<*name*>_in	1 – 1024	Agent → Host	No	The input signal of a logical tristate signal.
<*name*>_out	1 – 1024	Host → Agent	No	The output signal of a logical tristate signal.
<*name*>_outen	1	Host → Agent	No	The output enable for a logical tristate signal.

5.15.16. Avalon Tri-State Agent Interface Signal Types

Table 119. Tri-state Agent Interface Signal Types

Name	Width	Direction	Required	Description
`address`	1 - 32	input	No	Address lines to the agent port. Specifies a byte offset into the agent's address space.
`read` `read_n`	1	input	No	Read-request signal. Not required if the agent port never outputs data. If present, data must also be used.
`write` `write_n`	1	input	No	Write-request signal. Not required if the agent port never receives data from a host. If present, data must also be present, and `writebyteenable` cannot be present.
`chipselect` `chipselect_n`	1	input	No	When present, the agent port ignores all Avalon memory mapped signals unless `chipselect` is asserted. `chipselect` is always present in combination with read or write
`outputenable` `outputenable_n`	1	input	Yes	Output-enable signal. When deasserted, a tri-state agent port must not drive its data lines otherwise data contention may occur.
`data`	8,16, 32, 64, 128, 256, 512, 1024	bidir	No	Bidirectional data. During write transfers, the FPGA drives the data lines. During read transfers the agent device drives the data lines, and the FPGA captures the data signals and provides them to the host.
`byteenable` `byteenable_n`	2, 4, 8,16, 32, 64, 128	input	No	Enables specific byte lanes during transfers. Each bit in byteenable corresponds to a byte lane in data. During writes, `byteenables` specify which bytes the host is writing to the agent. During

continued...

Name	Width	Direction	Required	Description
				reads, `byteenables` indicates which bytes the host is reading. Agents that simply return data with no side effects are free to ignore `byteenables` during reads. When more than one byte lane is asserted, all asserted lanes are guaranteed to be adjacent. The number of adjacent lines must be a power of 2, and the specified bytes must be aligned on an address boundary for the size of the data. The following are legal values for a 32-bit agent: `1111 writes full 32 bits` `0011 writes lower 2 bytes` `1100 writes upper 2 bytes` `0001 writes byte 0 only` `0010 writes byte 1 only` `0100 writes byte 2 only` `1000 writes byte 3 only`
`writebyteenable` `writebyteenable_n`	2,4,8,16, 32, 64,128	input	No	Equivalent to the logical AND of the `byteenable` and write signals. When used, the write signal is not used.
`begintransfer1`	1	input	No	Asserted for the first cycle of each transfer.
Note: All Avalon signals are active high. Avalon signals that can also be asserted low list both versions in the **Signal Role** column.				

5.15.17. Avalon Interrupt Sender Signal Roles

Table 120. Interrupt Sender Signal Roles

Signal Role	Width	Direction	Required	Description
irq irq_n	1-2048	Output	Yes	Interrupt Request. An interrupt sender drives an interrupt signal to an interrupt receiver.

5.15.18. Avalon Interrupt Receiver Signal Roles

Table 121. Interrupt Receiver Signal Roles

Signal Role	Width	Direction	Required	Description
irq	1–2048	Input	Yes	irq is an <*n*>-bit vector, where each bit corresponds directly to one IRQ sender with no inherent assumption of priority.

5.16. Platform Designer Interconnect Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2023.10.02	23.3	• Removed misplaced Reset Synchronous Edges description from *Clock Interfaces* topic. • Removed obsolete upper limit from *IRQ Fan-Out* topic.
2023.04.03	23.1	• The product family name is updated to "Intel Agilex 7" to reflect the different family members.
2022.06.20	22.2	• Added new *IRQ Fanout* topic. • Updated *IRQ Mapper* topic to describe Remove Clock and Reset Ports parameters. • Updated legal IRQ width from 32 to 2048 in the following topics: — *Interrupt Interfaces* — *Individual Requests IRQ Scheme* — *Avalon Interrupt Sender Signal Roles* — *Avalon Interrupt Receiver Signal Roles*
2022.04.02	22.1	• Updated entire chapter for new AXI "manager" and AXI "subordinate" replacement terms. Refer to the AMBA® AXI and ACE Protocol Specification. • Updated parameter values for all *Avalon Streaming Credit* IPs. • Revised wording in *Terms and Concepts* topic for greater clarity. • Revised AMBA 4 AXI-Lite Signals topic for optional signals information. • Added new AMBA 4 AXI-Lite Optional Port Support and Interconnect topic describing optional port support details.
2021.03.29	21.1	• Converted to "host" and "agent" inclusive terminology for Avalon memory mapped interface descriptions and related GUI elements throughout.

continued...

Document Version	Quartus Prime Version	Changes
2020.12.14	20.4	• Revised "Interconnect Pipelining" for clarity and latest GUI. • Revised "Fixed Priority Arbitration" for latest GUI. • Revised "Add Pipeline Stages in the Interconnect Schematic" for Add All Pipelines and Remove All Pipelines controls. • Revised statement in "Ordering Model" topic to clarify applies to all versions. • Revised "Pipeline Placement in Arbitration Logic" diagram color coding.
2020.06.26	20.1	Removed incorrect statement about compile time option to enforce strict ordering from the "AXI and Avalon Ordering" topic.
2020.05.08	20.1	Added some clarification for the timing behavior of the signal `writeresponsevalid` to the *Avalon Memory-Mapped Interface Signal Roles* section. Updated the bus widths for the `data` and `empty` signals in the *Avalon Streaming Interface Signal Roles* section.
2020.05.01	20.1	• Updated "Avalon Streaming Credit Interface Signal Roles" to indicate that return_credit is required. • Added "Avalon Streaming Credit Interfaces" section. • Added "Avalon Streaming Credit Interface Signal Roles" topic. • Added "Avalon Streaming Credit User Signals" topic.
2019.11.11	19.1.0	• Added note to "Burst Adaptation: AXI to Avalon" about AXI3 and AXI4 4KB boundary restriction for burst transactions. • Added "Adjacent Bytelanes with Partial Width Transactions" topic.
2019.06.19	19.1.0	• Corrected statement about preventing reordering in "Ordering Model."
2019.04.01	19.1.0	• Described new default use of synchronous reset option for Stratix 10 designs in "Reset Interfaces."
2018.12.10	18.1.0	• Replaced references to System Contents tab with new System View tab.
2018.09.24	18.1.0	• Updated location of **Limit interconnect pipeline stages to** option in Platform Designer GUI • In *Avalon Memory-Mapped Interface Signal Roles*, added consecutive byte-enable support. • Specified minimum duration of reset that the Platform Design Interconnect requires to work correctly.
2018.06.15	18.0.0	Clarified behavior of Error Correction Coding (ECC) in Interconnect.
2018.05.07	18.0.0	• Added support for `waitrequestAllowance` adapter. • Added support for ACE-Lite connections.
2017.11.06	17.1.0	• Changed instances of *Qsys Pro* to Platform Designer • Updated information about the Reset Sequencer.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented *Qsys* rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Fixed Priority Arbitration. • Added topic: *Read and Write Responses*. • Added topic: *Error Correction Coding (ECC) in Qsys Interconnect*. • Added: `response [1:0]`, *Avalon Memory-Mapped Interface Signal Roles*. • Added `writeresponsevalid`, *Avalon Memory-Mapped Interface Signal Roles*.

continued...

Document Version	Quartus Prime Version	Changes
December 2014	14.1.0	• Read error responses, Avalon Memory-Mapped Interface Signal, `response`. • Burst Adapter Implementation Options: Generic converter (slower, lower area), Per-burst-type converter (faster, higher area).
August 2014	14.0a10.0	• Updated Qsys Packet Format for Memory-Mapped Host and Agent Interfaces table, `Protection`. • Streaming Interface renamed to Avalon Streaming Interfaces. • Added *Response Merging* under *Memory-Mapped Interfaces*.
June 2014	14.0.0	• AXI4-Lite support. • AXI4-Stream support. • Avalon streaming adapter parameters. • IRQ Bridge. • Handling Read Side Effects note added.
November 2013	13.1.0	• HSSI clock support. • Reset Sequencer. • Interconnect pipelining.
May 2013	13.0.0	• AMBA APB support. • Auto-inserted Avalon streaming adapters feature. • Moved Address Span Extender to the *Qsys System Design Components* chapter.
November 2012	12.1.0	• AMBA AXI4 support.
June 2012	12.0.0	• AMBA AXI3 support. • Avalon streaming adapters. • Address Span Extender.
November 2011	11.0.1	Template update.
May 2011	11.0.0	Removed beta status.
December 2010	10.1.0	Initial release.

6. Platform Designer System Design Components

You can use Platform Designer IP components to create Platform Designer systems. Platform Designer interfaces include components appropriate for streaming high-speed data, reading and writing registers and memory, controlling off-chip devices, and transporting data between components.

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

Related Information

- Creating a System with Platform Designer on page 11
- Platform Designer Interconnect on page 251
- Embedded Peripherals IP User Guide
- Avalon Interface Specifications

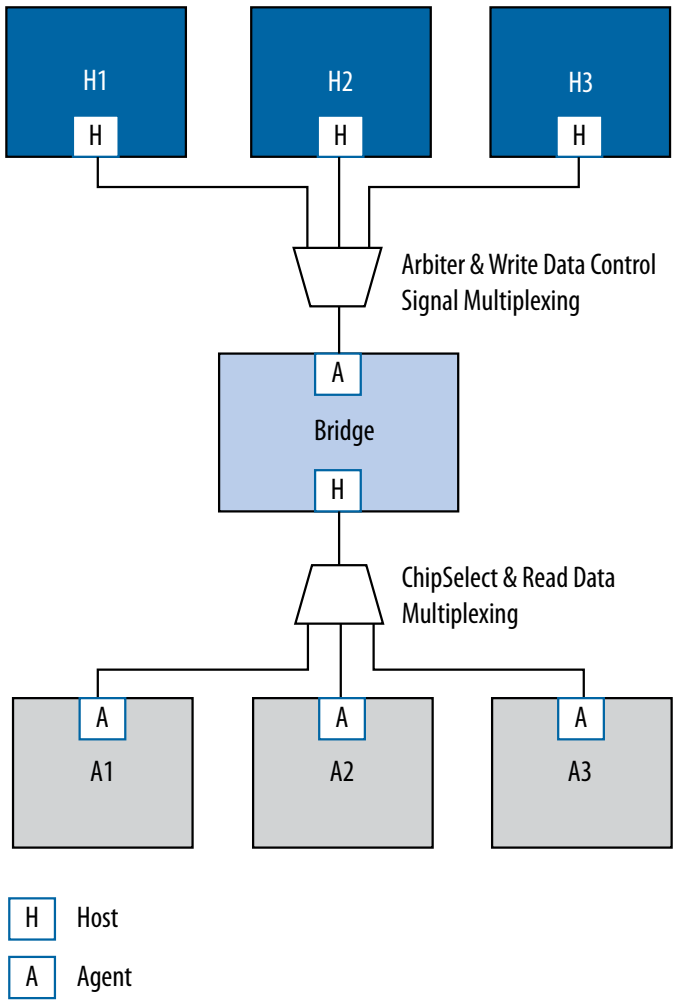
6.1. Bridges

Bridges affect the way Platform Designer transports data between components. You can insert bridges between host[24] and agent interfaces to control the topology of a Platform Designer system, which affects the interconnect that Platform Designer generates. You can also use bridges to separate components into different clock domains to isolate clock domain crossing logic.

A bridge has one agent interface and one host interface. In Platform Designer, one or more host interfaces from other components connect to the bridge agent. The bridge host connects to one or more agent interfaces on other components. In the following example, three hosts have logical connections to three agents, although physically each host connects only to the bridge. Transfers initiated to the agent propagate to the host in the same order in which the transfers are initiated on the agent.

[24] Platform Designer now replaces non-inclusive terms with "host" and "agent" inclusive terms for Avalon memory mapped interface descriptions and related GUI elements.

Figure 247. Using a Bridge in a Platform Designer System

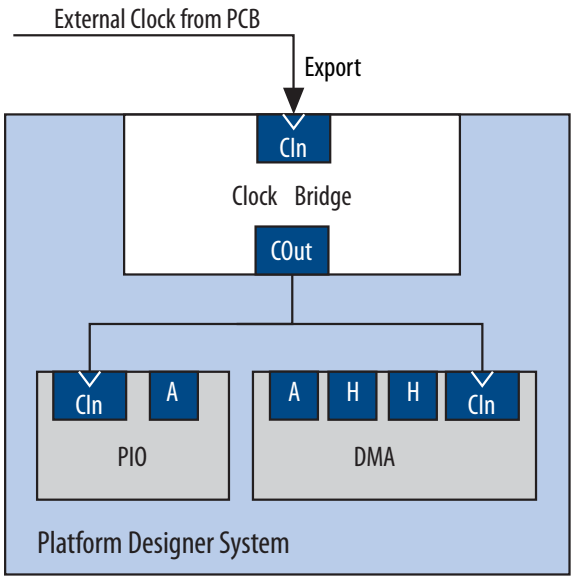


6.1.1. Clock Bridge Intel FPGA IP

The Clock Bridge Intel FPGA IP connects a clock source to multiple clock input interfaces. You can use the clock bridge to connect a clock source that is outside the Platform Designer system. Create the connection through an exported interface, and then connect to multiple clock input interfaces.

Clock outputs match fan-out performance without the use of a bridge. You require a bridge only when you want a clock from an exported source to connect internally to more than one source.

Figure 248. Clock Bridge Intel FPGA IP



6.1.2. Avalon Memory Mapped Clock Crossing Bridge Intel FPGA IP

The Avalon Memory Mapped Clock Crossing Bridge Intel FPGA IP transfers Avalon memory mapped commands and responses between different clock domains. You can also use the Avalon Memory Mapped Clock Crossing Bridge between AXI hosts and agents of different clock domains.

The Avalon Memory Mapped Clock Crossing Bridge uses asynchronous FIFOs to implement clock crossing logic. The bridge parameters control the depth of the command and response FIFOs in both the host and agent clock domains. If the number of active reads exceeds the depth of the response FIFO, the Clock Crossing Bridge stops sending reads.

To maintain throughput for high-performance applications, increase the response FIFO depth from the default minimum depth, which is twice the maximum burst size.

Note: When you use the FIFO-based clock crossing a Platform Designer system, the DC FIFO is automatically inserted in the Platform Designer system. The reset inputs for the DC FIFO connect to the reset sources for the connected host and agent components on either side of the DC FIFO. For this configuration, you must assert both the resets on the host and the agent sides at the same time to ensure the DC FIFO resets properly. Alternatively, you can drive both resets from the same reset source to guarantee that the DC FIFO resets properly.

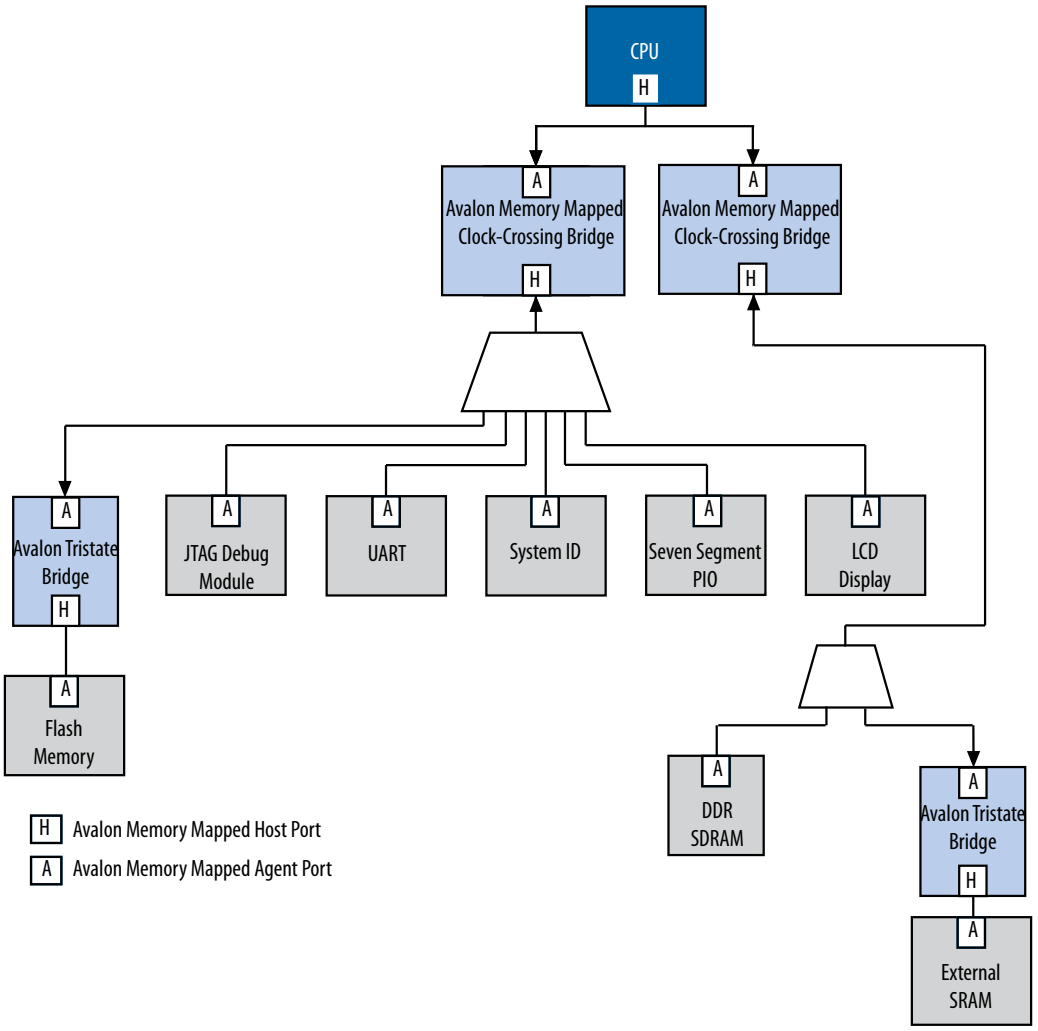
Note: The clock crossing bridge includes appropriate SDC constraints for its internal asynchronous FIFOs. For these SDC constraints to work correctly, do not set false paths on the pointer crossings in the FIFOs. Do not split the bridge's clocks into separate clock groups when you declare SDC constraints; the split has the same effect as setting false paths.

6.1.2.1. Avalon Memory Mapped Clock Crossing Bridge Example

In the example shown below, the Avalon Memory Mapped Clock Crossing bridges separate agent components into two groups. The Avalon Memory Mapped Clock Crossing Bridge places the low performance agent components behind a single bridge and clocks the components at a lower speed. The bridge places the high-performance components behind a second bridge and clocks it at a higher speed.

By inserting clock-crossing bridges, you simplify the Platform Designer interconnect and allow the Quartus Prime Fitter to optimize paths that require minimal propagation delay.

Figure 249. Avalon Memory Mapped Clock Crossing Bridge



6.1.2.2. Avalon Memory Mapped Clock Crossing Bridge Parameters

Table 122. Avalon Memory Mapped Clock Crossing Bridge Parameters

Parameters	Values	Description
Data width	8, 16, 32, 64, 128, 256, 512, 1024 bits	Determines the data width of the interfaces on the bridge, and affects the size of both FIFOs. For the highest bandwidth, set **Data width** to be as wide as the widest host that connects to the bridge.
Symbol width	1, 2, 4, 8, 16, 32, 64 (bits)	Number of bits per symbol. For example, byte-oriented interfaces have 8-bit symbols.
Address width	1-32 bits	The address bits needed to address the downstream agents.
Use automatically-determined address width	-	The minimum bridge address width that is required to address the downstream agents.
Maximum burst size	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 bits	Determines the maximum length of bursts that the bridge supports.
Command FIFO depth	2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 bits	Command (host-to-agent) FIFO depth.
Respond FIFO depth	2, 4, 8,16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 bits	agent-to-host FIFO depth.
Master clock domain synchronizer depth	2, 3, 4, 5 bits	The number of pipeline stages in the clock crossing logic in the issuing host to target agent direction. Increasing this value leads to a larger mean time between failures (MTBF). You can determine the MTBF for a design by running a timing analysis.
Slave clock domain synchronizer depth	2, 3, 4, 5 bits	The number of pipeline stages in the clock crossing logic in the target agent to host direction. Increasing this value leads to a larger meantime between failures (MTBF). You can determine the MTBF for a design by running a timing analysis.

6.1.3. Avalon Memory Mapped Pipeline Bridge Intel FPGA IP

The Avalon Memory Mapped Pipeline Bridge Intel FPGA IP inserts a register stage in the Avalon memory mapped command and response paths. The bridge accepts commands on its agent port and propagates the commands to its host port. The pipeline bridge provides separate parameters to turn on pipelining for command and response signals.

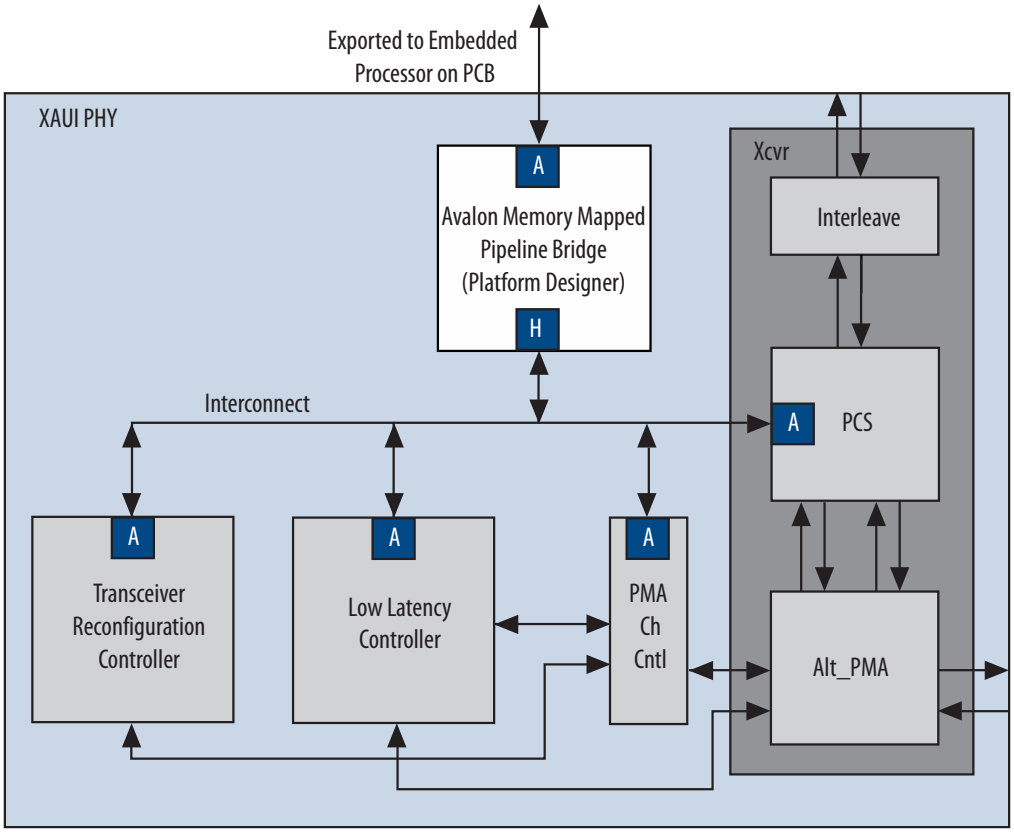
Note: The Avalon Memory Mapped Pipeline Bridge Intel FPGA IP has a `readLatency` of `0` and `readdatavalid` of `1`. Therefore, the bridge does not support fixed-latency pipelined read transfers.

The **Maximum pending read transactions** parameter is the maximum number of pending reads that the Avalon Memory Mapped bridge can queue up. To determine the best value for this parameter, review this same option for the bridge's connected agents and identify the highest value of the parameter, and then add the internal buffering requirements of the Avalon Memory Mapped bridge. In general, the value is between 4 and 32. The limit for maximum queued transactions is 64.

You can use the Avalon Memory Mapped bridge to export a single Avalon memory mapped agent interface to control multiple Avalon memory mapped agent devices. The pipelining feature is optional.

Figure 250. Avalon Memory Mapped Pipeline Bridge IP and XAUI PHY Transceiver IP

In this example, the bridge transfers commands received on its agent interface to its host port.

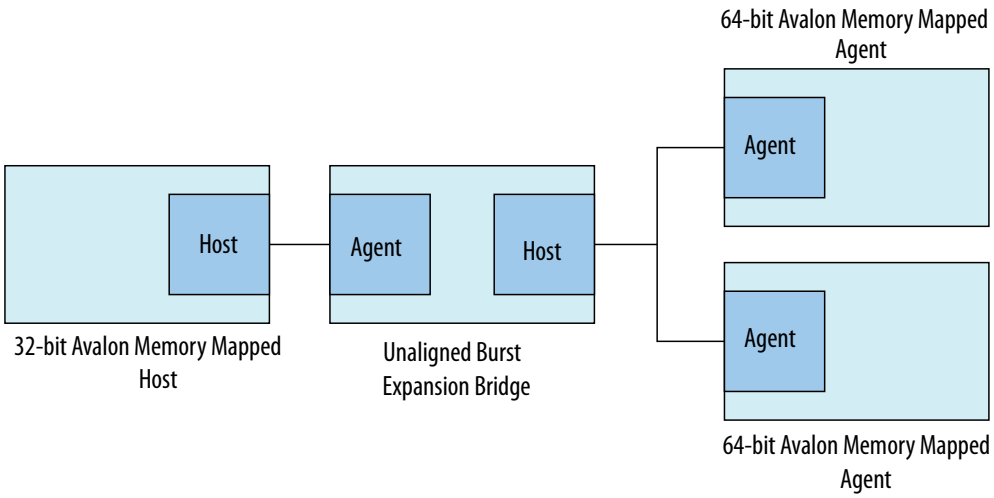


Because the agent interface is exported to the pins of the device, having a single agent port, rather than separate ports for each agent device, reduces the pin count of the FPGA. Refer to Interconnect Pipelining on page 328 for more information.

6.1.4. Avalon Memory Mapped Unaligned Burst Expansion Bridge Intel FPGA IP

The Avalon Memory Mapped Unaligned Burst Expansion Bridge Intel FPGA IP aligns read burst transactions from hosts connected to its agent interface, to the address space of agents connected to its host interface. This alignment ensures that all read burst transactions are delivered to the agent as a single transaction.

Figure 251. Avalon Memory Mapped Unaligned Burst Expansion Bridge Intel FPGA IP



You can use the Avalon Unaligned Burst Expansion Bridge to align read burst transactions from hosts that have narrower data widths than the target agents. Using the bridge for this purpose improves bandwidth utilization for the host-agent pair, and ensures that unaligned bursts are processed as single transactions rather than multiple transactions.

Note: Do not use the Avalon Memory Mapped Unaligned Burst Expansion Bridge if any connected agent has read side effects from reading addresses that are exposed to any connected host's address map. This bridge can cause read side effects due to alignment modification to read burst transaction addresses.

Note: The Avalon Memory Mapped Unaligned Burst Expansion Bridge does not support VHDL simulation.

6.1.4.1. Using the Avalon Memory Mapped Unaligned Burst Expansion Bridge

When a host sends a read burst transaction to an agent, the Avalon Memory Mapped Unaligned Burst Expansion Bridge initially determines whether the start address of the read burst transaction is aligned to the agent's memory address space. If the base address is aligned, the bridge does not change the base address. If the base address is not aligned, the bridge aligns the base address to the nearest aligned address that is less than the requested base address.

The Avalon Memory Mapped Unaligned Burst Expansion Bridge then determines whether the final word requested by the host is the last word at the agent read burst address. If a single agent address contains multiple words, all those words must be requested for a single read burst transaction to occur.

- If the final word requested by the host is the last word at the agent read burst address, the bridge does not modify the burst length of the read burst command to the agent.
- If the final word requested by the host is not the last word at the agent read burst address, the bridge increases the burst length of the read burst command to the agent. The final word requested by the modified read burst command is then the last word at the agent read burst address.

The bridge stores information about each aligned read burst command that it sends to agents connected to a host interface. When a read response is received on the host interface, the bridge determines if the base address or burst length of the issued read burst command was altered.

If the bridge alters either the base address or the burst length of the issued read burst command, it receives response words that the host did not request. The bridge suppresses words that it receives from the aligned burst response that are not part of the original read burst command from the host.

6.1.4.2. Avalon Memory Mapped Unaligned Burst Expansion Bridge Parameters

Figure 252. Avalon Memory Mapped Unaligned Burst Expansion Bridge Parameter Editor

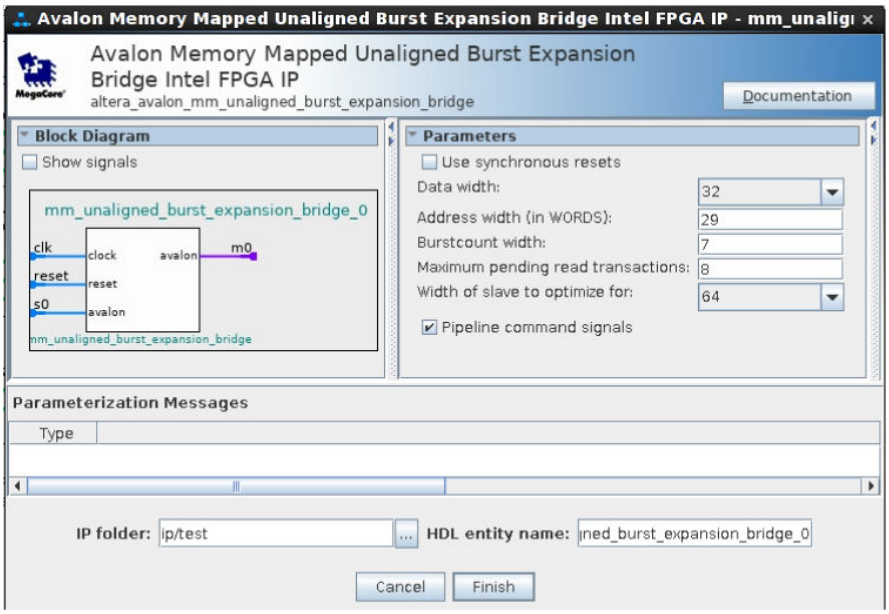


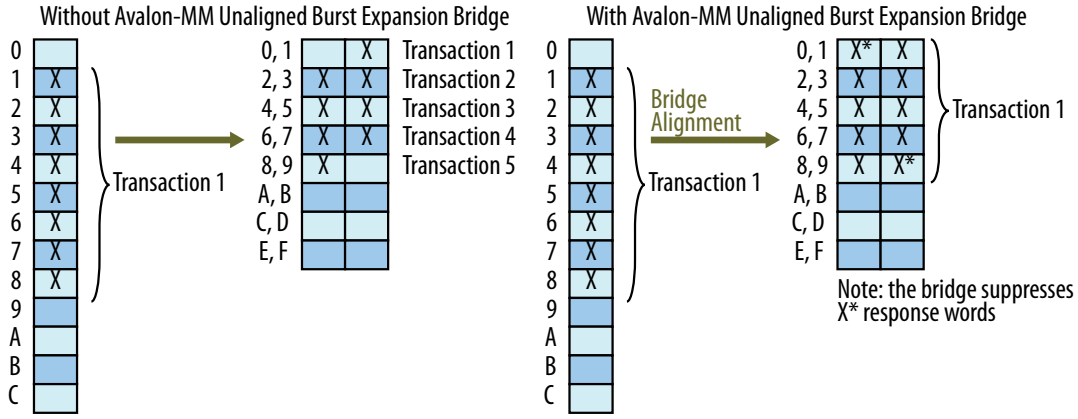
Table 123. Avalon Memory Mapped Unaligned Burst Expansion Bridge Parameters

Parameter	Description
Data width	Data width of the host connected to the bridge.
Address width (in WORDS)	The address width of the host connected to the bridge.
Burstcount width	The burstcount signal width of the host connected to the bridge.
Maximum pending read transactions	The **Maximum pending read transactions** parameter is the maximum number of pending reads that the Avalon Memory Mapped bridge can queue up. To determine the best value for this parameter, review this same option for the bridge's connected agents and identify the highest value of the parameter, and then add the internal buffering requirements of the Avalon Memory Mapped bridge. In general, the value is between 4 and 32. The limit for maximum queued transactions is 64.
Width of agent to optimize for	The data width of the connected agent. Supported values are: 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096 bits. *Note:* If you connect multiple agents, all agents must have the same data width.
Pipeline command signals	When turned on, the command path is pipelined, minimizing the bridge's critical path at the expense of increased logic usage and latency.

6.1.4.3. Avalon Memory Mapped Unaligned Burst Expansion Bridge Example

The following example shows an unaligned read burst command from a host that the Avalon Memory Mapped Unaligned Burst Expansion Bridge converts to an aligned request for a connected agent. The figure also shows the suppression of words due to the aligned read burst command. In this example, a 32-bit host requests an 8-beat burst of 32-bit words from a 64-bit agent with a start address that is not 64-bit aligned.

Figure 253. Unaligned Burst Expansion Bridge



Because the target agent has a 64-bit data width, address 1 is unaligned in the agent's address space. As a result, several smaller burst transactions are needed to request the data associated with the host's read burst command.

With an Avalon memory mapped Unaligned Burst Expansion Bridge in place, the bridge issues a new read burst command to the target agent beginning at address 0 with burst length 10, which requests data up to the word stored at address 9.

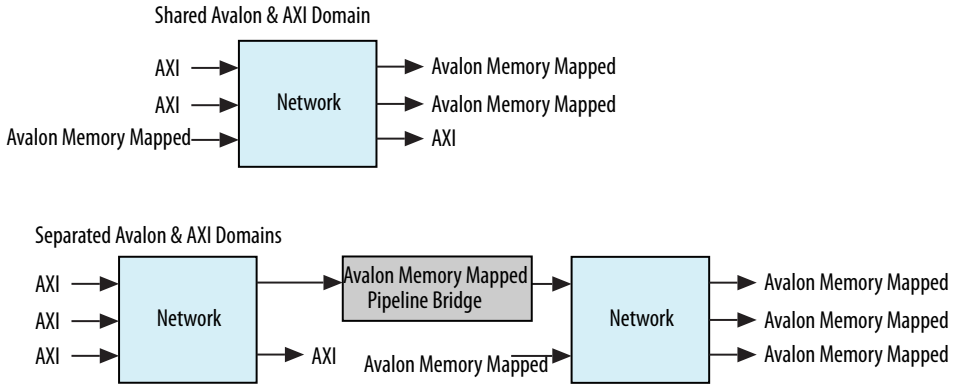
When the bridge receives the word corresponding to address 0, it suppresses it from the host, and then delivers the words corresponding to addresses 1 through 8 to the host. When the bridge receives the word corresponding to address 9, it suppresses that word from the host.

6.1.5. Bridges Between Avalon and AXI Interfaces

When designing a Platform Designer system, you can make connections between AXI and Avalon interfaces without the use of explicitly-instantiated bridges; the interconnect provides all necessary bridging logic. However, this does not prevent the use of explicit bridges to separate the AXI and Avalon domains. Using an explicit

Avalon memory mapped bridge to separate the AXI and Avalon domains reduces the amount of bridging logic in the interconnect at the expense of concurrency, as the following example shows:

Figure 254. Avalon Memory Mapped Pipeline Bridge Between Avalon Memory-Mapped and AXI Domains



6.1.6. AXI Bridge Intel FPGA IP

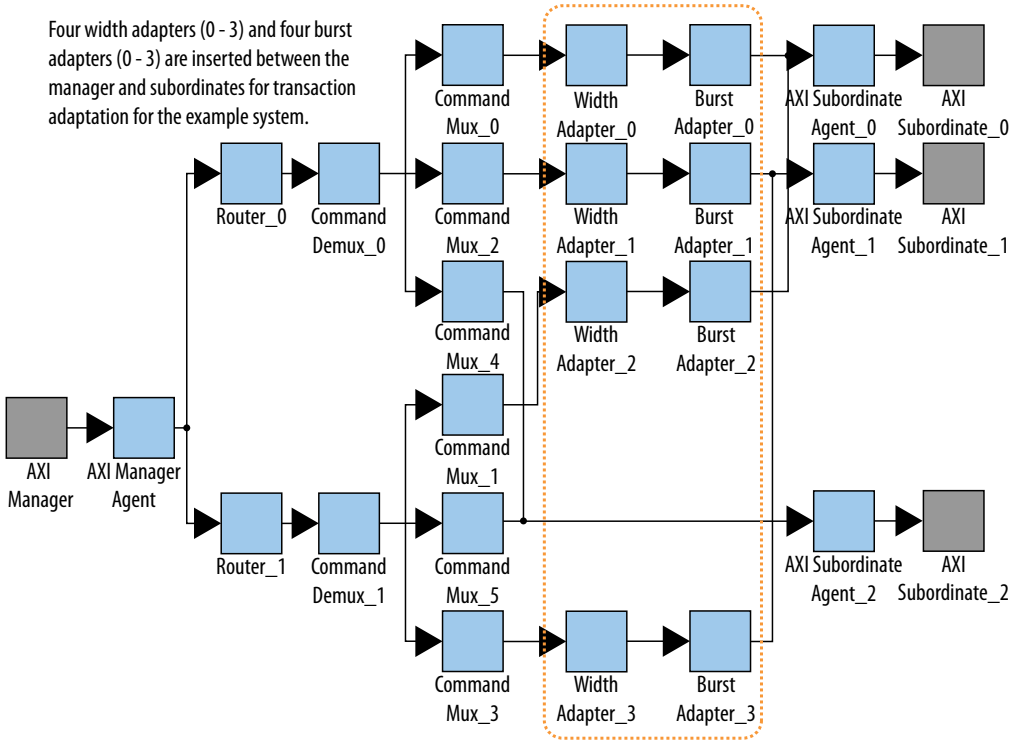
With an AXI bridge, you can influence the placement of resource-intensive components, such as the width and burst adapters. Depending on its use, an AXI bridge may reduce throughput and concurrency, in return for higher f_{MAX} and less logic.

You can use an AXI Bridge Intel FPGA IP to group different parts of your Platform Designer system. Other parts of the system can then connect to the bridge interface instead of to multiple separate manager or subordinate interfaces. You can also use an AXI bridge to export AXI interfaces from Platform Designer systems.

The AXI bridge also supports ACE-Lite interface signals when enabled on the manager or subordinate side. Platform Designer treats these ACE-Lite interface signals as pass through signals.

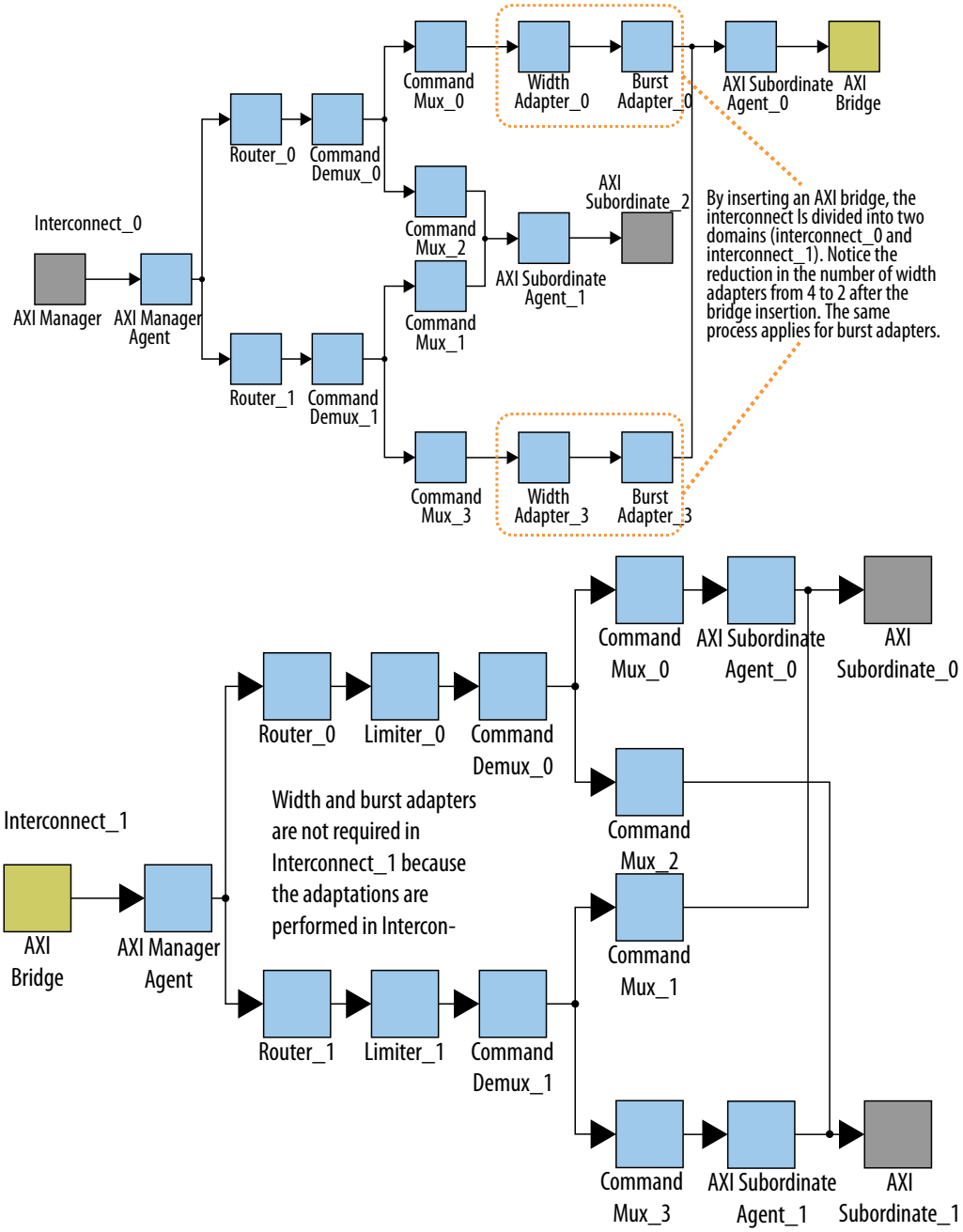
The following figure shows a system with a single AXI manager and three AXI subordinates. It also has various interconnect components, such as routers, demultiplexers, and multiplexers. Two of the subordinates have a narrower data width than the manager; 16-bit subordinates versus a 32-bit manager.

Figure 255. AXI System Without a Bridge



In this system, Platform Designer interconnect creates four width adapters and four burst adapters to access the two subordinates. You can improve resource usage by adding an AXI bridge. Then, Platform Designer needs to add only two width adapters and two burst adapters; one pair for the read channels, and another pair for the write channel.

Figure 256. Width and Burst Adapters Added to System With a Bridge



The figure shows the same system with an AXI bridge component, and the decrease in the number of width and burst adapters. Platform Designer creates only two width adapters and two burst adapters, as compared to the four width adapters and four burst adapters in the previous figure. Even though this system includes more components, the overall system performance improves because there are fewer resource-intensive width and burst adapters.

6.1.6.1. AXI Bridge Signal Types

Based on parameter selections that you make for the AXI Bridge component, Platform Designer instantiates either the AMBA 3 AXI or AMBA 4 AXI manager and subordinate interfaces into the component.

Note: In AMBA 3 AXI, `aw/aruser` accommodates sideband signal usage by hard processor systems (HPS).

Table 124. Sets of Signals for the AXI Bridge Based on the Protocol

Signal Name	AMBA 3 AXI	AMBA 4 AXI
awid / arid	yes	yes
awaddr / araddr	yes	yes
awlen / arlen	yes (4-bit)	yes (8-bit)
awsize / arsize	yes	yes
awburst / arburst	yes	yes
awlock / arlock	yes	yes (1-bit optional)
awcache / arcache	yes (2-bit)	yes (optional)
awprot / arprot	yes	yes
awuser / aruser	yes	yes
awvalid / arvalid	yes	yes
awready / arready	yes	yes
awqos / arqos	no	yes
awregion / arregion	no	yes
wid	yes	no (optional)
wdata / rdata	yes	yes
wstrb	yes	yes
wlast / rvalid	yes	yes
wvalid / rlast	yes	yes
wready / rready	yes	yes
wuser / ruser	no	yes
bid / rid	yes	yes
bresp / rresp	yes	yes (optional)
bvalid	yes	yes
bready	yes	yes

6.1.6.2. AXI Bridge Parameters

In the parameter editor, you can customize the parameters for the AXI bridge according to the requirements of your design.

Figure 257. AXI Bridge Parameter Editor

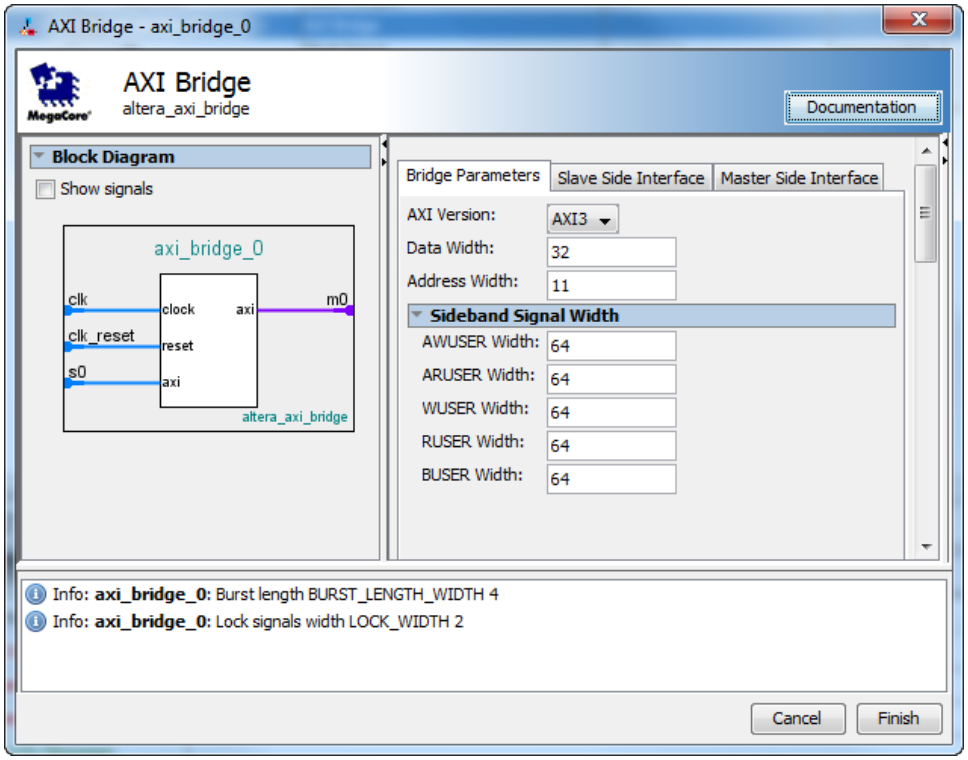


Table 125. AXI Bridge Parameters

Parameter	Type	Range	Description
AXI Version	string	AMBA 3 AXI or AMBA 4 AXI	Specifies the AXI version and signals that Platform Designer generates for the subordinate and manager interfaces of the bridge.
Data Width	integer	8:1024	Controls the width of the data for the manager and subordinate interfaces.
Address Width	integer	1-64 bits	Controls the width of the address for the manager and subordinate interfaces.
AWUSER Width	integer	1-64 bits	Controls the width of the write address channel sideband signals of the manager and subordinate interfaces.
ARUSER Width	integer	1-64 bits	Controls the width of the read address channel sideband signals of the manager and subordinate interfaces.
WUSER Width	integer	1-64 bits	Controls the width of the write data channel sideband signals of the manager and subordinate interfaces.
RUSER Width	integer	1-16 bits	Controls the width of the read data channel sideband signals of the manager and subordinate interfaces.
BUSER Width	integer	1-16 bits	Controls the width of the write response channel sideband signals of the manager and subordinate interfaces.

6.1.6.3. AXI Bridge Subordinate and Manager Interface Parameters

Table 126. AXI Bridge Subordinate and Manager Interface Parameters

Parameter	Description
ID Width	Controls the width of the thread ID of the manager and subordinate interfaces.
Write/Read/Combined Acceptance Capability	Controls the depth of the FIFO that Platform Designer needs in the interconnect agents based on the maximum pending commands that the subordinate interface accepts.
Write/Read/Combined Issuing Capability	Controls the depth of the FIFO that Platform Designer needs in the interconnect agents based on the maximum pending commands that the manager interface issues. Issuing capability must follow acceptance capability to avoid unnecessary creation of FIFOs in the bridge.

Note: Maximum acceptance/issuing capability is a model-only parameter and does not influence the bridge HDL. The bridge does not backpressure when this limit is reached. Downstream components or the interconnect must apply backpressure.

6.1.7. AXI Timeout Bridge Intel FPGA IP

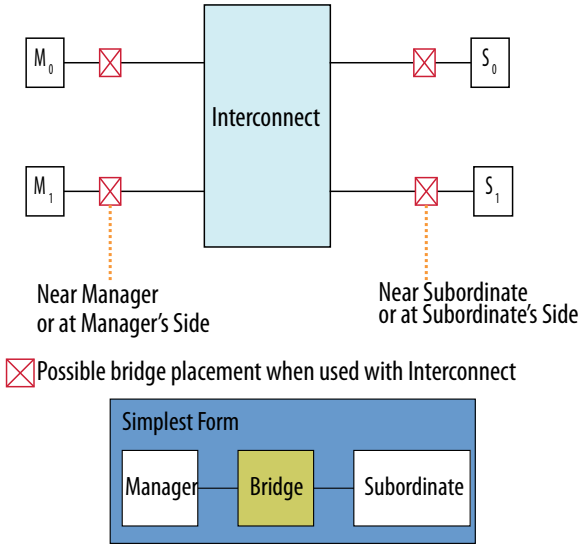
The AXI Timeout Bridge Intel FPGA IP allows your system to recover when it freezes, and facilitates debugging. You can place an AXI Timeout Bridge between a single manager and a single subordinate if you know that the subordinate may time out and cause your system to freeze. If a subordinate does not accept a command or respond to a command it accepted, its manager can wait indefinitely.

For a domain with multiple managers and subordinates, placement of an AXI Timeout Bridge in your design may be beneficial in the following scenarios:

- To recover from a freeze, place the bridge near the subordinate. If the manager attempts to communicate with a subordinate that freezes, the AXI Timeout Bridge frees the manager by generating error responses. The manager is then able to communicate with another subordinate.
- When debugging your system, place the AXI Timeout Bridge near the manager. This placement enables you to identify the origin of the burst, and to obtain the full address from the manager. Additionally, placing an AXI Timeout Bridge near the manager enables you to identify the target subordinate for the burst.

  Note: If you place the bridge at the subordinate's side and you have multiple subordinates connected to the same manager, you do not get the full address.

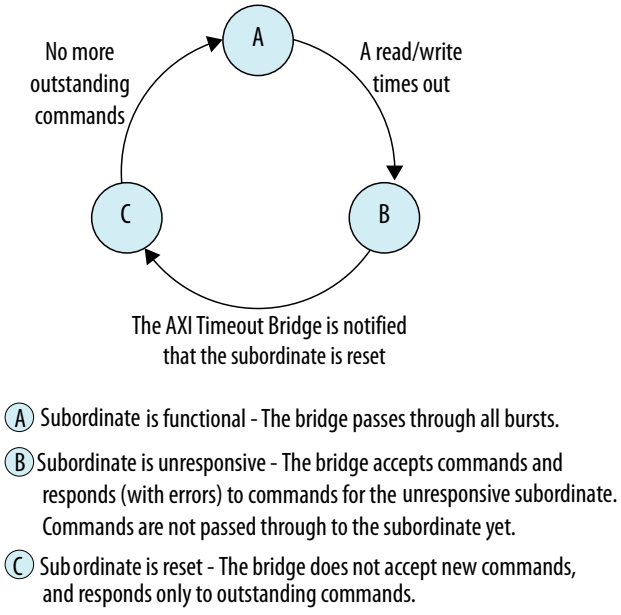
Figure 258. AXI Timeout Bridge Placement



6.1.7.1. AXI Timeout Bridge Stages

A timeout occurs when the internal timer in the bridge exceeds the specified number of cycles within which a burst must complete from start to end.

Figure 259. AXI Timeout Bridge Stages



- When a timeout occurs, the AXI Timeout Bridge asserts an interrupt and reports the burst that caused the timeout to the Configuration and Status Register (CSR).
- The bridge then generates error responses back to the manager on behalf of the unresponsive subordinate. This stage frees the manager and certifies the unresponsive subordinate as dysfunctional.
- The AXI Timeout Bridge accepts subsequent write addresses, write data, and read addresses to the dysfunctional subordinate. The bridge does not accept outstanding write responses, and read data from the dysfunctional subordinate is not passed through to the manager.
- The `awvalid`, `wvalid`, `bready`, `arvalid`, and `rready` ports are held low at the manager interface of the bridge.

Note: After a timeout, `awvalid`, `wvalid`, and `arvalid` may be dropped before they are accepted by `awready` at the manager interface. While the behavior violates the AXI specification, it occurs only on an interface connected to the subordinate which has been certified dysfunctional by the AXI Timeout Bridge.

Write channel refers to the AXI write address, data and response channels. Similarly, read channel refers to the AXI read address and data channels. AXI write and read channels are independent of each other. However, when a timeout occurs on either channel, the bridge generates error responses on both channels.

Table 127. Burst Start and End Definitions for the AXI Timeout Bridge

Channel	Start	End
Write	When an address is issued. First cycle of `awvalid`, even if data of the same burst is issued before the address (first cycle of `wvalid`).	When the response is issued. First cycle of `bvalid`.
Read	When an address is issued. First cycle of `arvalid`.	When the last data is issued. First cycle of `rvalid` and `rlast`.

The AXI Timeout Bridge has four required interfaces: manager, subordinate, Configuration and Status Register (CSR) (AMBA 3 AXI-Lite), and Interrupt. Platform Designer allows the AXI Timeout Bridge to connect to any AMBA 3 AXI, AMBA 4 AXI, or Avalon host or agent interface. Avalon hosts must utilize the bridge's interrupt output to detect a timeout.

The bridge subordinate interface accepts write addresses, write data, and read addresses, and then generates the SLVERR response at the write response and read data channels. Do not use `buser`, `rdata`, and `ruser` at this stage of processing.

To resume normal operation, the dysfunctional subordinate must be reset and the bridge notified of the change in status via the CSR. Once the CSR notifies the bridge that the subordinate is ready, the bridge does not accept new commands until all outstanding bursts are responded to with an error response.

The CSR has a 4-bit address width and a 32-bit data width. The CSR reports status and address information when the bridge asserts an interrupt.

Table 128. CSR Interrupt Status Information for the AXI Timeout Bridge

Address	Attribute	Name	Description	
0x0	write-only	Subordinate is reset	Write a 1 to this address to notify the AXI timeout bridge that the subordinate is reset and is ready. This also clears the interrupt.	
0x4	read-only	Timed out operation	Legacy behavior=1	The operation of the burst that causes the timeout: • 01: write operation • 00: read operation
			Legacy behavior=0	CSR Values: • 00: No timeout occurs • 10: Read operation causes timeout • 11: Write operation causes timeout
0x8 through 0xF	read-only	Timed out address	The address of the burst that causes the timeout. Note: If you use an ADDRESS_WIDTH of more than 32 bits, two CSR reads (from addresses 0x8 and 0xc) are required to get the complete address, given that the data width of the CSR interface is only 32 bits wide.	

6.1.7.2. AXI Timeout Bridge Parameters

Table 129. AXI Timeout Bridge Parameters

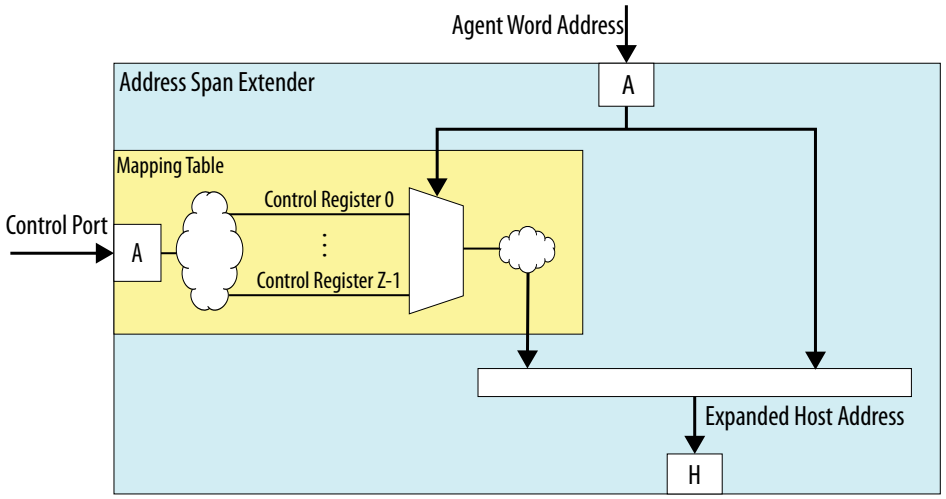
Parameter	Description
ID width	The width of awid, bid, arid, or rid.
Address width	The width of awaddr or araddr.
Data width	The width of wdata or rdata.
User width	The width of awuser, wuser, buser, aruser, or ruser.
Maximum number of outstanding writes	The expected maximum number of outstanding writes.
Maximum number of outstanding reads	The expected maximum number of outstanding reads.
Maximum number of cycles	The number of cycles within which a burst must complete.
Use synchronous resets	When this option is off (default), the IP allows asynchronous resets. When this option is on, the IP uses internal reset synchronization and allows no asynchronous resets.
Use Legacy Behavior	When this option is off, the IP uses the new behavior of the CSR and allows asynchronous resets. When this option is on, the IP uses the legacy (previous) behavior of the CSR, and you cannot access the CSR except during a timeout. For a new instance of the IP, the default **Use Legacy Behavior** is OFF. If you update the old version of the IP, the default Legacy Behavior is ON as previous behavior.

6.1.8. Address Span Extender Intel FPGA IP

The Address Span Extender Intel FPGA IP allows memory-mapped host interfaces to access a larger or smaller address map than the width of their address signals allows. The Address Span Extender IP splits the addressable space into multiple separate windows, so that the host can access the appropriate part of the memory through the window.

The Address Span Extender does not limit host and agent widths to a 32-bit and 64-bit configuration. You can use the Address Span Extender with 1-64 bit address windows.

Figure 260. Address Span Extender Intel FPGA IP



If a processor can address only 2 GB of an address span, and your system contains 4 GB of memory, the Address Span Extender can provide two, 2 GB windows in the 4 GB memory address space. This issue sometimes occurs with Intel SoC devices.

For example, an HPS subsystem in an SoC device can address only 1 GB of an address span within the FPGA, using the HPS-to-FPGA bridge. The Address Span Extender enables the SoC device to address all the address space in the FPGA using multiple 1 GB windows.

6.1.8.1. CTRL Register Layout

The control registers consist of one 64-bit register for each window, where you specify the window's base address. For example, if CTRL_BASE is the base address of the control register, and address span extender contains two windows (0 and 1), then window 0's control register starts at CTRL_BASE, and window 1's control register starts at CTRL_BASE + 8 (using byte addresses).

6.1.8.2. Address Span Extender Parameters

Table 130. Address Span Extender Parameters

Parameter	Description
Datapath Width	Width of write data and read data signals.
Expanded Master Byte Address Width	Width of the host byte address port. That is, the address span size of all the downstream agents that attach to the address span extender.
Slave Word Address Width	Width of the agent word address port. That is, the address span size of the downstream agents that the upstream host accesses.
Burstcount Width	Burst count port width of the downstream agent and the upstream host that attach to the address span extender.

continued...

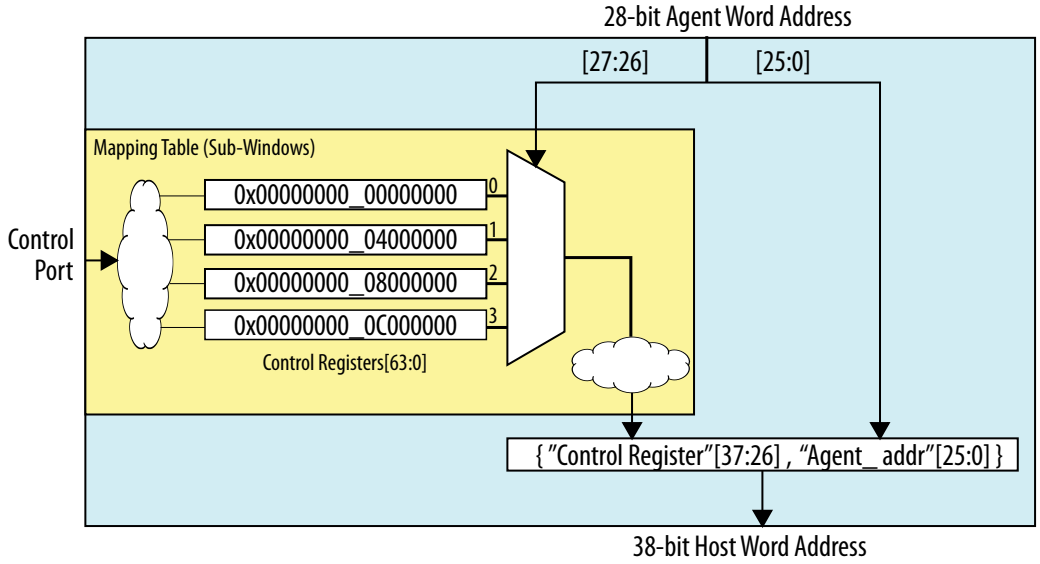
Parameter	Description
Number of sub-windows	The agent port can represent one or more windows in the host address span. You can subdivide the agent address span into *N* equal spans in *N* sub-windows. A remapping register in the CSR agent represents each sub-window, and configures the base address that each sub-window remaps to. The address span extender replaces the upper bits of the address with the stored index value in the remapping register before the host initiates a transaction.
Enable Slave Control Port	Dictates run-time control over the sub-window indexes. If you can define static re-mappings that do not need any change, you do not need to enable this CSR agent.
Maximum Pending Reads	Sets the bridge agent's `maximumPendingReadTransactions` property. In certain system configurations, you must increase this value to improve performance. This value usually aligns with the properties of the downstream agents that you attach to this bridge.

6.1.8.3. Calculating the Address Span Extender Agent Address

The diagram describes how Platform Designer calculates the agent address. In this example, the address span extender is configured with a 28-bit address space for agents. The upper 2 bits `[27:26]` are used to select the control registers.

The lower 26 bits (`[25:0]`) originate from the address span extender's data port, and are the offset into a particular window.

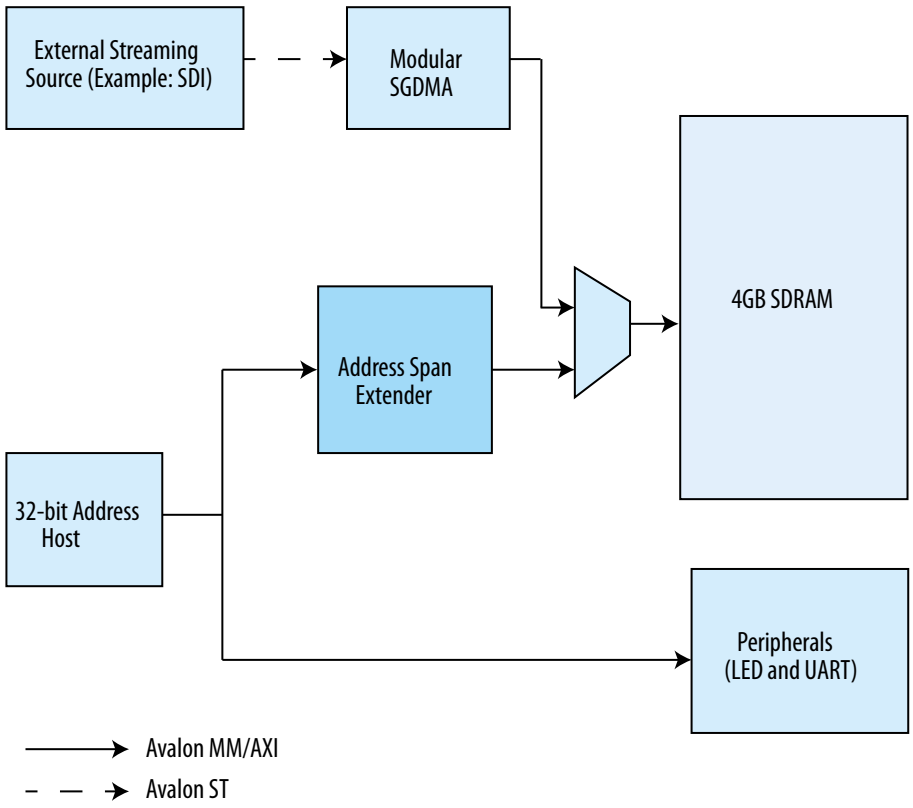
Figure 261. Address Span Extender



6.1.8.4. Using the Address Span Extender

This example shows when and how to use address span extender component in your Platform Designer design.

Figure 262. Block Diagram with Address Span Extender



In the above design, a 32-bit host shares 4 GB SDRAM with an external streaming interface. The host has the path to access streaming data from the SDRAM DDR memory. However, if you connect the whole 32-bit address bus of the host to the SDRAM DDR memory, you cannot connect the host to peripherals such as LED or UART. To avoid this situation, you can implement the address span extender between the host and DDR memory. The address span extender allows the host to access the SDRAM DDR memory and the peripherals at the same time.

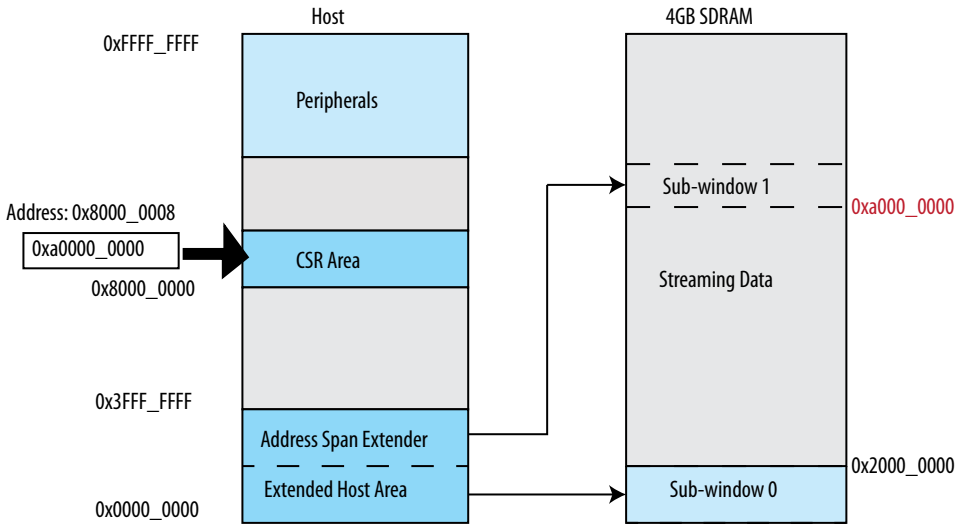
To implement address span extender for the above example, you can divide the address window of the address span extender into two sub-windows of 512 MB each. The sub-window 0 is for the host program area. You can dynamically map the sub-window 1 to any area other than the program area.

You can change the offset of the address window by setting the base address of sub-window 1 to the control register of the address span extender. However, you must make sure that the sub-window address span masks the base address. You can choose any arbitrary base address. If you set the value 0xa000_0000 to the control register, Platform Designer maps the sub-window 1 to 0xa000_0000.

Table 131. CSR Mapping Table

Address	Data
0x8000_0000	0x0000_0000
0x8000_0008	0xa000_0000

Figure 263. Memory mapping for Address Span Extender

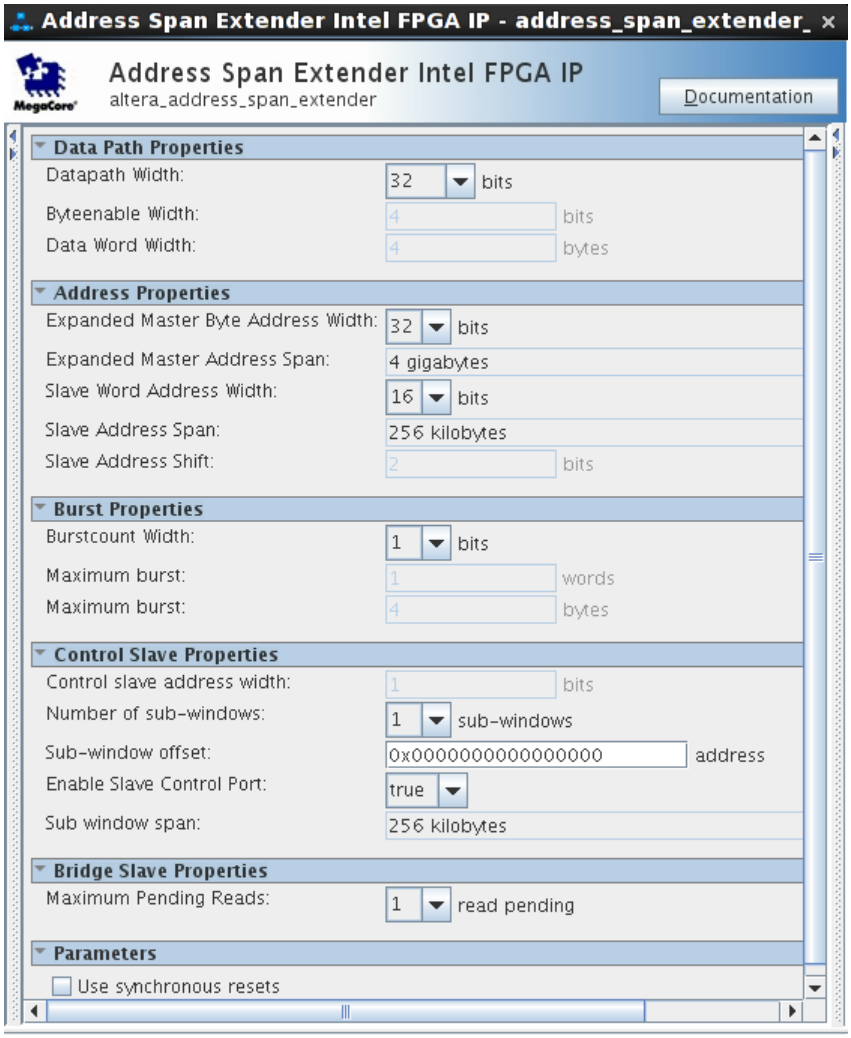


The table below indicates the Platform Designer parameter settings for this address span extender example.

Table 132. Parameter Settings for the Address Span Extender Example

Parameter	Value	Description
Datapath Width	32 bits	The CPU has 32-bits data width and the SDRAM DDR memory has 512-bits data width. Since the transaction between the host and SDRAM DDR memory is minimal, set the datapath width to align with the upstream host.
Expanded Master Byte Address	32 bits	The address span extender has a 4 GB address span.
Slave Word Address Width	18 bits	There are two 512 MB sub-windows in reserve for the host. The number of bytes over the data word width in the **Datapath Properties** (4 bytes for this example) accounts for the agent address.
Burstcount Width	4 bits	The address span extender must handle up to 8 words burst in this example.
Number of sub-windows	2	Address window of the address span extender has two sub-windows of 512 MB each.
Enable Slave Control Port	true	The address span extender component must have control to change the base address of the sub-window.
Maximum Pending Reads	4	This number is the same as SDRAM DDR memory burst count.

Figure 264. Address Span Extender Parameter Editor



Note: You can view the address span extender connections in the **System View** tab. The windowed agent port and control port connect to the host. The expanded host port connects to the SDRAM DDR memory.

6.1.8.5. Alternate Options for the Address Span Extender

You can set parameters for the address span extender with an initial fixed address value. Enter an address for the **Reset Default for Master Window** option, and select **True** for the **Disable Slave Control Port** option. This allows the address span extender to function as a fixed, non-programmable component.

Each sub-window is equal in size and stacks sequentially in the windowed agent interface's address space. To control the fixed address bits of a particular sub-window, you can write to the sub-window's register in the register control agent interface. Platform Designer structures the logic so that Platform Designer can optimize and remove bits that are not needed.

If **Burstcount Width** is greater than 1, Platform Designer processes the read burst in a single cycle, and assumes all `byteenable` signals are asserted on every cycle.

6.1.8.6. Nios II Support

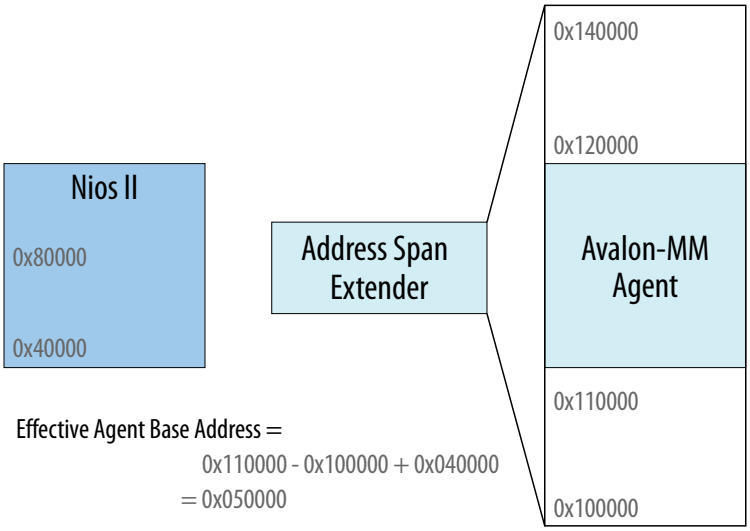
If the address span extender window is fixed, for example, the **Disable Slave Control Port** option is turned on, then the address span extender performs as a bridge. Components on the agent side of the address span extender that are within the window are visible to the Nios II processor. Components partially within a window appear to the Nios II processor as if they have a reduced span. For example, a memory partially within a window appears as having a smaller size.

You can also use the address span extender to provide a window for the Nios II processor, so that the HPS memory map is visible to the Nios II processor. This technique allows the Nios II processor to communicate with HPS peripherals.

In the example, a Nios II processor has an address span extender from address `0x40000` to `0x80000`. There is a window within the address span extender starting at `0x100000`. Within the address span extender's address space there is an agent at base address `0x1100000`. The agent appears to the Nios II processor as being at address:

```
0x110000 - 0x100000 + 0x40000 = 0x050000
```

Figure 265. Nios II Support and the Address Span Extender



The address span extender window is dynamic. For example, when the **Disable Slave Control Port** option is turned off, the Nios II processor is unable to see components on the agent side of the address span extender.

6.2. Error Response Slave Intel FPGA IP

The Error Response Slave IP provides a predictable error response service for host interfaces that attempt to access an undefined memory region.

The Error Response Slave is an AMBA 3 AXI component, and appears in the Platform Designer IP Catalog under **Platform Designer Interconnect**.

To comply with the AXI protocol, the interconnect logic must return the DECERR error response in cases where the interconnect cannot decode subordinate access. Therefore, an AXI system with address space not fully decoded to subordinate interfaces requires the Error Response Slave.(25)

The Error Response Slave behaves like any other component in the system, and connects to other components via translation and adaptation interconnect logic. Connecting an Error Response Slave to managers or hosts of different data widths, including Avalon hosts or AXI-Lite managers, can increase resource usage.

An Error Response Slave can connect to clock, reset, and IRQ signals as well as AMBA 3 AXI and AMBA 4 AXI manager interfaces without instantiating a bridge. When you connect an Error Response Slave to a manager or host, the Error Response Slave accepts cycles sent from the manager or host, and returns the DECERR error response. On the AXI interface, the Error Response Slave supports only a read and write acceptance of capability 1, and does not support write data interleaving. The Error Response Slave can return responses when simultaneously targeted by a read and write cycle, because its read and write channels are independent.

An optional Avalon interface on the Error Response Slave provides information in a set of CSR registers. CSR registers log the required information when returning an error response.

- To set the Error Response Slave as the default subordinate or agent for a manager or host interface in your system, connect the subordinate to the manager or connect the agent to the host in your Platform Designer system.
- A system can contain more than one Error Response Slave.
- As a best practice, instantiate separate Error Response Slave components for each AXI manager in your system.

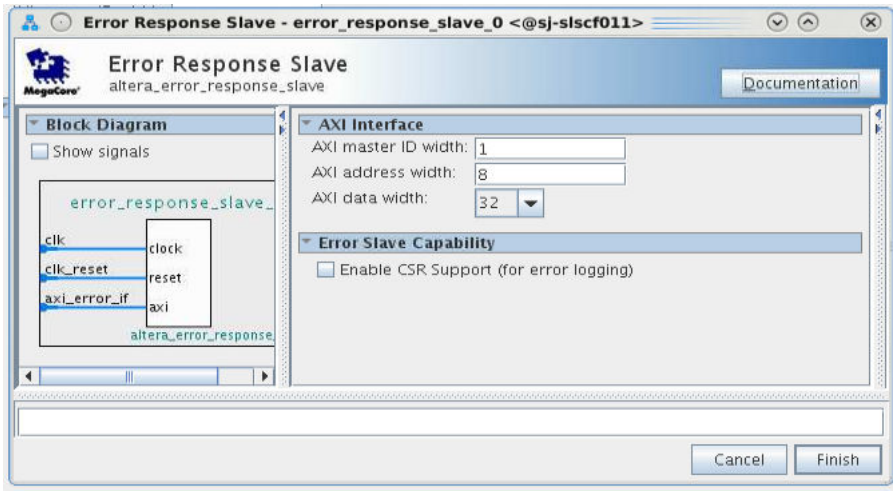
Related Information

- AMBA 3 AXI Protocol Specification Support (version 1.0) on page 332
- Designating a Default Agent on page 396

(25) This document now refers to the AXI "manager" and "subordinate" to replace the former terms. Refer to the latest *AMBA AXI and ACE Protocol Specification*.

6.2.1. Error Response Slave Parameters

Figure 266. Error Response Slave Parameter Editor



If you turn on **Enable CSR Support (for error logging)** more parameters become available.

Figure 267. Error Response Slave Parameter Editor with Enabled CSR Support

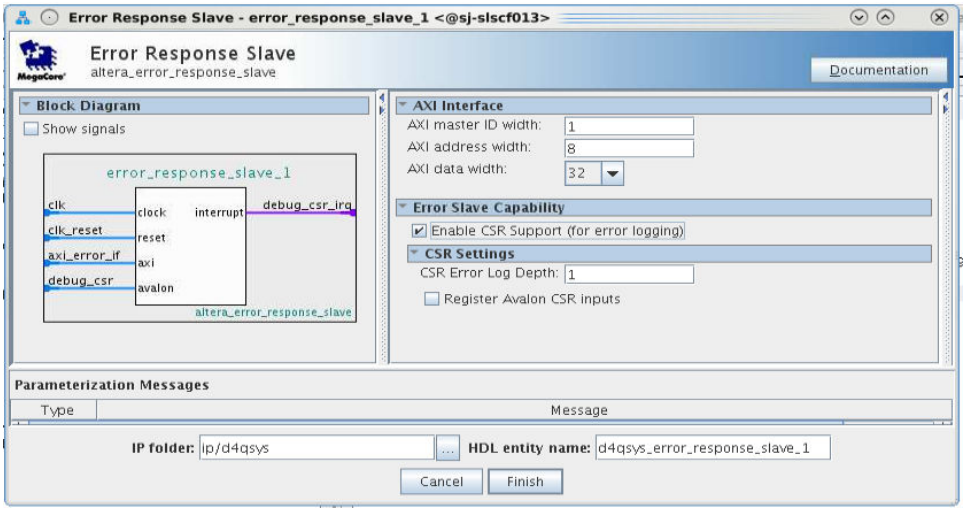


Table 133. Error Response Slave Parameters

Parameter	Value	Description
AXI master ID width	1-8 bits	Specifies the manager ID width for error logging.
AXI address width	8-64 bits	Specifies the address width for error logging. This value also affects the overall address width of the system, and should not exceed the maximum address width required in the system.

continued...

Parameter	Value	Description
AXI data width	32, 64, or 128 bits	Specifies the data width for error logging.
Enable CSR Support (for error logging)	On / Off	When turned on, instantiates an Avalon CSR interface for error logging.
CSR Error Log Depth	1-16 bits	Depth of the transaction log, for example, the number of transactions the CSR logs for cycles with errors.
Register Avalon CSR inputs	On / Off	When turned on, controls debug access to the CSR interface.

6.2.2. Error Response Slave CSR Registers

The Error Response Slave with enabled CSR support provides a service to handle access violations. This service uses CSR registers for status and logging purposes.

The sequence of actions in the access violation service is equivalent for read and write access violations, but the CSR status bits and log registers are different.

6.2.2.1. Error Response Slave Access Violation Service

When an access violation occurs, and the CSR port is enabled:

1. The Error Response Slave generates an interrupt:
   - For a read access violation, the Error Response Slave sets the `Read Access Violation Interrupt register` bit in the `Interrupt Status` register.
   - For a write access violation, the Error Response Slave sets the `Write Access Violation Interrupt register` bit in the `Interrupt Status` register.
2. The Error Response Slave transfers transaction information to the access violation log FIFO. The amount of information that the FIFO can handle is given by the **Error Log Depth** parameter.

   You define the **Error Log Depth** in the **Parameter Editor**, when you enable CSR Support.
3. Software reads entries of the access violation log FIFO until the corresponding `cycle log valid` bit is cleared, and then exits the service routine.
   - The `Read cycle log valid` bit is in the `Read Access Violation Log` CSR Registers.
   - The `Write cycle log valid` bit is in the `Write Access Violation Log` CSR Registers.
4. The Error Response Slave clears the interrupt bit when there are no access violations to report.

Some special cases are:

- If any error occurs when the FIFO is full, the Error Response Slave sets the corresponding `Access Violation Interrupt Overflow register` bit (bits 2 and 3 of the Status Register for write and read access violations, respectively). Setting this bit means that not all error entries were written to the access violation log.
- After Software reads an entry in the Access Violation log, the Error Response Slave can write a new entry to the log.
- Software can specify the number of entries to read before determining that the access violation service is taking too long to complete, and exit the routine.

6.2.2.2. CSR Interrupt Status Registers

Table 134. CSR Interrupt Status Registers

For CSR register maps: `Address = Memory Address Base + Offset`.

Offset	Bits	Attribute	Default	Description
0x00	31:4	Reserved.		
	3	RW1C	0	`Read Access Violation Interrupt Overflow register` Asserted when a read access causes the Interconnect to return a DECERR response, and the buffer log depth is full. Indicates that there is a logging error lost due to an exceeded buffer log depth. Cleared by setting the bit to 1.
	2	RW1C	0	`Write Access Violation Interrupt Overflow register` Asserted when a write access causes the Interconnect to return a DECERR response, and the buffer log depth is full. Indicates that there is a logging error lost due to an exceeded buffer log depth. Cleared by setting the bit to 1.
	1	RW1C	0	`Read Access Violation Interrupt register` Asserted when a read access causes the Interconnect to return a DECERR response. Cleared by setting the bit to 1. *Note:* Access violation are logged until the bit is cleared.
	0	RW1C	0	`Write Access Violation Interrupt register` Asserted when a write access causes the Interconnect to return a DECERR response. Cleared by setting the bit to 1. *Note:* Access violation are logged until the bit is cleared.

6.2.2.3. CSR Read Access Violation Log Registers

The CSR read access violation log settings are valid only when an associated read interrupt register is set. Read this set of registers until the validity bit is cleared.

Table 135. CSR Read Access Violation Log Registers

Offset	Bits	Attribute	Default	Description
0x100	31:13	Reserved.		
	12:11	R0	0	`Offending Read cycle burst type`: Specifies the burst type of the initiating cycle that causes the access violation.
	10:7	R0	0	`Offending Read cycle burst length`: Specifies the burst length of the initiating cycle that causes the access violation.
	6:4	R0	0	`Offending Read cycle burst size`: Specifies the burst size of the initiating cycle that causes the access violation.

continued...

Offset	Bits	Attribute	Default	Description
	3:1	R0	0	Offending Read cycle PROT: Specifies the PROT of the initiating cycle that causes the access violation.
	0	R0	0	Read cycle log valid: Specifies the validity of the read access violation log. This bit is cleared when the interrupt register is cleared.
0x104	31:0	R0	0	Offending read cycle ID: Manager ID for the cycle that causes the access violation.
0x108	31:0	R0	0	Offending read cycle target address: Target address for the cycle that causes the access violation (lower 32-bit).
0x10C	31:0	R0	0	Offending read cycle target address: Target address for the cycle that causes the access violation (upper 32-bit). Valid only if widest address in system is larger than 32 bits. *Note:* When this register is read, the current read access violation log is recovered from FIFO.

6.2.2.4. CSR Write Access Violation Log Registers

The CSR write access violation log settings are valid only when an associated write interrupt register is set. Read this set of registers until the validity bit is cleared.

Table 136. CSR Write Access Violation Log

Offset	Bits	Attribute	Default	Description
0x190	31:13	Reserved.		
	12:11	R0	0	Offending write cycle burst type: Specifies the burst type of the initiating cycle that causes the access violation.
	10:7	R0	0	Offending write cycle burst length: Specifies the burst length of the initiating cycle that causes the access violation.
	6:4	R0	0	Offending write cycle burst size: Specifies the burst size of the initiating cycle that causes the access violation.
	3:1	R0	0	Offending write cycle PROT: Specifies the PROT of the initiating cycle that causes the access violation.
	0	R0	0	Write cycle log valid: Specifies whether the log for the transaction is valid. This bit is cleared when the interrupt register is cleared.
0x194	31:0	R0	0	Offending write cycle ID: Manager ID for the cycle that causes the access violation.
0x198	31:0	R0	0	Offending write cycle target address: Write target address for the cycle that causes the access violation (lower 32-bit).
0x19C	31:0	R0	0	Offending write cycle target address: Write target address for the cycle that causes the access violation (upper 32-bit). Valid only if widest address in system is larger than 32 bits.
0x1A0	31:0	R0	0	Offending write cycle first write data: First 32 bits of the write data for the write cycle that causes the access violation. *Note:* When this register is read, the current write access violation log is recovered from FIFO, when the data width is 32 bits.

continued...

Offset	Bits	Attribute	Default	Description
0x1A4	31:0	R0	0	Offending write cycle first write data: Bits [63:32] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 32 bits.
0x1A8	31:0	R0	0	Offending write cycle first write data: Bits [95:64] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 64 bits.
0x1AC	31:0	R0	0	Offending write cycle first write data: The first bits [127:96] of the write data for the write cycle that causes the access violation. Valid only if the data width is greater than 64 bits. *Note:* When this register is read, the current write access violation log is recovered from FIFO.

6.2.3. Designating a Default Agent

You can designate any agent in your Platform Designer system as the error response default agent. The default agent you designate provides an error response service for hosts that attempt access to an undefined memory region.

1. In your Platform Designer system, in the **System View** tab, right-click the header and turn on **Show Default Slave Column**.
2. Select the agent that you want to designate as the default agent, and then click the checkbox for the agent in the **Default Slave** column.
3. In the **System View** tab, in the **Connections** column, connect the designated default agent to one or more hosts.

   Note: If you do not specify a value for the **Default Slave** option, and the **Automate default slave insertion** option is off, Platform Designer automatically assigns the AXI subordinate or Avalon agent in the system. Platform Designer automatically assigns the AXI subordinate or Avalon agent that has largest address span within the memory map for the issuing AXI manager or Avalon host. In the case of multiple, large AXI subordinates or Avalon agents that have the same address span, Platform Designer selects the AXI subordinate or Avalon agent at the lowest base offset.

Related Information

Specifying a Default Avalon Agent or AXI Subordinate on page 80

6.3. Tri-State Components

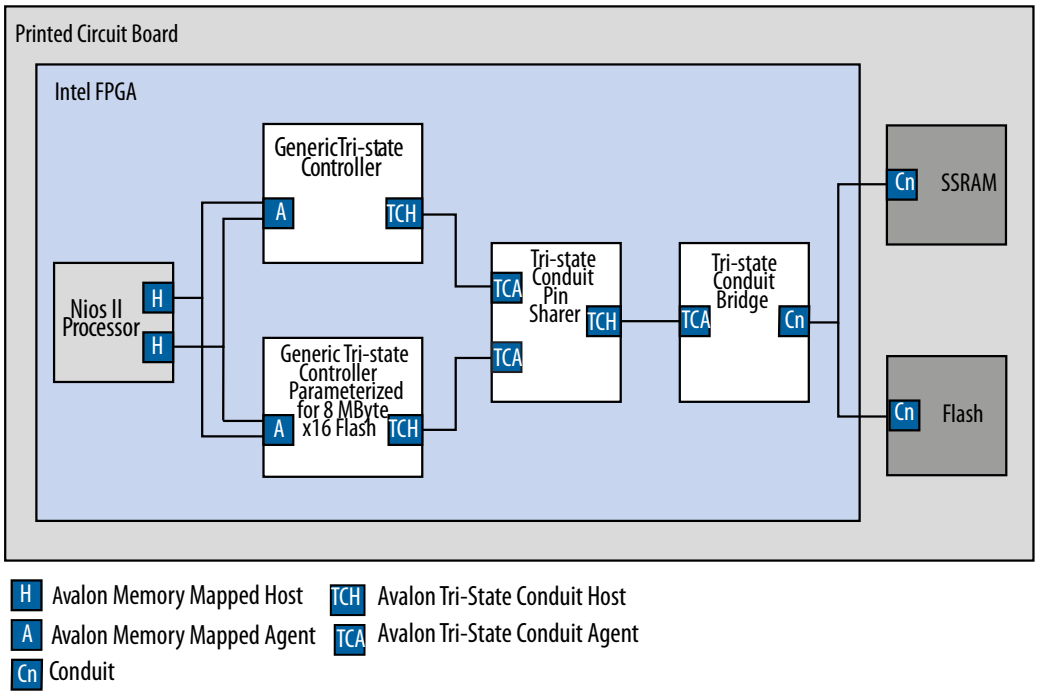
The tri-state interface type allows you to design Platform Designer subsystems that connect to tri-state devices on your PCB. You can use tri-state components to implement pin sharing, convert between unidirectional and bidirectional signals, and create tri-state controllers for devices whose interfaces can be described using the tri-state signal types.

Example 27. Tri-State Conduit System to Control Off-Chip SRAM and Flash Devices

In this example, there are two generic Tri-State Conduit Controllers. The first is customized to control a flash memory. The second is customized to control an off-chip SSRAM. The Tri-State Conduit Pin Sharer multiplexes between these two controllers, and the Tri-State Conduit Bridge converts between an on-chip encoding of tri-state

signals and true bidirectional signals. By default, the Tri-State Conduit Pin Sharer and Tri-State Conduit Bridge present byte addresses. Typically, each address location contains more than one byte of data.

Figure 268. Tri-State Conduit System to Control Off-Chip SRAM and Flash Devices

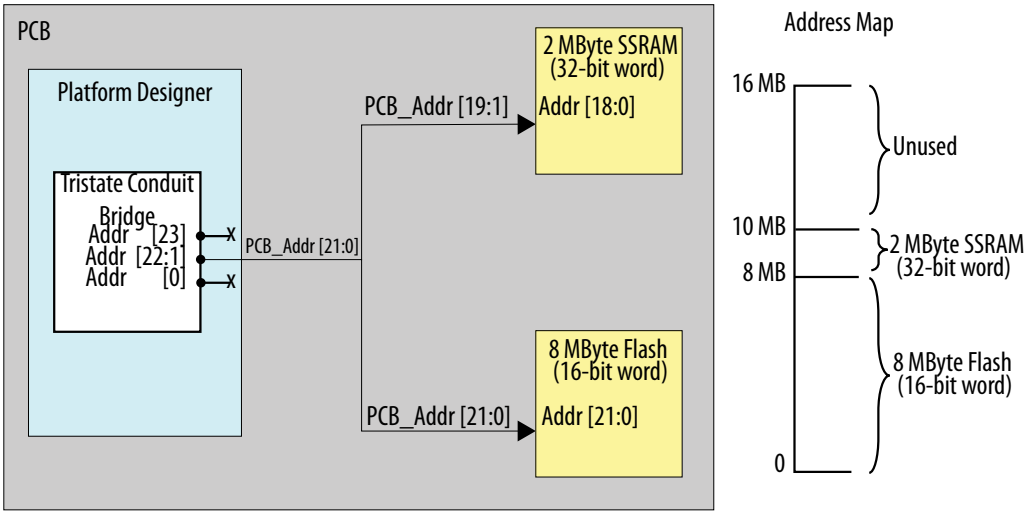


Address Connections from Platform Designer System to PCB

The flash device operates on 16-bit words and must ignore the least-significant bit of the Avalon memory mapped address. The figure shows `addr[0]`as not connected. The SSRAM memory operates on 32-bit words and must ignore the two low-order memory bits. Because neither device requires a byte address, `addr[0]` is not routed on the PCB.

The flash device responds to address range 0 MB to 8 MB-1. The SSRAM responds to address range 8 MB to 10 MB-1. The PCB schematic for the PCB connects `addr[21:0]` to `addr[18:0]` of the SSRAM device because the SSRAM responds to 32-bit word address. The 8 MB flash device accesses 16-bit words; consequently, the schematic does not connect `addr[0]`. The `chipselect` signals select between the two devices.

Figure 269. Address Connections from Platform Designer System to PCB



Note: If you create a custom tri-state conduit host with word aligned addresses, the Tri-state Conduit Pin Sharer does not change or align the address signals.

Figure 270. Tri-State Conduit System in Platform Designer

Related Information

Avalon Interface Specifications

6.3.1. Generic Tri-State Controller Intel FPGA IP

The Generic Tri-State Controller Intel FPGA IP provides a template for a controller. You can customize the Generic Tri-State Controller with various parameters to reflect the behavior of an off-chip device. The following types of parameters are available for the Generic Tri-State Controller:

- Width of the address and data signals
- Read and write wait times
- Bus turnaround time
- Data hold time

Note: In calculating delays, the Generic Tri-State Controller chooses the larger of the bus turnaround time and read latency. Turnaround time is measured from the time that a command is accepted, not from the time that the previous read returned data.

The Generic Tri-State Controller includes the following interfaces:

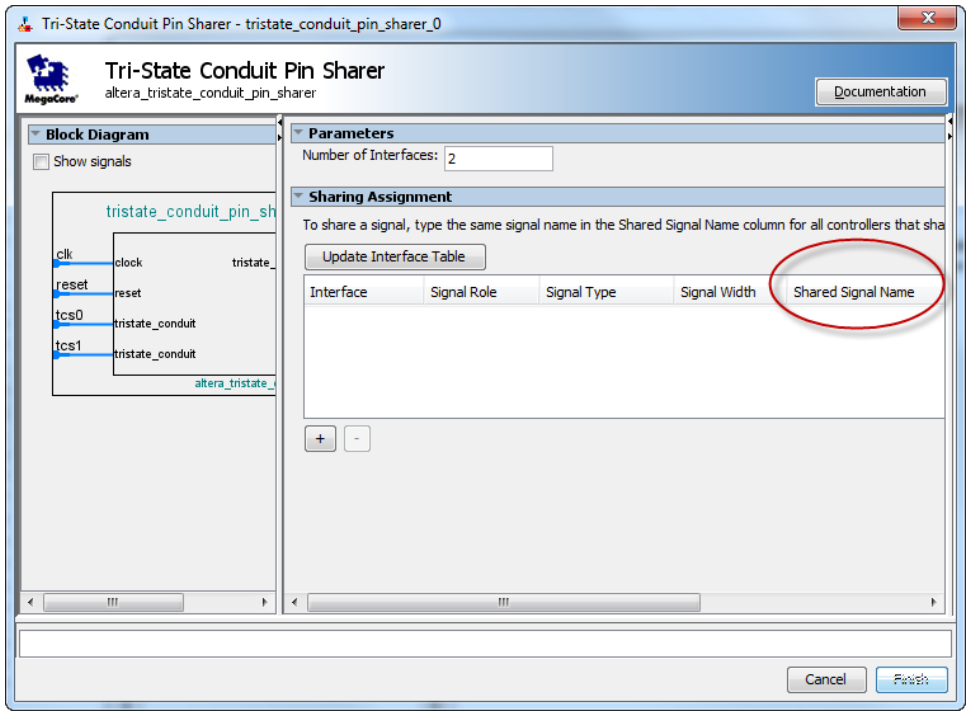
- **Memory-mapped agent interface**—This interface connects to a memory-mapped host, such as a processor.
- **Tristate Conduit Host interface**—The tri-state host interface usually connects to the tri-state conduit agent interface of the tri-state conduit pin sharer.
- **Clock sink**—The component's clock reference. You must connect this interface to a clock source.
- **Reset sink**—This interface connects to a reset source interface.

6.3.2. Tri-State Conduit Pin Sharer Intel FPGA IP

The Tri-State Conduit Pin Sharer Intel FPGA IP multiplexes between the signals of the connected tri-state controllers. You connect all signals from the tri-state controllers to the Tri-State Conduit Pin Sharer IP, and then use the parameter editor to specify the signals that are shared.

The parameter editor includes a **Shared Signal Name** column. If the widths of shared signals differ, the signals are aligned on their 0^{th} bit and the higher-order pins are driven to 0 whenever the smaller signal has control of the bus. Unshared signals always propagate through the pin sharer. The tri-state conduit pin sharer uses the round-robin arbiter to select between tri-state conduit controllers.

Figure 271. Tri-State Conduit Pin Sharer Parameter Editor



Note: All tri-state conduit components connected to a pin sharer must be in the same clock domain.

Related Information

Avalon Streaming Round Robin Scheduler Intel FPGA IP on page 423

6.3.3. Tri-State Conduit Bridge Intel FPGA IP

The Tri-State Conduit Bridge Intel FPGA IP instantiates bidirectional signals for each tri-state signal while passing all other signals straight through the component. The Tri-State Conduit Bridge registers all outgoing and incoming signals, which adds two cycles of latency for a read request. You must account for this additional pipelining when designing a custom controller. During reset, all outputs are placed in a high-impedance state. Outputs are enabled in the first clock cycle after reset is deasserted, and the output signals are then bidirectional.

6.4. Avalon Data Pattern Generator and Checker Intel FPGA IP

You can use the Avalon Data Pattern Generator IP to insert different error conditions, and then use the Avalon Data Pattern Checker IP to report these error conditions to the control interface, via an Avalon Memory-Mapped agent.

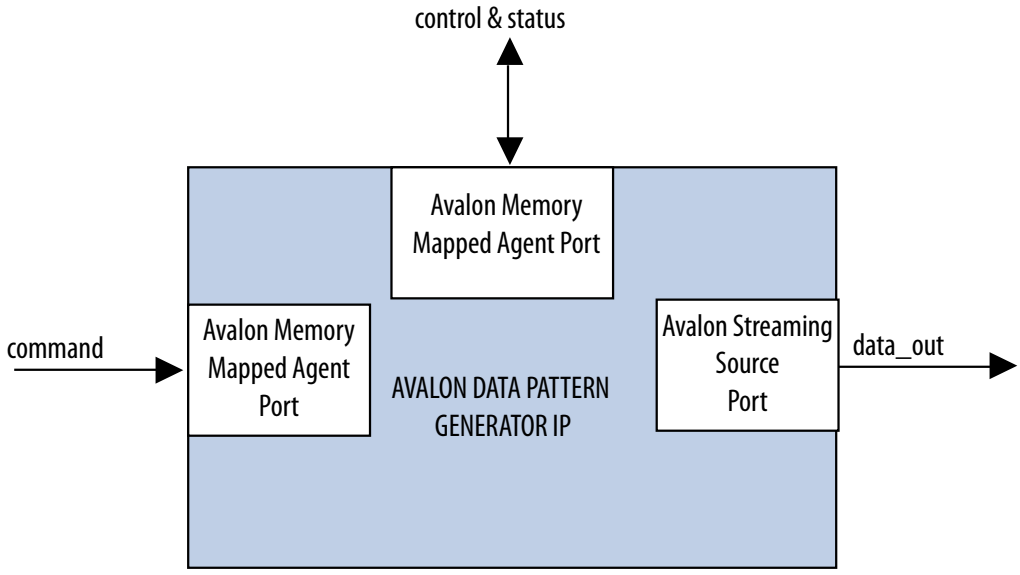
Similarly, for Avalon streaming interfaces, Avalon Data Pattern Generator IP generates data, and sends the data out on an Avalon streaming data interface. The Avalon Data Pattern Checker IP verifies the data. Optionally, you can format the data as packets, with accompanying `start_of_packet` and `end_of_packet` signals.

The **Throttle Seed** is the starting value for the throttle control random number generator. Intel recommends a unique value for each instance of the data pattern generator and checker IP cores in a system.

6.4.1. Avalon Data Pattern Generator Intel FPGA IP

The Avalon Data Pattern Generator IP accepts commands to generate data via an Avalon memory mapped command interface, and drives the generated data to an Avalon streaming data interface. You can parameterize most aspects of the Avalon streaming data interface, such as the number of error bits and data signal width, thus allowing you to test components with different interfaces.

Figure 272. Avalon Data Pattern Generator Intel FPGA IP



The data pattern is calculated as: *Symbol Value* = *Symbol Position in Packet* XOR *Data Error Mask*. Data that is not organized in packets is a single stream with no beginning or end. The Avalon Data Pattern Generator IP has a throttle register that is set via the Avalon memory mapped control interface. The Avalon Data Pattern Generator IP uses the value of the throttle register in conjunction with a pseudo-random number generator to throttle the data generation rate.

6.4.1.1. Avalon Data Pattern Generator IP Command Interface

The command interface for the Avalon Data Pattern Generator is a 32-bit Avalon memory mapped write agent that accepts data generation commands. It is connected to a 16-element deep FIFO, thus allowing a host peripheral to drive commands into the Avalon Data Pattern Generator IP.

The command interface maps to the following registers: `cmd_lo` and `cmd_hi`. The command is pushed into the FIFO when the register `cmd_lo` (address 0) is addressed. When the FIFO is full, the command interface asserts the `waitrequest` signal. You can create errors by writing to the register `cmd_hi` (address 1). The errors are cleared when 0 is written to this register, or its respective fields.

6.4.1.2. Avalon Data Pattern Generator IP Control and Status Interface

The control and status interface of the Avalon Data Pattern Generator IP is a 32-bit Avalon memory mapped agent that allows you to enable or disable the data generation, as well as set the throttle. This interface also provides generation-time information, such as the number of channels and whether data packets are supported.

6.4.1.3. Avalon Data Pattern Generator IP Output Interface

The output interface of the Avalon Data Pattern Generator IP is an Avalon streaming interface that optionally supports data packets. You can configure the output interface to align with your system requirements. Depending on the incoming stream of commands, the output data may contain interleaved packet fragments for different channels. To keep track of the current symbol's position within each packet, the Avalon Data Pattern Generator IP maintains an internal state for each channel.

You can configure the output interface of the Avalon Data Pattern Generator IP with the following parameters:

- **Number of Channels**—Number of channels that the Avalon Data Pattern Generator IP supports. Valid values are 1 to 256.
- **Data Bits Per Symbol**—Bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—Number of symbols (words) that are transferred per beat. Valid values are 1 to 256.
- **Include Packet Support**—Indicates whether packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Error Signal Width (bits)**—Width of the error signal on the output interface. Valid values are 0 to 31. A value of `0` indicates that the error signal is not in use.

Note: If you change only bits per symbol, and do not change the data width, errors are generated.

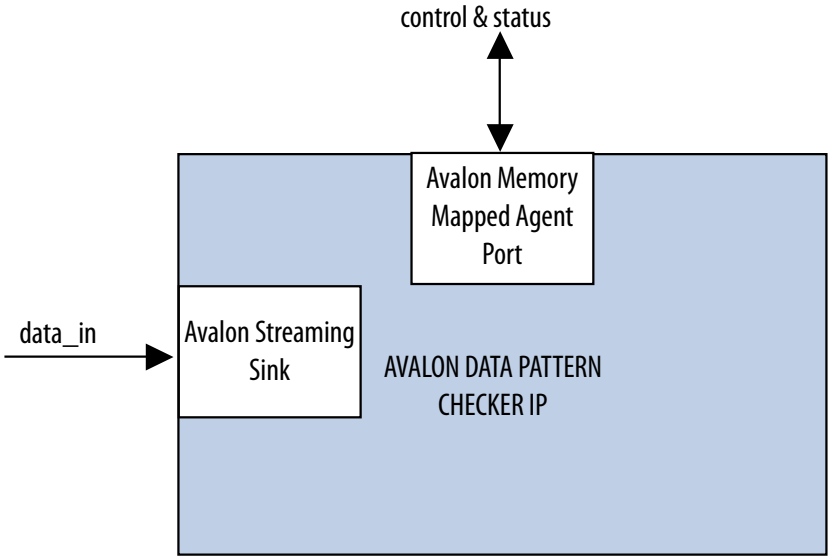
6.4.1.4. Avalon Data Pattern Generator IP Functional Parameter

The Avalon Data Pattern Generator IP functional parameter allows you to configure the Avalon Data Pattern Generator as an entire system.

6.4.2. Avalon Data Pattern Checker Intel FPGA IP

The Avalon Data Pattern Checker Intel FPGA IP accepts data via an Avalon streaming interface and verifies it against a predetermined pattern that the Avalon Data Pattern Generator Intel FPGA IP produces. The Avalon Data Pattern Checker IP reports any exceptions to the control interface. You can parameterize most aspects of the Avalon Data Pattern Checker's Avalon streaming interface, such as the number of error bits, and the data signal width. This IP allows you to test components with different interfaces. The Avalon Data Pattern Checker IP has a throttle register that is set via the Avalon memory mapped control interface. The value of the throttle register controls the rate at which data is accepted.

Figure 273. Avalon Data Pattern Checker Intel FPGA IP



The Avalon Data Pattern Checker IP detects exceptions and reports them to the control interface via a 32-element deep internal FIFO. Possible exceptions are data error, missing start-of-packet (SOP), missing end-of-packet (EOP), and signaled error.

As each exception occurs, an exception descriptor is pushed into the FIFO. If the same exception occurs more than once consecutively, only one exception descriptor is pushed into the FIFO. All exceptions are ignored when the FIFO is full. Exception descriptors are deleted from the FIFO after they are read by the control and status interface.

6.4.2.1. Avalon Data Pattern Checker IP Input Interface

The Avalon Data Pattern Checker IP input interface is an Avalon streaming interface that optionally supports data packets. You can configure the input interface to align with your system requirements. Incoming data may contain interleaved packet fragments. To keep track of the current symbol’s position, the test pattern checker maintains an internal state for each channel.

6.4.2.2. Avalon Data Pattern Checker IP Control and Status Interface

The Avalon Data Pattern Checker IP control and status interface is a 32-bit Avalon memory mapped agent that allows you to enable or disable data acceptance, as well as set the throttle. This interface provides generation-time information, such as the number of channels and whether the Avalon Data Pattern Checker supports data packets. The control and status interface also provides information on the exceptions detected by the Avalon Data Pattern Checker IP. The interface obtains this information by reading from the exception FIFO.

6.4.2.3. Avalon Data Pattern Checker IP Functional Parameter

The Avalon Data Pattern Checker IP functional parameter allows you to configure the Avalon Data Pattern Checker IP as a whole system.

6.4.2.4. Avalon Data Pattern Checker Input Parameters

You can configure the input interface of the Avalon Data Pattern Checker IP using the following parameters:

- **Data Bits Per Symbol**—Bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—Number of symbols (words) that are transferred per beat. Valid values are 1 to 32.
- **Include Packet Support**—Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Number of Channels**—Number of channels that the test pattern checker supports. Valid values are 1 to 256.
- **Error Signal Width (bits)**—Width of the `error` signal on the input interface. Valid values are 0 to 31. A value of `0` indicates that the `error` signal in not in use.

Note: If you change only bits per symbol, and do not change the data width, errors are generated.

6.4.3. Avalon Data Pattern Generator and Checker IP Software Programming Model

The HAL system library support, software files, and register maps describe the software programming model for the test pattern generator and checker cores.

6.4.3.1. HAL System Library Support

For Nios II processor users, Intel provides HAL system library drivers that allow you to initialize and access the Avalon Data Pattern Generator and Checker IPs. Intel recommends you use the provided drivers to access the IPs instead of accessing the registers directly.

For Nios II IDE users, copy the provided drivers from the following installation folders to your software application directory:

- `<IP installation directory>/ip/sopc_builder_ip/altera_Avalon_data_source/HAL`
- `<IP installation directory>/ip/sopc_builder_ip/altera_Avalon_data_sink/HAL`

Note: This instruction does not apply if you use the Nios II command-line tools.

6.4.3.2. Avalon Data Pattern Generator and Checker IP Files

The following files define the low-level access to the hardware, and provide the routines for the HAL device drivers.

- Avalon Data Pattern Generator IP files in `<installation directory>/ip/sopc_builder_ip/altera_Avalon_data_source/HAL`:
  - `data_source_regs.h`—header file that defines the test pattern generator's register maps.
  - `data_source_util.h`, `data_source_util.c`—header and source code for the functions and variables required to integrate the driver into the HAL system library.
- Avalon Data Pattern Checker IP files in `<installation directory>/ip/sopc_builder_ip/altera_Avalon_data_sink/HAL`
  - `data_sink_regs.h`—header file that defines the IP register maps.
  - `data_sink_util.h`, `data_sink_util.c`—header and source code for the functions and variables required to integrate the driver into the HAL system library.

Note: Do not modify the Avalon Data Pattern Generator or Avalon Data Pattern Checker IP files.

6.4.3.3. Avalon Data Pattern Generator and Checker IP Register Maps

6.4.3.3.1. Avalon Data Pattern Generator IP Control and Status Registers

Table 137. Avalon Data Pattern Generator IP Control and Status Register Map

Each register is 32-bits wide.

Offset	Register Name
base + 0	`status`
base + 1	`control`
base + 2	`fill`

Table 138. Avalon Data Pattern Generator IP Status Register Bits

Bits	Name	Access	Description
[15:0]	ID	RO	A constant value of 0x64.
[23:16]	NUMCHANNELS	RO	The configured number of channels.
[30:24]	NUMSYMBOLS	RO	The configured number of symbols per beat.
[31]	SUPPORTPACKETS	RO	A value of 1 indicates data packet support.

Table 139. Avalon Data Pattern Generator IP Control Register Bits

Bits	Name	Access	Description
[0]	ENABLE	RW	Setting this bit to 1 enables the data pattern generator IP.
[7:1]	Reserved		
[16:8]	THROTTLE	RW	Specifies the throttle value which can be between 0–256, inclusively. The Data Pattern Generator IP uses this value in conjunction with a pseudo-random number generator to throttle the data generation rate.
[17]	SOFT RESET	RW	When this bit is set to 1, all internal counters and statistics are reset. Write 0 to this bit to exit reset.
[31:18]	Reserved		

Table 140. Avalon Data Pattern Generator IP Fill Register Bits

Bits	Name	Access	Description
[0]	BUSY	RO	A value of 1 indicates that data transmission is in progress, or that there is at least one command in the command queue.
[6:1]	Reserved		
[15:7]	FILL	RO	The number of commands currently in the command FIFO.
[31:16]	Reserved		

6.4.3.3.2. Avalon Data Pattern Generator IP Command Registers

Table 141. Avalon Data Pattern Generator IP Command Register Map

Shows the offset for the command registers. Each register is 32-bits wide.

Offset	Register Name
base + 0	cmd_lo
base + 1	cmd_hi

The cmd_lo is pushed into the FIFO only when the cmd_lo register is addressed.

Table 142. `cmd_lo` Register Bits

Bits	Name	Access	Description
[15:0]	SIZE	RW	The segment size in symbols. Except for the last segment in a packet, the size of all segments must be a multiple of the configured number of symbols per beat. If this condition is not met, the Data Pattern Generator IP inserts additional symbols to the segment to ensure the condition is fulfilled.
[29:16]	CHANNEL	RW	The channel to send the segment on. If the `channel` signal is less than 14 bits wide, the Data Pattern Generator IP uses the low order bits of this register to drive the signal.
[30]	SOP	RW	Set this bit to 1 when sending the first segment in a packet. This bit is ignored when data packets are not supported.
[31]	EOP	RW	Set this bit to 1 when sending the last segment in a packet. This bit is ignored when data packets are not supported.

Table 143. `cmd_hi` Register Bits

Bits	Name	Access	Description
[15:0]	SIGNALED ERROR	RW	Specifies the value to drive the `error` signal. A non-zero value creates a signaled error.
[23:16]	DATA ERROR	RW	The output data is XORed with the contents of this register to create data errors. To stop creating data errors, set this register to 0.
[24]	SUPPRESS SOP	RW	Set this bit to 1 to suppress the assertion of the `startofpacket` signal when the first segment in a packet is sent.
[25]	SUPRESS EOP	RW	Set this bit to 1 to suppress the assertion of the `endofpacket` signal when the last segment in a packet is sent.

6.4.3.3.3. Avalon Data Pattern Checker IP Control and Status Registers

Table 144. Avalon Data Pattern Generator and Checker IP Control and Status Register Map

Shows the offset for the control and status registers. Each register is 32 bits wide.

Offset	Register Name
base + 0	`status`
base + 1	`control`
base + 2	Reserved
base + 3	
base + 4	
base + 5	`exception_descriptor`
base + 6	`indirect_select`
base + 7	`indirect_count`

Table 145. Avalon Data Pattern Checker IP Status Register Bits

Bits	Name	Access	Description
[15:0]	ID	RO	Contains a constant value of 0x65.
[23:16]	NUMCHANNELS	RO	The configured number of channels.
[30:24]	NUMSYMBOLS	RO	The configured number of symbols per beat.
[31]	SUPPORTPACKETS	RO	A value of 1 indicates packet support.

Table 146. Avalon Data Pattern Checker IP Control Register Bits

Bits	Name	Access	Description
[0]	ENABLE	RW	Setting this bit to 1 enables the Data Pattern Checker IP.
[7:1]	Reserved		
[16:8]	THROTTLE	RW	Specifies the throttle value which can be between 0–256, inclusively. Platform Designer uses this value in conjunction with a pseudo-random number generator to throttle the data generation rate. Setting THROTTLE to 0 stops the Avalon Data Pattern Checker IP. Setting it to 256 causes the Avalon Data Pattern Checker IP to run at full throttle. Values between 0–256 result in a data rate proportional to the throttle value.
[17]	SOFT RESET	RW	When this bit is set to 1, all internal counters and statistics are reset. Write 0 to this bit to exit reset.
[31:18]	Reserved		

If there is no exception, reading the `exception_descriptor` register bit register returns 0.

Table 147. `exception_descriptor` Register Bits

Bits	Name	Access	Description
[0]	DATA ERROR	RO	A value of 1 indicates that an error is detected in the incoming data.
[1]	MISSINGSOP	RO	A value of 1 indicates missing start-of-packet.
[2]	MISSINGEOP	RO	A value of 1 indicates missing end-of-packet.
[7:3]	Reserved		
[15:8]	SIGNALLED ERROR	RO	The value of the `error` signal.
[23:16]	Reserved		
[31:24]	CHANNEL	RO	The channel on which the exception was detected.

Table 148. `indirect_select` Register Bits

Bit	Bits Name	Access	Description
[7:0]	INDIRECT CHANNEL	RW	Specifies the channel number that applies to the INDIRECT PACKET COUNT, INDIRECT SYMBOL COUNT, and INDIRECT ERROR COUNT registers.
[15:8]	Reserved		
[31:16]	INDIRECT ERROR	RO	The number of data errors that occurred on the channel specified by INDIRECT CHANNEL.

Table 149. `indirect_count` Register Bits

Bit	Bits Name	Access	Description
[15:0]	INDIRECT PACKET COUNT	RO	The number of data packets received on the channel specified by INDIRECT CHANNEL.
[31:16]	INDIRECT SYMBOL COUNT	RO	The number of symbols received on the channel specified by INDIRECT CHANNEL.

6.4.4. Avalon Data Pattern Generator IP API

The following subsections describe application programming interface (API) for the Avalon Data Pattern Generator IP.

Note: API functions are currently not available from the interrupt service routine (ISR).

6.4.4.1. data_source_reset()

Table 150. data_source_reset()

Information Type	Description
Prototype	`void data_source_reset(alt_u32 base);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status agent.
Returns	void
Description	Resets the Avalon Data Pattern Generator IP, including all internal counters and FIFOs. The control and status registers are not reset by this function.

6.4.4.2. data_source_init()

Table 151. data_source_init()

Information Type	Description
Prototype	`int data_source_init(alt_u32 base, alt_u32 command_base);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status agent. `command_base`—Base address of the command agent.
Returns	`1`—Initialization is successful. `0`—Initialization is unsuccessful.
Description	Performs the following operations to initialize the Avalon Data Pattern Generator IP: • Resets and disables the Avalon Data Pattern Generator IP. • Sets the maximum throttle. • Clears all inserted errors.

6.4.4.3. data_source_get_id()

Table 152. data_source_get_id()

Information Type	Description
Prototype	`int data_source_get_id(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	Avalon Data Pattern Generator IP identifier.
Description	Retrieves the Avalon Data Pattern Generator IP identifier.

6.4.4.4. data_source_get_supports_packets()

Table 153. data_source_get_supports_packets()

Information Type	Description
Prototype	`int data_source_init(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	`1`—Data packets are supported. `0`—Data packets are not supported.
Description	Checks if the Avalon Data Pattern Generator IP supports data packets.

6.4.4.5. data_source_get_num_channels()

Table 154. data_source_get_num_channels()

Description	Description
Prototype	`int data_source_get_num_channels(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status agent.
Returns	Number of channels supported.
Description	Retrieves the number of channels supported by the Avalon Data Pattern Generator IP.

6.4.4.6. data_source_get_symbols_per_cycle()

Table 155. data_source_get_symbols_per_cycle()

Description	Description
Prototype	`int data_source_get_symbols(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status agent.
Returns	Number of symbols transferred in a beat.
Description	Retrieves the number of symbols transferred by the Avalon Data Pattern Generator IP in each beat.

6.4.4.7. data_source_get_enable()

Table 156. data_source_get_enable()

Information Type	Description
Prototype	`int data_source_get_enable(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status agent.
Returns	Value of the ENABLE bit.
Description	Retrieves the value of the ENABLE bit.

6.4.4.8. data_source_set_enable()

Table 157. data_source_set_enable()

Information Type	Description
Prototype	`void data_source_set_enable(alt_u32 base, alt_u32 value);`
Thread-safe	No

continued...

Information Type	Description
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status agent. `value`— ENABLE bit set to the value of this parameter.
Returns	`void`
Description	Enables or disables the Avalon Data Pattern Generator IP. When disabled, the Avalon Data Pattern Generator IP stops data transmission but continues to accept commands and stores them in the FIFO

6.4.4.9. data_source_get_throttle()

Table 158. data_source_get_throttle()

Information Type	Description
Prototype	`int data_source_get_throttle(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	Throttle value.
Description	Retrieves the current throttle value.

6.4.4.10. data_source_set_throttle()

Table 159. data_source_set_throttle()

Information Type	Description
Prototype	`void data_source_set_throttle(alt_u32 base, alt_u32 value);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status agent. `value`—Throttle value.
Returns	`void`
Description	Sets the throttle value, which can be between 0–256 inclusively. The throttle value, when divided by 256 yields the rate at which the Avalon Data Pattern Generator IP sends data.

6.4.4.11. data_source_is_busy()

Table 160. data_source_is_busy()

Information Type	Description
Prototype	`int data_source_is_busy(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*

continued...

Information Type	Description
Parameters	`base`—Base address of the control and status agent.
Returns	`1`—Avalon Data Pattern Generator IP is busy. `0`—Avalon Data Pattern Generator IP is not busy.
Description	Checks if the Avalon Data Pattern Generator IP is busy. The Avalon Data Pattern Generator IP is busy when it is sending data or has data in the command FIFO to be sent.

6.4.4.12. data_source_fill_level()

Table 161. data_source_fill_level()

Information Type	Description
Prototype	`int data_source_fill_level(alt_u32 base);`
Thread-safe	Yes
Include	*<data_source_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	Number of commands in the command FIFO.
Description	Retrieves the number of commands currently in the command FIFO.

6.4.4.13. data_source_send_data()

Table 162. data_source_send_data()

Information Type	Description
Prototype	`int data_source_send_data(alt_u32 cmd_base, alt_u16 channel, alt_u16 size, alt_u32 flags, alt_u16 error, alt_u8 data_error_mask);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	`cmd_base`—Base address of the command agent. `channel`—Channel to send the data. `size`—Data size. `flags` —Specifies whether to send or suppress SOP and EOP signals. Valid values are `DATA_SOURCE_SEND_SOP`, `DATA_SOURCE_SEND_EOP`, `DATA_SOURCE_SEND_SUPRESS_SOP` and `DATA_SOURCE_SEND_SUPRESS_EOP`. `error`—Value asserted on the `error` signal on the output interface. `data_error_mask`—Parameter and the data are XORed together to produce erroneous data.
Returns	Returns 1.
Description	Sends a data fragment to the specified channel. If data packets are supported, applications must ensure consistent usage of SOP and EOP in each channel. Except for the last segment in a packet, the length of each segment is a multiple of the data width. If data packets are not supported, applications must ensure that there are no SOP and EOP indicators in the data. The length of each segment in a packet is a multiple of the data width.

6.4.5. Avalon Data Pattern Checker IP API

The following subsections describe API for the Avalon Data Pattern Checker IP. The API functions are currently not available from the ISR.

6.4.5.1. data_sink_reset()

Table 163. data_sink_reset()

Information Type	Description
Prototype	`void data_sink_reset(alt_u32 base);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	`void`
Description	Resets the Avalon Data Pattern Checker IP, including all internal counters.

6.4.5.2. data_sink_init()

Table 164. data_sink_init()

Information Type	Description
Prototype	`int data_source_init(alt_u32 base);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	`1`—Initialization is successful. `0`—Initialization is unsuccessful.
Description	Performs the following operations to initialize the Avalon Data Pattern Checker IP: • Resets and disables the Avalon Data Pattern Checker IP. • Sets the throttle to the maximum value.

6.4.5.3. data_sink_get_id()

Table 165. data_sink_get_id()

Information Type	Description
Prototype	`int data_sink_get_id(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	Avalon Data Pattern Checker IP identifier.
Description	Retrieves the Avalon Data Pattern Checker IP identifier.

6.4.5.4. data_sink_get_supports_packets()

Table 166. data_sink_get_supports_packets()

Information Type	Description
Prototype	`int data_sink_init(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	`1`—Data packets are supported. `0`—Data packets are not supported.
Description	Checks if the Avalon Data Pattern Checker IP supports data packets.

6.4.5.5. data_sink_get_num_channels()

Table 167. data_sink_get_num_channels()

Information Type	Description
Prototype	`int data_sink_get_num_channels(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	Number of channels supported.
Description	Retrieves the number of channels supported by the Avalon Data Pattern Checker IP.

6.4.5.6. data_sink_get_symbols_per_cycle()

Table 168. data_sink_get_symbols_per_cycle()

Information Type	Description
Prototype	`int data_sink_get_symbols(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	Number of symbols received in a beat.
Description	Retrieves the number of symbols received by the Avalon Data Pattern Checker IP in each beat.

6.4.5.7. data_sink_get_enable()

Table 169. data_sink_get_enable()

Information Type	Description
Prototype	`int data_sink_get_enable(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`base`—Base address of the control and status agent.
Returns	Value of the ENABLE bit.
Description	Retrieves the value of the ENABLE bit.

6.4.5.8. data_sink_set enable()

Table 170. data_sink_set enable()

Information Type	Description
Prototype	`void data_sink_set_enable(alt_u32 base, alt_u32 value);`
Thread-safe	No

continued...

Information Type	Description
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status agent. value—ENABLE bit is set to the value of the parameter.
Returns	void
Description	Enables the Avalon Data Pattern Checker IP.

6.4.5.9. data_sink_get_throttle()

Table 171. data_sink_get_throttle()

Information Type	Description
Prototype	`int data_sink_get_throttle(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status agent.
Returns	Throttle value.
Description	Retrieves the throttle value.

6.4.5.10. data_sink_set_throttle()

Table 172. data_sink_set_throttle()

Information Type	Description
Prototype	`void data_sink_set_throttle(alt_u32 base, alt_u32 value);`
Thread-safe	No
Include:	*<data_sink_util.h >*
Parameters	base—Base address of the control and status agent. value—Throttle value.
Returns	void
Description	Sets the throttle value, which can be between 0–256 inclusively. The throttle value, when divided by 256 yields the rate at which the Avalon Data Pattern Checker IP receives data.

6.4.5.11. data_sink_get_packet_count()

Table 173. data_sink_get_packet_count()

Information Type	Description
Prototype	`int data_sink_get_packet_count(alt_u32 base, alt_u32 channel);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status agent.

continued...

Information Type	Description
	channel—Channel number.
Returns	Number of data packets received on the channel.
Description	Retrieves the number of data packets received on a channel.

6.4.5.12. data_sink_get_error_count()

Table 174. data_sink_get_error_count()

Information Type	Description
Prototype	`int data_sink_get_error_count(alt_u32 base, alt_u32 channel);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status agent. channel—Channel number.
Returns	Number of errors received on the channel.
Description	Retrieves the number of errors received on a channel.

6.4.5.13. data_sink_get_symbol_count()

Table 175. data_sink_get_symbol_count()

Information Type	Description
Prototype	`int data_sink_get_symbol_count(alt_u32 base, alt_u32 channel);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status agent. channel—Channel number.
Returns	Number of symbols received on the channel.
Description	Retrieves the number of symbols received on a channel.

6.4.5.14. data_sink_get_exception()

Table 176. data_sink_get_exception()

Information Type	Description
Prototype	`int data_sink_get_exception(alt_u32 base);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status agent.
Returns	First exception descriptor in the exception FIFO. 0—No exception descriptor found in the exception FIFO.
Description	Retrieves the first exception descriptor in the exception FIFO and pops it off the FIFO.

6.4.5.15. data_sink_exception_is_exception()

Table 177. data_sink_exception_is_exception()

Information Type	Description
Prototype	`int data_sink_exception_is_exception(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor
Returns	`1`—Indicates an exception. `0`—No exception.
Description	Checks if an exception descriptor describes a valid exception.

6.4.5.16. data_sink_exception_has_data_error()

Table 178. data_sink_exception_has_data_error()

Information Type	Description
Prototype	`int data_sink_exception_has_data_error(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	`1`—Data has errors. `0`—No errors.
Description	Checks if an exception indicates erroneous data.

6.4.5.17. data_sink_exception_has_missing_sop()

Table 179. data_sink_exception_has_missing_sop()

Information Type	Description
Prototype	`int data_sink_exception_has_missing_sop(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	`1`—Missing SOP. `0`—Other exception types.
Description	Checks if an exception descriptor indicates missing SOP.

6.4.5.18. data_sink_exception_has_missing_eop()

Table 180. data_sink_exception_has_missing_eop()

Information Type	Description
Prototype	`int data_sink_exception_has_missing_eop(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	`1`—Missing EOP. `0`—Other exception types.
Description	Checks if an exception descriptor indicates missing EOP.

6.4.5.19. data_sink_exception_signalled_error()

Table 181. data_sink_exception_signalled_error()

Information Type	Description
Prototype	`int data_sink_exception_signalled_error(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	Signal error value.
Description	Retrieves the value of the signaled error from the exception.

6.4.5.20. data_sink_exception_channel()

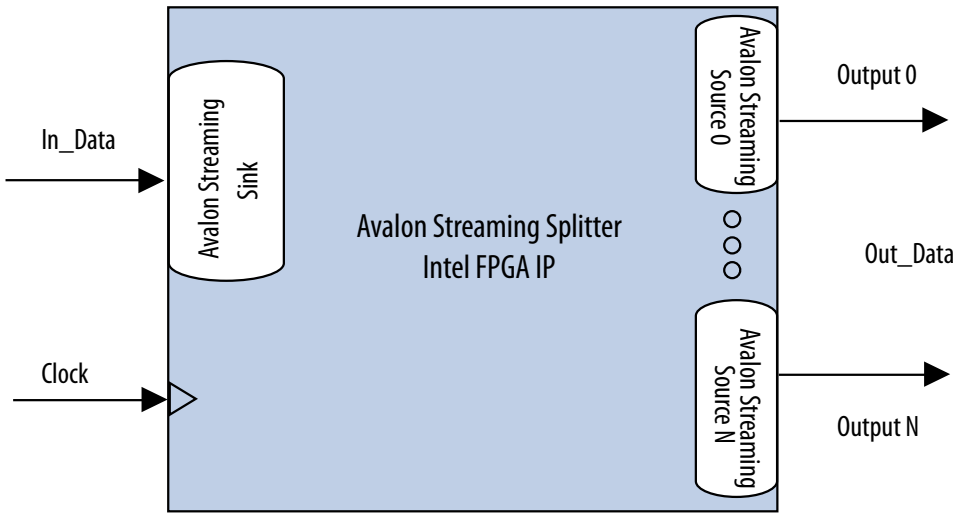
Table 182. data_sink_exception_channel()

Information Type	Description
Prototype	`int data_sink_exception_channel(int exception);`
Thread-safe	Yes
Include	*<data_sink_util.h >*
Parameters	`exception`—Exception descriptor.
Returns	Channel number on which an exception occurred.
Description	Retrieves the channel number on which an exception occurred.

6.5. Avalon Streaming Splitter Intel FPGA IP

The Avalon Streaming Splitter Intel FPGA IP allows you to replicate transactions from an Avalon streaming sink interface to multiple Avalon streaming source interfaces. This IP supports from 1 to 16 outputs.

Figure 274. Avalon Streaming Splitter Intel FPGA IP



The Avalon Streaming Splitter IP copies input signals from the input interface to the corresponding output signals of each output interface without altering the size or functionality. This includes all signals except for the `ready` signal. The IP includes a clock signal to determine the Avalon streaming interface and clock domain where the IP resides. Because the Avalon Streaming Splitter IP does not use the `clock` signal internally, latency is not introduced when using this IP.

6.5.1. Avalon Streaming Splitter Intel FPGA IP Backpressure

The Avalon Streaming Splitter Intel FPGA IP integrates with backpressure by AND-ing the `ready` signals from the output interfaces and sending the result to the input interface. As a result, if an output interface deasserts the `ready` signal, the input interface receives the deasserted `ready` signal, as well. This functionality ensures that backpressure on the output interfaces is propagated to the input interface.

When the **Qualify Valid Out** option is enabled, the `out_valid` signals on all other output interfaces are gated when backpressure is applied from one output interface. In this case, when any output interface deasserts its `ready` signal, the `out_valid` signals on the other output interfaces are also deasserted.

When the **Qualify Valid Out** option is disabled, the output interfaces have a non-gated `out_valid` signal when backpressure is applied. In this case, when an output interface deasserts its ready signal, the `out_valid` signals on the other output interfaces are not affected.

Because the logic is combinational, the Intel FPGA IP introduces no latency.

6.5.2. Avalon Streaming Splitter Intel FPGA IP Interfaces

The Avalon Streaming Splitter Intel FPGA IP supports streaming data, with optional packet, channel, and error signals. The Intel FPGA IP propagates backpressure from any output interface to the input interface.

Table 183. Avalon Streaming Splitter Intel FPGA IP Support

Feature	Support
Backpressure	Ready latency = 0.
Data Width	Configurable.
Channel	Supported (optional).
Error	Supported (optional).
Packet	Supported (optional).

6.5.3. Avalon Streaming Splitter Intel FPGA IP Parameters

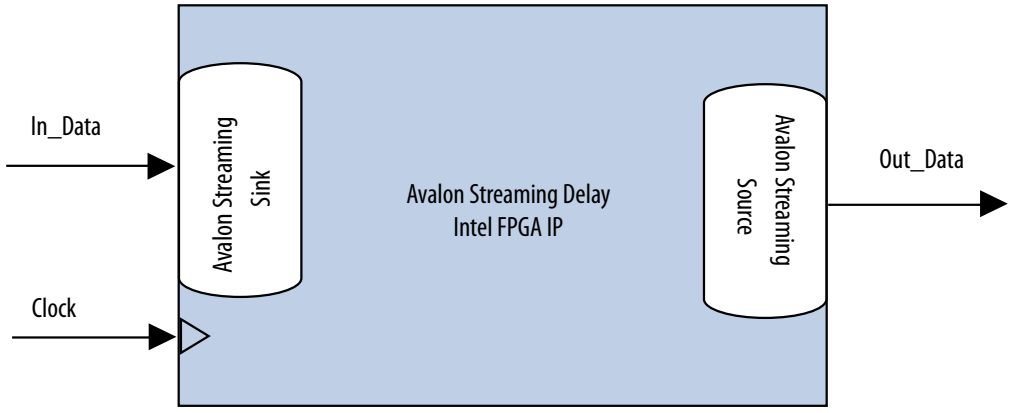
Table 184. Avalon Streaming Splitter Intel FPGA IP Parameters

Parameter	Legal Values	Default Value	Description
Number Of Outputs	1 to 16	2	The number of output interfaces. Platform Designer supports 1 for some systems where no duplicated output is required.
Qualify Valid Out	Enabled, Disabled	Enabled	If enabled, the `out_valid` signal of all output interfaces is gated when back pressure is applied.
Data Width	1–512	8	The width of the data on the Avalon streaming data interfaces.
Bits Per Symbol	1–512	8	The number of bits per symbol for the input and output interfaces. For example, byte-oriented interfaces have 8-bit symbols.
Use Packets	Enabled, Disabled	Disabled	Enable support of data packet transfers. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
Use Channel	Enabled, Disabled	Disabled	Enable the channel signal.
Channel Width	0-8	1	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is set to 0.
Max Channels	0-255	1	The maximum number of channels that a data interface can support. This parameter is disabled when **Use Channel** is set to 0.
Use Error	Enabled, Disabled	Disabled	Enable the error signal.
Error Width	0–31	1	The width of the `error` signal on the output interfaces. A value of `0` indicates that the IP is not using the `error` signal. This parameter is disabled when **Use Error** is set to `0`.

6.6. Avalon Streaming Delay Intel FPGA IP

The Avalon Streaming Delay Intel FPGA IP provides a solution to delay Avalon streaming transactions by a constant number of clock cycles. This IP supports up to 16 clock cycle delays.

Figure 275. Avalon Streaming Delay Intel FPGA IP



The Avalon Streaming Delay Intel FPGA IP adds a delay between the input and output interfaces. The IP accepts transactions presented on the input interface and reproduces them on the output interface N cycles later without changing the transaction.

The input interface delays the input signals by a constant N number of clock cycles to the corresponding output signals of the output interface. The **Number Of Delay Clocks** parameter defines the constant N, which must be from 0 to 16. The change of the `in_valid` signal is reflected on the `out_valid` signal exactly N cycles later.

6.6.1. Avalon Streaming Delay Intel FPGA IP Reset Signal

The Avalon Streaming Delay Intel FPGA IP has a `reset` signal that is synchronous to the `clk` signal. When the IP asserts the `reset` signal, the output signals are held at 0. After the `reset` signal is deasserted, the output signals are held at 0 for N clock cycles. The delayed values of the input signals are then reflected at the output signals after N clock cycles.

6.6.2. Avalon Streaming Delay Intel FPGA IP Interfaces

The Avalon Streaming Delay Intel FPGA IP supports streaming data, with optional packet, channel, and error signals. The Avalon Streaming Delay Intel FPGA IP does not support backpressure.

Table 185. Avalon Streaming Delay Intel FPGA IP Support

Feature	Support
Backpressure	Not supported.
Data Width	Configurable.
Channel	Supported (optional).
Error	Supported (optional).
Packet	Supported (optional).

6.6.3. Avalon Streaming Delay Intel FPGA IP Parameters

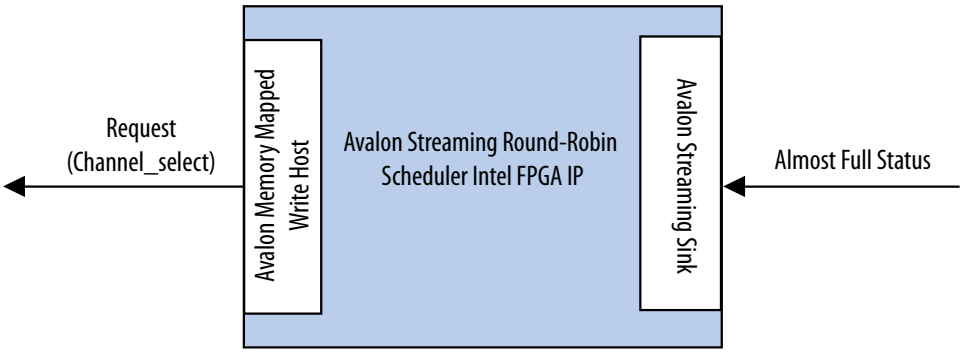
Table 186. Avalon Streaming Delay Intel FPGA IP Parameters

Parameter	Legal Values	Default Value	Description
Number Of Delay Clocks	0 to 16	1	Specifies the delay the IP introduces, in clock cycles. Platform Designer supports 0 for some systems where no delay is required.
Data Width	1–512	8	The width of the data on the Avalon streaming data interfaces.
Bits Per Symbol	1–512	8	The number of bits per symbol for the input and output interfaces. For example, byte-oriented interfaces have 8-bit symbols.
Use Packets	0 or 1	0	Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
Use Channel	0 or 1	0	The option to enable or disable the channel signal.
Channel Width	0-8	1	The width of the `channel` signal on the data interfaces. This parameter is disabled when **Use Channel** is set to 0.
Max Channels	0-255	1	The maximum number of channels that a data interface can support. This parameter is disabled when **Use Channel** is set to 0.
Use Error	0 or 1	0	The option to enable or disable the error signal.
Error Width	0–31	1	The width of the `error` signal on the output interfaces. A value of 0 indicates that the error signal is not in use. This parameter is disabled when **Use Error** is set to 0.

6.7. Avalon Streaming Round Robin Scheduler Intel FPGA IP

The Avalon Streaming Round Robin Scheduler Intel FPGA IP controls the read operations from a multi-channel Avalon streaming component that buffers data by channels. The IP reads the almost-full threshold values from the multiple channels in the multi-channel component, and then issues the read request to the Avalon streaming source according to a round-robin scheduling algorithm.

Figure 276. Avalon Streaming Round Robin Scheduler Intel FPGA IP



In a multi-channel component, the IP can store data either in the sequence that it comes in (FIFO), or in segments according to the channel. When data is stored in segments according to channels, a scheduler is needed to schedule the read operations.

6.7.1. Avalon Streaming Round Robin Scheduler IP Almost-Full Status Interface

The Almost-Full Status interface is an Avalon streaming sink interface that collects the almost-full status from the sink components for the channels in the sequence provided.

Table 187. Avalon Streaming Interface Feature Support

Feature	Property
Backpressure	Not supported
Data Width	Data width = 1; Bits per symbol = 1
Channel	Maximum channel = 32; Channel width = 5
Error	Not supported
Packet	Not supported

6.7.2. Avalon Streaming Round Robin Scheduler IP Request Interface

The Request Interface is an Avalon memory mapped write host interface that requests data from a specific channel. The Avalon Streaming Round Robin Scheduler cycles through the channels it supports and schedules data to be read.

6.7.3. Avalon Streaming Round Robin Scheduler IP Operation

If a particular channel is almost full, the Avalon Streaming Round Robin Scheduler does not schedule data to be read from that channel in the source component.

The scheduler only requests 1 bit of data from a channel at each transaction. To request 1 bit of data from channel *n*, the scheduler writes the value 1 to address (4 ×*n*). For example, if the scheduler is requesting data from channel 3, the scheduler writes 1 to address `0xC`. At every clock cycle, the scheduler requests data from the next channel. Therefore, if the scheduler starts requesting from channel 1, at the next clock cycle, it requests from channel 2. The scheduler does not request data from a particular channel if the almost-full status for the channel is asserted. In this case, the scheduler uses one clock cycle without a request transaction.

The Avalon Streaming Round Robin Scheduler cannot determine if the requested component is able to service the request transaction. The component asserts `waitrequest` when it cannot accept new requests.

Table 188. Avalon Streaming Round Robin Scheduler Ports

Signal	Direction	Description
Clock and Reset		
`clk`	In	Clock reference.
		continued...

Signal	Direction	Description
reset_n	In	Asynchronous active low reset.
Avalon Memory Mapped Request Interface		
request_address (log_2 Max_Channels–1:0)	Out	The write address that indicates which channel has the request.
request_write	Out	Write enable signal.
request_writedata	Out	The amount of data requested from the particular channel. This value is always fixed at 1.
request_waitrequest	In	Wait request signal that pauses the scheduler when the agent cannot accept a new request.
Avalon Streaming Almost-Full Status Interface		
almost_full_valid	In	Indicates that almost_full_channel and almost_full_data are valid.
almost_full_channel (Channel_Width–1:0)	In	Indicates the channel for the current status indication.
almost_full_data (log_2 Max_Channels–1:0)	In	A 1-bit signal that is asserted high to indicate that the channel indicated by almost_full_channel is almost full.

6.7.4. Avalon Streaming Round Robin Scheduler IP Parameters

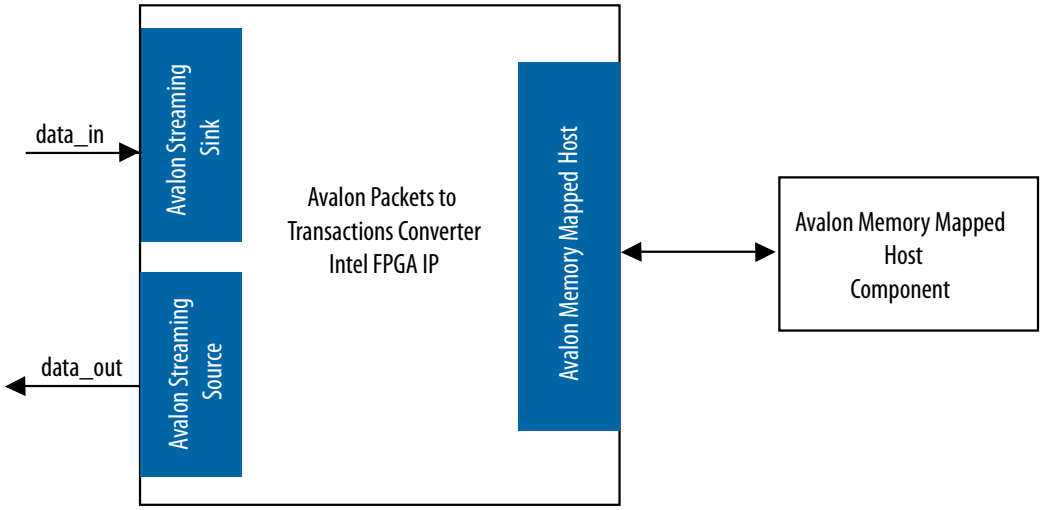
Table 189. Avalon Streaming Round Robin Scheduler IP Parameters

Parameters	Legal Values	Default Value	Description
Number of channels	2–32	2	Specifies the number of channels the Avalon Streaming Round Robin Scheduler supports.
Use almost-full status	Enabled, Disabled	Disabled	If enabled, the scheduler uses the almost-full interface. If not, the IP requests data from the next channel at the next clock cycle.

6.8. Avalon Packets to Transactions Converter Intel FPGA IP

The Avalon Packets to Transactions Converter Intel FPGA IP receives streaming data from upstream components and initiates Avalon memory mapped transactions. The IP then returns Avalon memory mapped transaction responses to the requesting components.

Figure 277. Avalon Packets to Transactions Converter Intel FPGA IP



Note: The SPI Agent to Avalon Host Bridge, and the JTAG to Avalon Host Bridge, are examples of the Packets to Transactions Converter IP. For more information, refer to the *Avalon Interface Specifications*.

Related Information

Avalon Interface Specifications

6.8.1. Avalon Packets to Transactions Converter IP Interfaces

Table 190. Properties of Avalon Streaming Interfaces

Feature	Property
Backpressure	Ready latency = 0.
Data Width	Data width = 8 bits; Bits per symbol = 8.
Channel	Not supported.
Error	Not used.
Packet	Supported.

The Avalon memory mapped host interface supports read and write transactions. The data width is set to 32 bits, and burst transactions are not supported.

6.8.2. Avalon Packets to Transactions Converter IP Operation

The Avalon Packets to Transactions Converter IP receives streams of packets on its Avalon streaming sink interface and initiates Avalon memory mapped transactions. Upon receiving transaction responses from Avalon memory mapped agents, the IP transforms the responses to packets and returns them to the requesting components via its Avalon streaming source interface. The IP does not report Avalon streaming errors.

6.8.2.1. Avalon Packets to Transactions Converter IP Data Packet Formats

A response packet is returned for every write transaction. The IP also returns a response packet if a no transaction (`0x7f`) is received. An invalid transaction code is regarded as a no transaction. For read transactions, the IP returns the data read.

The Avalon Packets to Transactions Converter IP expects incoming data streams to be in the formats shown in the table below.

Table 191. Data Packet Formats

Byte	Field	Description
Transaction Packet Format		
0	`Transaction code`	Type of transaction.
1	Reserved	Reserved for future use.
[3:2]	`Size`	Transaction size in bytes. For write transactions, the size indicates the size of the `data` field. For read transactions, the size indicates the total number of bytes to read.
[7:4]	`Address`	32-bit address for the transaction.
[n:8]	`Data`	Transaction data; data to be written for write transactions.
Response Packet Format		
0	`Transaction code`	The transaction code with the most significant bit inversed.
1	Reserved	Reserved for future use.
[4:2]	`Size`	Total number of bytes read/written successfully.

Related Information

Avalon Packets to Transactions Converter IP Interfaces on page 426

6.8.2.2. Avalon Packets to Transactions Converter IP Supported Transactions

The Avalon Packets to Transactions Converter IP supports the following Avalon memory mapped transactions:

Table 192. Avalon Packets to Transactions Converter IP Supported Transactions

Transaction Code	AvalonMemory Mapped Transaction	Description
`0x00`	Write, non-incrementing address.	Writes data to the address until the total number of bytes written to the same word address equals to the value specified in the `size` field.
`0x04`	Write, incrementing address.	Writes transaction data starting at the current address.
`0x10`	Read, non-incrementing address.	Reads 32 bits of data from the address until the total number of bytes read from the same address equals to the value specified in the `size` field.
`0x14`	Read, incrementing address.	Reads the number of bytes specified in the `size` parameter starting from the current address.
`0x7f`	No transaction.	No transaction is initiated. You can use this transaction type for testing purposes. Although no transaction is initiated on the Avalon memory mapped interface, the IP still returns a response packet for this transaction code.

The Avalon Packets to Transactions Converter IP can process only a single transaction at a time. The `ready` signal on the IP Avalon streaming sink interface is asserted only when the current transaction is completely processed.

No internal buffer is implemented on the datapaths. Data received on the Avalon streaming interface is forwarded directly to the Avalon memory mapped interface and vice-versa. Asserting the `waitrequest` signal on the Avalon memory mapped interface backpressures the Avalon streaming sink interface. In the opposite direction, if the Avalon streaming source interface is backpressured, the `read` signal on the Avalon memory mapped interface is not asserted until the backpressure is alleviated. Backpressuring the Avalon streaming source in the middle of a read can result in data loss. In this cases, the IP returns the data that is successfully received.

A transaction is considered complete when the IP receives an EOP. For write transactions, the actual data size is expected to be the same as the value of the `size` property. Whether or not both values agree, the IP always uses the end of packet (EOP) to determine the end of data.

6.8.2.3. Avalon Packets to Transactions IP Converter Malformed Packets

The following are examples of malformed packets:

- **Consecutive start of packet (SOP)**—An SOP marks the beginning of a transaction. If an SOP is received in the middle of a transaction, the IP drops the current transaction without returning a response packet for the transaction, and initiates a new transaction. This effectively processes packets without an end of packet (EOP).
- **Unsupported transaction codes**—The IP processes unsupported transactions as a no transaction.

6.9. Avalon Streaming Pipeline Stage Intel FPGA IP

The Avalon Streaming Pipeline Stage Intel FPGA IP receives data from an Avalon streaming source interface, and outputs the data to an Avalon streaming sink interface. In the absence of back pressure, the Avalon Streaming Pipeline Stage Intel FPGA IP source interface outputs data one cycle after receiving the data on its sink interface.

If the pipeline stage receives back pressure on its source interface, the pipeline stage continues to assert its source interface's current data output. While the pipeline stage is receiving back pressure on its source interface, and then receives new data on its sink interface, the pipeline stage internally buffers the new data. It then asserts back pressure on its sink interface.

After the backpressure is deasserted, the pipeline stage's source interface is deasserted and the pipeline stage asserts internally buffered data (if present). Additionally, the pipeline stage deasserts back pressure on its sink interface.

Figure 278. Pipeline Stage Simple Register

If the ready signal is not pipelined, the pipeline stage becomes a simple register.

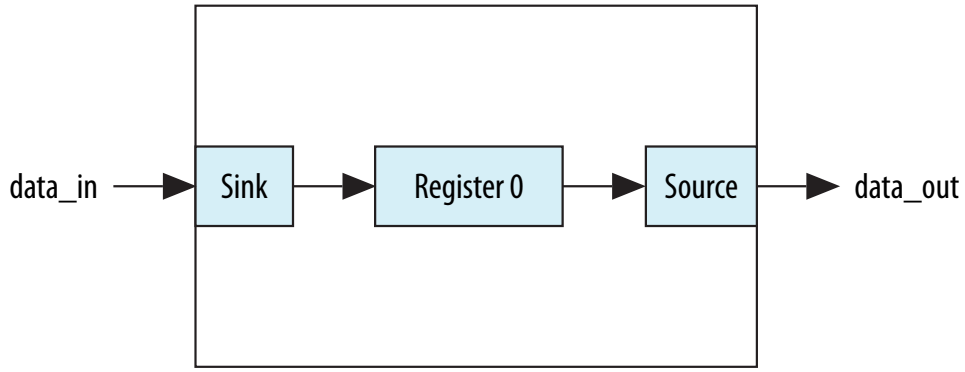
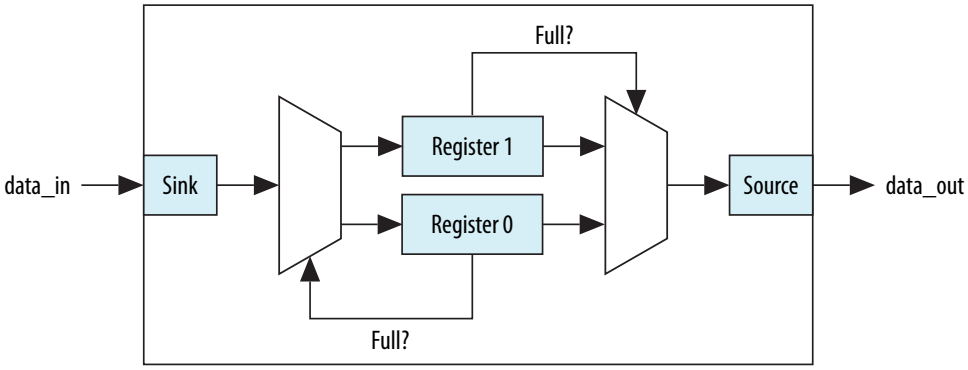


Figure 279. Pipeline Stage Holding Register

If the ready signal is pipelined, the pipeline stage must also include a second "holding" register.



6.10. Avalon Streaming Multiplexer and Demultiplexer Intel FPGA IP

The Avalon Streaming Multiplexer Intel FPGA IP receives data from various input interfaces and multiplexes the data into a single output interface, using the optional `channel` signal to indicate the origin of the data. The Avalon Streaming Multiplexer Intel FPGA IP receives data from a channelized input interface and drives that data to multiple output interfaces, where the output interface is selected by the input `channel` signal.

The Multiplexer and Demultiplexer IPs can transfer data between interfaces on that supports a unidirectional flow of data. The Multiplexer and Demultiplexer IP allow you to create multiplexed or demultiplexed datapaths without having to write custom HDL code. The Multiplexer IP includes an Avalon Streaming Round Robin Scheduler.

Related Information

Avalon Streaming Round Robin Scheduler Intel FPGA IP on page 423

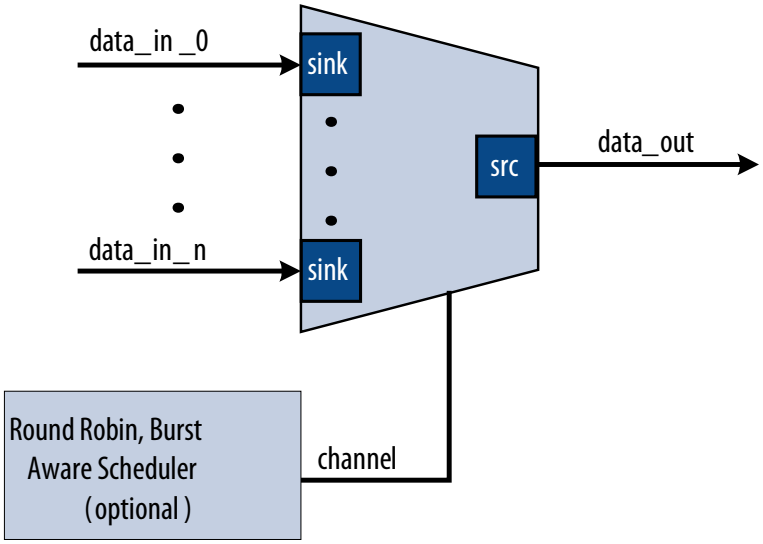
6.10.1. Avalon Streaming Multiplexer and Demultiplexer Software Programming Model

The Multiplexer and Demultiplexer IP components do not have any user-visible control or status registers. Therefore, Platform Designer cannot control or configure any aspect of the Multiplexer or Demultiplexer at run-time. These IP components cannot generate interrupts.

6.10.2. Avalon Streaming Multiplexer Intel FPGA IP

The Avalon Streaming Multiplexer Intel FPGA IP takes data from a variety of input data interfaces, and multiplexes the data onto a single output interface. The multiplexer includes a round-robin scheduler that selects from the next input interface that has data. Each input interface has the same width as the output interface, so that the other input interfaces are backpressured when the multiplexer is carrying data from a different input interface.

Figure 280. Avalon Streaming Multiplexer Intel FPGA IP



The multiplexer includes an optional channel signal that enables each input interface to carry channelized data. The output interface channel width is equal to:

(log2 (n-1)) + 1 + w

where n is the number of input interfaces, and w is the channel width of each input interface. All input interfaces must have the same channel width. These bits are appended to either the most or least significant bits of the output channel signal.

The scheduler processes one input interface at a time, selecting it for transfer. Once an input interface has been selected, data from that input interface is sent until one of the following scenarios occurs:

- The specified number of cycles have elapsed.
- The input interface has no more data to send and the `valid` signal is deasserted on a ready cycle.
- When packets are supported, `endofpacket` is asserted.

6.10.2.1. Avalon Streaming Multiplexer IP Input Interfaces

Each input interface is an Avalon streaming data interface that optionally supports packets. The input interfaces are identical; they have the same symbol and data widths, error widths, and channel widths.

6.10.2.2. Avalon Multiplexer IP Output Interface

The output interface carries the multiplexed data stream with data from the inputs. The symbol, data, and error widths are the same as the input interfaces.

The width of the `channel` signal is the same as the input interfaces, with the addition of the bits needed to indicate the origin of the data.

You can configure the following parameters for the output interface:

- **Data Bits Per Symbol**—the bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—the number of symbols (words) that are transferred per beat (transfer). Valid values are 1 to 32.
- **Include Packet Support**—indicates whether packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Channel Signal Width (bits)**— the number of bits Platform Designer uses for the channel signal for output interfaces. For example, set this parameter to 1 if you have two input interfaces with no channel, or set this parameter to 2 if you have two input interfaces with a channel width of 1 bit. The input channel can have a width between 0-31 bits.
- **Error Signal Width (bits)**—The width of the `error` signal for input and output interfaces. A value of `0` means the `error` signal is not in use.

Note: If you change only bits per symbol, and do not change the data width, errors are generated.

6.10.2.3. Avalon Multiplexer IP Parameters

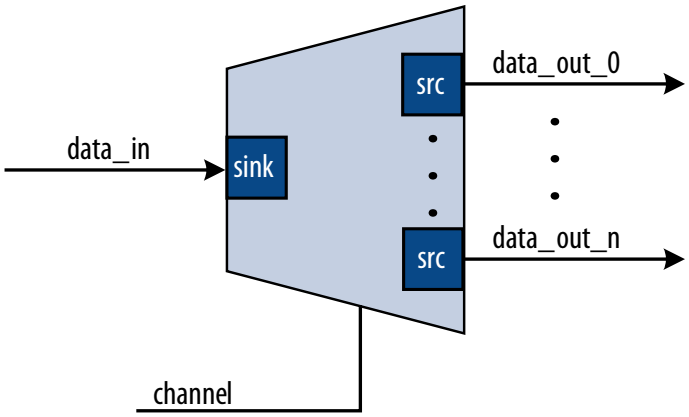
You can configure the following parameters for the multiplexer:

- **Number of Input Ports**—the number of input interfaces that the multiplexer supports. Valid values are 2 to 16.
- **Scheduling Size (Cycles)**—the number of cycles that are sent from a single channel before changing to the next channel.
- **Use Packet Scheduling**—when this parameter is turned on, the multiplexer only switches the selected input interface on packet boundaries. Therefore, packets on the output interface are not interleaved.
- **Use high bits to indicate source port**—when this parameter is turned on, the multiplexer uses the high bits of the output `channel` signal to indicate the origin of the input interface of the data. For example, if the input interfaces have 4-bit channel signals, and the multiplexer has 4 input interfaces, the output interface has a 6-bit channel signal. If this parameter is turned on, bits [5:4] of the output channel signal indicate origin of the input interface of the data, and bits [3:0] are the channel bits that were presented at the input interface.

6.10.3. Avalon Streaming Demultiplexer Intel FPGA IP

The Avalon Streaming Demultiplexer Intel FPGA IP takes data from a channelized input data interface and provides that data to multiple output interfaces, where the output interface selected for a particular transfer is specified by the input `channel` signal.

Figure 281. Avalon Streaming Demultiplexer



The data is delivered to the output interfaces in the same order it is received at the input interface, regardless of the value of `channel`, `packet`, `frame`, or any other signal. Each of the output interfaces has the same width as the input interface; each output interface is idle when the demultiplexer is driving data to a different output interface. The demultiplexer uses $\log_2$ (`num_output_interfaces`) bits of the `channel` signal to select the output for the data; the remainder of the channel bits are forwarded to the appropriate output interface unchanged.

6.10.3.1. Avalon Streaming Demultiplexer IP Input Interface

Each input interface is an Avalon streaming data interface that optionally supports packets. You can configure the following parameters for the input interface:

- **Data Bits Per Symbol**—The bits per symbol is related to the width of `readdata` and `writedata` signals, which must be a multiple of the bits per symbol.
- **Data Symbols Per Beat**—The number of symbols (words) that are transferred per beat (transfer). Valid values are 1 to 32.
- **Include Packet Support**—Indicates whether data packet transfers are supported. Packet support includes the `startofpacket`, `endofpacket`, and `empty` signals.
- **Channel Signal Width (bits)**—The number of bits for the `channel` signal for output interfaces. A value of `0` means that output interfaces do not use the optional `channel` signal.
- **Error Signal Width (bits)**—The width of the `error` signal for input and output interfaces. A value of `0` means the `error` signal is in use.

Note: If you change only bits per symbol, and do not change the data width, errors are generated.

6.10.3.2. Avalon Streaming Demultiplexer IP Output Interface

Each output interface carries data from a subset of channels from the input interface. Each output interface is identical; all have the same symbol and data widths, error widths, and channel widths. The symbol, data, and error widths are the same as the input interface. The width of the `channel` signal is the same as the input interface, without the bits that the demultiplexer uses to select the output interface.

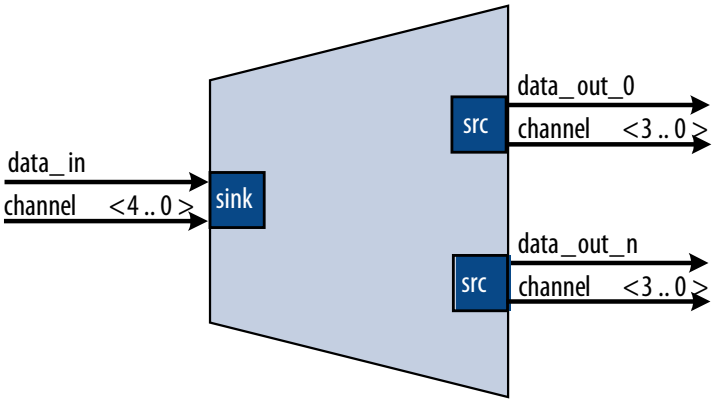
6.10.3.3. Avalon Streaming Demultiplexer IP Parameters

You can configure the following parameters for the demultiplexer:

- **Number of Output Ports**—The number of output interfaces that the multiplexer supports Valid values are 2 to 16.
- **High channel bits select output**—When this option is turned on, the demultiplexing function uses the high bits of the input `channel` signal, and the low order bits are passed to the output. When this option is turned off, the demultiplexing function uses the low order bits, and the high order bits are passed to the output.

Where you place the signals in your design affects the functionality; for example, there is one input interface and two output interfaces. If the low-order bits of the channel signal select the output interfaces, the even channels go to channel 0, and the odd channels go to channel 1. If the high-order bits of the channel signal select the output interface, channels 0 to 7 go to channel 0 and channels 8 to 15 go to channel 1.

Figure 282. Select Bits for the Demultiplexer



6.11. Avalon Streaming Single-Clock and Dual-Clock FIFO Intel FPGA IP

The Avalon Streaming Single-Clock and Avalon Streaming Dual-Clock FIFO Intel FPGA IP are FIFO buffers which operate with a common clock and independent clocks for input and output ports respectively.

Figure 283. Avalon Streaming Single Clock FIFO Intel FPGA IP

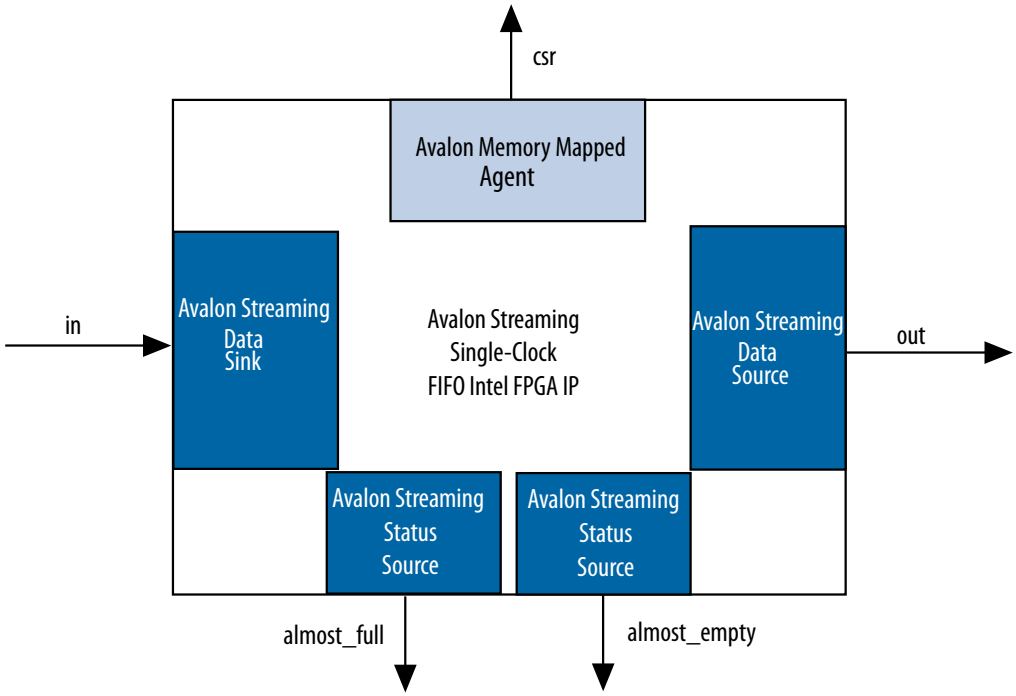
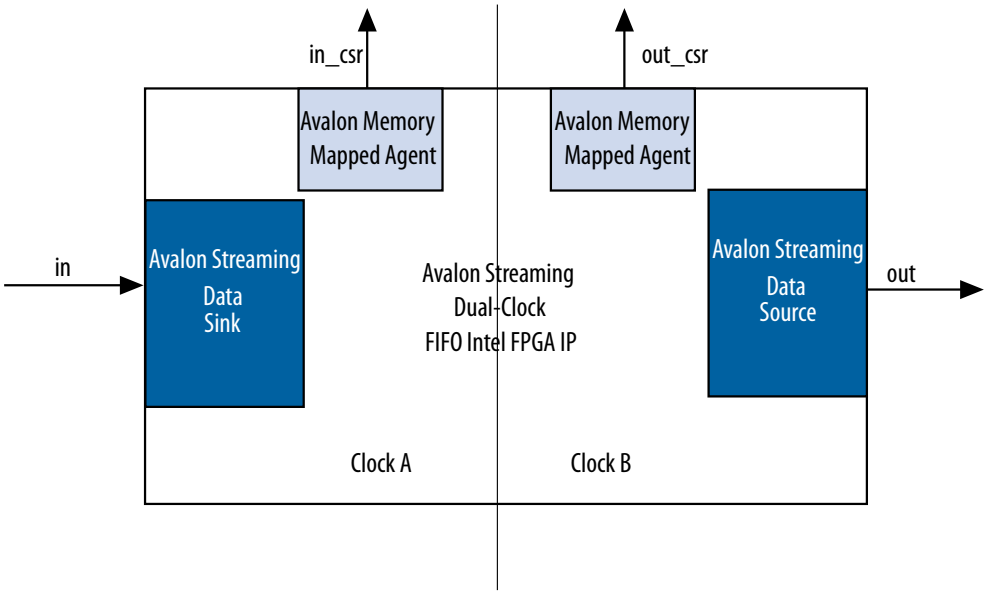


Figure 284. Avalon Streaming Dual Clock FIFO Intel FPGA IP



6.11.1. Interfaces Implemented in FIFO Cores

The following interfaces are implemented in FIFO Intel FPGA IP:

Avalon Streaming Data Interface on page 435

Avalon Memory Mapped Control and Status Register Interface on page 435
Avalon Streaming Status Interface on page 435

6.11.1.1. Avalon Streaming Data Interface

Each FIFO IP has an Avalon Streaming data sink and source interface. The data sink and source interfaces in the dual-clock FIFO IP are driven by different clocks.

Table 193. Avalon Streaming Interfaces Properties

Feature	Property
Backpressure	Ready latency = 0.
Data Width	Configurable.
Channel	Supported, up to 255 channels.
Error	Configurable.
Packet	Configurable.

6.11.1.2. Avalon Memory Mapped Control and Status Register Interface

You can configure the single-clock FIFO IP to include an optional Avalon memory mapped interface, and the dual-clock FIFO IP to include an Avalon memory mapped interface in each clock domain. The Avalon memory mapped interface provides access to 32-bit registers, which allows you to retrieve the FIFO buffer fill level and configure the almost-empty and almost-full thresholds. In the single-clock FIFO IP, you can also configure the packet and error handling modes.

6.11.1.3. Avalon Streaming Status Interface

The single-clock FIFO IP has two optional Avalon streaming status source interfaces from which you can obtain the FIFO buffer almost-full and almost empty statuses.

6.11.2. Avalon Streaming FIFO IP Operating Modes

- **Default mode**—the IP accepts incoming data on the `in` interface (Avalon streaming data sink) and forwards it to the `out` interface (Avalon streaming data source). The IP asserts the `valid` signal on the Avalon streaming source interface to indicate that data is available at the interface.
- **Store and forward mode**—this mode applies only to the single-clock FIFO IP. The IP asserts the `valid` signal on the `out` interface only when a full packet of data is available at the interface. In this mode, you can also enable the drop-on-error feature by setting the `drop_on_error` register to 1. When this feature is enabled, the IP drops all packets received with the `in_error` signal asserted.
- **Cut-through mode**—this mode applies only to the single-clock FIFO IP. The IP asserts the `valid` signal on the `out` interface to indicate that data is available for consumption when the number of entries specified in the `cut_through_threshold` register are available in the FIFO buffer.

Note: To turn on **Cut-through mode**, the **Use store and forward** parameter must be set to 0. Turning on **Use store and forward mode** prompts the user to turn on **Use fill level**, and then the CSR appears.

6.11.3. Avalon Streaming FIFO IP Buffer Fill Level

You can obtain the fill level of the Avalon Streaming FIFO IP buffer via the optional Avalon memory mapped control and status interface. Turn on the **Use fill level** parameter (**Use sink fill level** and **Use source fill level** in the Avalon Streaming Dual-Clock FIFO IP) and read the `fill_level` register.

The Avalon Streaming Dual-Clock FIFO IP has two fill levels, one in each clock domain. Due to the latency of the clock crossing logic, the fill levels reported in the input and output clock domains may be different for any instance. In both cases, the fill level may report badly for the clock domain; that is, the fill level is reported high in the input clock domain, and low in the output clock domain.

The Avalon Streaming Dual-Clock FIFO IP has an output pipeline stage to improve f_{MAX}. This output stage is accounted for when calculating the output fill level, but not when calculating the input fill level. Therefore, the best measure of the amount of data in the FIFO is by the fill level in the output clock domain. The fill level in the input clock domain represents the amount of space available in the FIFO (`available space = FIFO depth – input fill level`).

6.11.4. Almost-Full and Almost-Empty Thresholds to Prevent Overflow and Underflow

You can use almost-full and almost-empty thresholds as a mechanism to prevent FIFO IP overflow and underflow. This feature is available only in the Avalon Streaming Single-Clock FIFO IP. To use the thresholds, turn on the **Use fill level**, **Use almost-full status**, and **Use almost-empty status** parameters. You can access the `almost_full_threshold` and `almost_empty_threshold` registers via the csr interface and set the registers to an optimal value for your application.

You can obtain the almost-full and almost-empty statuses from `almost_full` and `almost_empty` interfaces (Avalon streaming status source). The IP asserts the `almost_full` signal when the fill level is equal to or higher than the almost-full threshold. Likewise, the IP asserts the `almost_empty` signal when the fill level is equal to or lower than the almost-empty threshold.

6.11.5. Avalon Streaming Single Clock and Dual Clock FIFO IP Parameters

Table 194. Avalon Streaming Single Clock FIFO IP Parameters

Parameter	Legal Values	Description
Symbols per beat	1–32	These parameters determine the width of the FIFO. FIFO width = **Bits per symbol** * **Symbols per beat**, where: **Bits per symbol** is the number of bits in a symbol, and **Symbols per beat** is the number of symbols transferred in a beat.
Bits per symbol	1–32	
FIFO depth	2^n	The FIFO depth. An output pipeline stage is added to the FIFO to increase performance, which increases the FIFO depth by one. <n> = n=1,2,3,4 and so on.
Channel width	1–32	The width of the `channel` signal.
Error width	0–32	The width of the `error` signal.

continued...

Parameter	Legal Values	Description
Use packets	On|Off	Turn on this parameter to enable data packet support on the Avalon streaming data interfaces.
Use fill level	On|Off	Turn on this parameter to include the Avalon memory mapped control and status register interface (CSR). The CSR is enabled when **Use fill level** is set to 1.
Use store and forward	On|Off	To turn on **Cut-through mode**, **Use store and forward** must be set to 0. Turning on **Use store and forward** prompts the user to turn on **Use fill level**, and then the CSR appears.
Use almost full status	On|Off	Enables a single-bit almost-full status streaming interface
Use almost empty status	On|Off	Enables a single-bit almost-empty status streaming interface.
Enable explicit maxChannel	On|Off	Turn on this parameter to specify the maximum channel number.
Explicit maxChannel	value	Maximum channel number.
Use synchronous resets	On|Off	Turing off allows asynchronous resets. Turning on uses internal reset synchronization.

Table 195. Avalon Streaming Dual Clock FIFO IP Parameters

Parameter	Legal Values	Description
Symbols per beat	1–32	These parameters determine the width of the FIFO. FIFO width = **Bits per symbol** * **Symbols per beat**, where: **Bits per symbol** is the number of bits in a symbol, and **Symbols per beat** is the number of symbols transferred in a beat.
Bits per symbol	1–32	
FIFO depth	2^n	The FIFO depth. An output pipeline stage is added to the FIFO to increase performance, which increases the FIFO depth by one. <n> = n=1,2,3,4 and so on.
Channel width	1–32	The width of the `channel` signal.
Error width	0–32	The width of the `error` signal.
Use packets	On|Off	Turn on this parameter to enable data packet support on the Avalon streaming data interfaces.
Use sink fill level	On|Off	Turn on this parameter to include the Avalon memory mapped control and status register interface in the input clock domain.
Use source fill level	On|Off	Turn on this parameter to include the Avalon memory mapped control and status register interface in the output clock domain.
Write pointer synchronizer length	2–8	The length of the write pointer synchronizer chain. Setting this parameter to a higher value leads to better metastability while increasing the latency of the IP.
Read pointer synchronizer length	2–8	The length of the read pointer synchronizer chain. Setting this parameter to a higher value leads to better metastability.
Enable explicit maxChannel	On|Off	Turn on this parameter to specify the maximum channel number.
Explicit maxChannel	value	Maximum channel number.
Pipeline pointers	On|Off	This option enables the pipeline pointer after clock domain crossing. Enable this option for better timing closure by adding one clock cycle of latency.
Use synchronous resets	On|Off	Turing off allows asynchronous resets. Turning on uses internal reset synchronization.

Note: For more information about metastability in Intel devices, refer to *Understanding Metastability in FPGAs*. For more information about metastability analysis and synchronization register chains, refer to the *Managing Metastability* in the *Quartus Prime Pro Edition User Guide: Design Recommendations*.

6.11.6. Avalon Streaming Single-Clock FIFO IP Registers

Table 196. Avalon Streaming Single-Clock FIFO IP Registers

The CSR interface in the Avalon Streaming Single Clock FIFO IP provides access to registers.

32-Bit Word Offset	Name	Access	Reset	Description
0	`fill_level`	R	0	24-bit FIFO fill level. Bits 24 to 31 are not used.
1	Reserved	—	—	Reserved for future use.
2	`almost_full_threshold`	RW	**FIFO depth**–1	Set this register to a value that indicates the FIFO buffer is getting full.
3	`almost_empty_threshold`	RW	0	Set this register to a value that indicates the FIFO buffer is getting empty.
4	`cut_through_threshold`	RW	0	**0**—Enables store and forward mode. **Greater than 0**—Enables cut-through mode and specifies the minimum of entries in the FIFO buffer before the `valid` signal on the Avalon streaming source interface is asserted. Once the FIFO IP starts sending the data to the downstream component, it continues to do so until the end of the packet. *Note:* To turn on **Cut-through mode**, **Use store and forward** must be set to 0. Turning on **Use store and forward mode** prompts the user to turn on **Use fill level**, and then the CSR appears.
5	`drop_on_error`	RW	0	**0**—Disables drop-on error. **1**—Enables drop-on error. This register applies only when the **Use packet** and **Use store and forward** parameters are turned on.

Table 197. Register Description for Avalon Streaming Dual-Clock FIFO

The `in_csr` and `out_csr` interfaces in the Avalon Streaming Dual Clock FIFO IP reports the FIFO fill level.

32-Bit Word Offset	Name	Access	Reset Value	Description
0	`fill_level`	R	0	24-bit FIFO fill level. Bits 24 to 31 are not used.

Related Information

Avalon Interface Specifications

6.12. Platform Designer System Design Components Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2023.04.03	23.1	• Added note about no support for fixed-latency pipelined read transfers to *Avalon Memory Mapped Pipeline Bridge Intel FPGA IP* topic. • The product family name is updated to "Intel Agilex 7" to reflect the different family members.
2022.09.26	22.3	• Revised note about the default agent selection to *Designating a Default Agent* topic. • Revised note about the default agent selection to *Specifying a Default Avalon Agent or AXI Subordinate* topic. • Revised note about the default agent selection to *Accessing Undefined Memory Regions* topic.
2022.04.02	22.1	• Updated entire chapter for new AXI "manager" and AXI "subordinate" replacement terms. Refer to the AMBA® AXI and ACE Protocol Specification. • Updated *AXI Timeout Bridge Stages* and *AXI Timeout Bridge Parameters* topics for new CSR behavior and parameters. • Updated the *Avalon Streaming Single-Clock and Dual-Clock FIFO IP Parameters* topic for Pipeline pointers parameter.
2021.03.29	21.1	• Converted to "host" and "agent" inclusive terminology for Avalon memory mapped interface descriptions and related GUI elements throughout.
2021.01.14	20.4	• Added descriptions to *CSR Interrupt Status Information for the AXI Timeout Bridge* table.
2019.11.11	19.1	• Updated the names of Intel FPGA IP components throughout. • Updated name of Test Pattern Checker IP to Avalon Data Pattern Checker IP throughout. • Updated Address Span Extender figure bit order. • Provided directory path in Test Pattern Generator
2018.12.15	18.1	• Replaced references to System Contents tab with new System View tab.
2017.11.06	17.1	• Changed instances of *Qsys Pro* to *Platform Designer*. • Changed instances of *AXI Default Slave* to *Error Response Slave*. • Updated topics: Error Response Slave. • Updated Figure: Error Response Slave Parameter Editor. • Added Figure: Error Response Slave Parameter Editor with Enabled CSR Support. • Updated topics: CSR Registers and renamed to Error Response Slave CSR Registers. • Added topic: Error Response Slave Access Violation Service.
2016.10.31	16.1	• Implemented Intel rebranding. • Implemented Qsys rebranding.
2016.05.03	16.0	Updated Address Span Extender • Address Span Extender register mapping better explained • Address Span Extender Parameters table added • Address Span Extender example added
2015.11.02	15.1	Changed instances of *Quartus II* to *Quartus Prime*.

continued...

Document Version	Quartus Prime Version	Changes
2015.05.04	15.0	Avalon Memory Mapped Unaligned Burst Expansion Bridge and Avalon Memory Mapped Pipeline Bridge, **Maximum pending read transactions** parameter. Extended description.
December 2014	14.1	• AXI Timeout Bridge. • Added notes to *Avalon Memory Mapped Clock Crossing Bridge* pertaining to: — SDC constraints for its internal asynchronous FIFOs. — FIFO-based clock crossing.
June 2014	14.0	• AXI Bridge support. • Address Span Extender updates. • Avalon Memory Mapped Unaligned Burst Expansion Bridge support.
November 2013	13.1	• Address Span Extender
May 2013	13.0	• Added Streaming Pipeline Stage support. • Added AMBA APB support.
November 2012	12.1	• Moved relevant content from the *Embedded Peripherals IP User Guide*.

7. Platform Designer Command-Line Utilities

You can perform many of the functions available in the Platform Designer GUI at the command-line, with Platform Designer command-line utilities.

You run Platform Designer command-line executables from the Quartus Prime installation directory:

```
<Quartus Prime installation directory>\quartus\sopc_builder\bin
```

For command-line help listing of all the options for any executable, type the following command:

```
<Quartus Prime installation directory>\quartus\sopc_builder\bin
\<executable name> --help
```

Note: You must add `$QUARTUS_ROOTDIR/sopc_builder/bin/` to the PATH variable to access command-line utilities. Once you add this PATH variable, you can launch the utility from any directory location.

7.1. Run the Platform Designer Editor with qsys-edit

The `qsys-edit` utility allows you to run the Platform Designer editor from command-line.

You can use the following options with the `qsys-edit` utility:

Table 198. qsys-edit Command-Line Options

Option	Usage	Description
1st arg file	Optional	Specifies the name of the `.qsys` system or `.qvar` variation file to edit.
`--search-path[=<value>]`	Optional	If you omit this command, Platform Designer uses a standard default path. If you provide a search path, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example: `/extra/dir,$` .
`--quartus-project[=<value>]`	Required	This option is mandatory if you are associating your Platform Designer system with an existing Quartus Prime project. Specifies the name of the Quartus Prime project file. If you do not provide the revision via `--rev`, Platform Designer uses the default revision as the Quartus Prime project name.

continued...

Option	Usage	Description
`--new-quartus-project[=<value>]`	Required	This option is mandatory if you are associating your Platform Designer system with a new Quartus Prime project. Specifies the name and path of the new Quartus Prime project. Creates a new Quartus Prime project at the specified path. You can also provide the revision name.
`--rev[=<value>]`	Optional	Specifies the name of the Quartus Prime project revision.
`--family[=<value>]`	Optional	Sets the device family.
`--part[=<value>]`	Optional	Sets the device part number. If set, this option overrides the `--family` option.
`--new-component-type[=<value>]`	Optional	Specifies the instance type for parameterization in a variation.
`--require-generation`	Optional	Marks the loading system as requiring generation.
`--debug`	Optional	Enables debugging features and output.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that Platform Designer uses when running `qsys-edit`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--metrics-log-file=[path_to_log_file]`	Optional	This switch creates a new file containing a set of metrics, at the path you specify. This switch must be used alongside the `--record-metrics` switch, or the switch does not function. The path to the log file must exist, or the `FileDoesNotExists` error appears. Platform Designer does not create any new directories listed in the path.
`--record-metrics[=<foobar>]`	Optional	Enables metric logging. This feature logs the time Platform Designer takes to perform various operations. The other information includes: • The operation performed. • The IP instance name and its `.ip` file location. • The IP component type and its `_hw.tcl` file location. The switch does not require a value, but it does accept a string value that has no effect on the metric logging. In the absence of the `--metrics-log-file` switch, Platform Designer creates the log file in the project directory with the name of `metrics_<top-level name>.txt` by default.
`--help`	Optional	Displays help for `qsys-edit`.

Important: The options `--quartus-project` and `--new-quartus-project` are mutually exclusive. If you use `--quartus-project` you cannot use `--new-quartus-project` and vice versa.

Extended Features with the --debug Options

The `--debug` option provides powerful tools for debugging. When you launch Platform Designer with the `--debug` option enabled, you can:

- View debug messages when opening a system or generating HDL for that system.
- Add the `--verbose` argument when generating IP or a system using command-line utilities.
- Access internal library components in the IP Catalog, for example, modules used to create interconnect fabric.
- Access to debug tools and files from the **Internal** menu.

Figure 285. Internal Menu Options

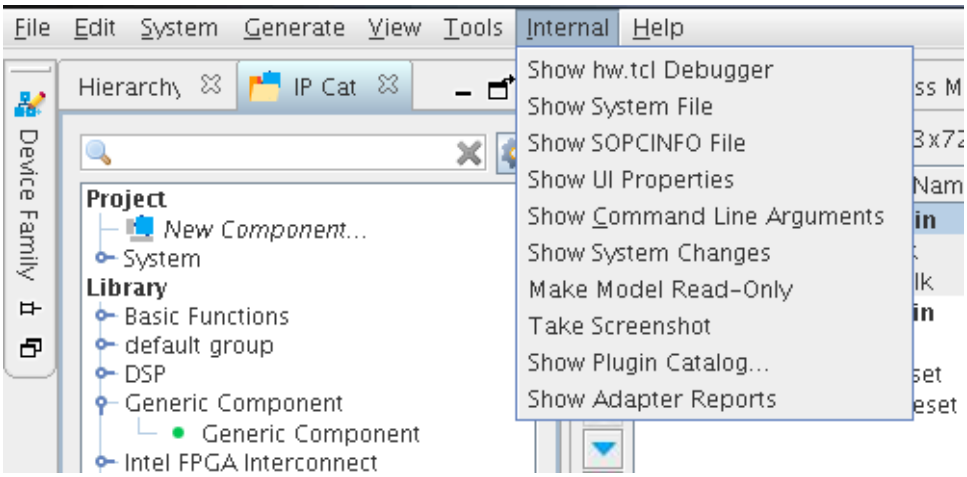


Table 199. Debug Options on the Internal Menu

Menu Item	Description
Show hw.tcl Debugger	Displays a Tcl debugger.
Show System File	Displays the current system XML in a text dialog box.
Show SOPCINFO File	Shows the SOPCINFO report XML in a text dialog box.
Show UI Properties	Displays the UI properties in a text dialog box.
Show Command Line Arguments	Displays all command-line arguments and environment variables in a text dialog box.
Show System Changes	Displays dynamic system changes in a text dialog box.
Make Model Read-only	Makes the system you are working in read-only.
Take Screenshots	Creates a `.png` file in the *<project_directory>* by default. You can navigate and save to a directory of your choice.
Show Plug-In Catalog	Displays library details such as type, version, tags, etc. for all IPs in the IP Catalog.
Show Adapter Reports	Displays adapter reports for any adapters added when transforming the system.

- You can view detailed debugging messages in the **Component Editor** while building a custom IP component.
- You can view the generated Tcl script while editing in the **Component Editor** with the **Advanced ➤ Show Tcl for Component** command.
- You can launch the System Console with debug logging.

7.2. Scripting IP Core Generation

Use the `qsys-script` and `qsys-generate` utilities to define and generate an IP core variation outside of the Quartus Prime GUI.

To parameterize and generate an IP core at command-line, follow these steps:

1. Run `qsys-script` to start a Tcl script that instantiates the IP and sets parameters:

```
qsys-script --script=<script_file>.tcl
```

2. Run `qsys-generate` to generate the IP core variation:

```
qsys-generate <IP variation file>.qsys
```

Related Information

Generate a Platform Designer System with qsys-script on page 451

7.2.1. qsys-generate Command-Line Options

Table 200. Command-Line Options for qsys-generate

Options in alphabetical order.

Option	Usage	Description
<1st arg file>	Required	Specifies the name of the `.qsys` system file to generate.
`--block-symbol-file`	Optional	Creates a Block Symbol File (`.bsf`) for the Platform Designer system.
`--bypass-quartus-project`	Optional	Override the project settings with values from the Platform Designer system, if possible. For example, if the Platform Designer system defines its own IP search path, use that path instead of the project settings IP search path. This option mainly affects the generation process.
`--clear-output-directory`	Optional	Clears the output directory corresponding to the selected target, that is, simulation or synthesis.
`--example-design=<value>`	Optional	Creates example design files. For example, `--example-design` or `--example-design=all`. The default is *All*, which generates example designs for all instances. Alternatively, choose specific filesets based on instance name and fileset name. For example `--example-design=instance0.example_design1,instance1.example_design` 2. Specify an output directory for the example design files creation.
`--family=<value>`	Optional	Sets the device family name.
`--help`	Optional	Displays help for `--qsys-generate`.

continued...

Option	Usage	Description
`--greybox`	Optional	If you are synthesizing your design with a third-party EDA synthesis tool, generate a netlist for the synthesis tool to estimate timing and resource usage for this design. *Note:* Generation of a timing and area estimation (gray box) netlist is available only for individual Intel FPGA IP, and not for Platform Designer systems.
`--ipxact`	Optional	If you specify this option, Platform Designer generates the post-generation system as an IPXACT-compatible component description. *Note:* Platform Designer supports importing and exporting files in IP-XACT 2009 format and exporting IP-XACT files in 2014 format.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that Platform Designer uses when running `qsys-generate`. You specify the value as `<size><unit>`, where `unit` is m (or M) for multiples of megabytes or g (or G) for multiples of gigabytes. The default value is 512m.
`--metrics-log-file=[path_to_log_file]`	Optional	This switch creates a new file containing a set of metrics, at the path you specify. This switch must be used alongside the `--record-metrics` switch, or the switch does not function. The path to the log file must exist, or the `FileDoesNotExists` error appears. Platform Designer does not create any new directories listed in the path.
`--parallel[=<level>]`	Optional	Directs Platform Designer to generate in parallel mode, with the level of parallelism that you specify. If you omit the level, Platform Designer determines a number based on processor availability and number of files to be generated.
`--part=<value>`	Optional	Sets the device part number. If set, this option overrides the `--family` option.
`--record-metrics[=<foobar>]`	Optional	Enables metric logging. This feature logs the time Platform Designer takes to perform various operations. The other information includes: • The operation performed. • The IP instance name and its `.ip` file location. • The IP component type and its `_hw.tcl` file location. The switch does not require a value, but it does accept a string value that has no effect on the metric logging. In the absence of the `--metrics-log-file` switch, Platform Designer creates the log file in the project directory with the name of `metrics_<top-level name>.txt` by default.
`--search-path=<value>`	Optional	If you omit this command, Platform Designer uses a standard default path. If you provide this command, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example, "`/extra/dir,$`".
`--simulation=<VERILOG|VHDL>`	Optional	Creates a simulation model for the Platform Designer system. The simulation model contains generated HDL files for the simulator, and may include simulation-only features. Specify the preferred simulation language. The default value is *VERILOG*.
`--synthesis=<VERILOG|VHDL>`	Optional	Creates synthesis HDL files that Platform Designer uses to compile the system in an Quartus Prime project. Specify the generation language for the top-level RTL file for the Platform Designer system. The default value is *VERILOG*.

continued...

Option	Usage	Description
`--testbench=<SIMPLE|STANDARD>`	Optional	Creates a testbench system that instantiates the original system, adding bus functional models (BFMs) to drive the top-level interfaces. When you generate the system, the BFMs interact with the system in the simulator. The default value is *STANDARD*.
`--testbench-simulation=<VERILOG|VHDL>`	Optional	After you create the testbench system, create a simulation model for the testbench system. The default value is *VERILOG*.
`--top-level-generation`	Optional	Only generate the top-level of the given Platform Designer system. Do not descend into hierarchy. Do not generate generic components.
`--upgrade-ip-cores`	Optional	Enables upgrading all the IP cores that support upgrade in the Platform Designer system you specify. This command has no impact on IP cores in any subsystem.
`--upgrade-variation-file`	Optional	If you set this option to true, the file argument for this command accepts a `.v` file, which contains a IP variant. This file parameterizes a corresponding instance in a Platform Designer system of the same name.

Related Information

Generating Simulation Files for Platform Designer Systems and IP Variants on page 101

7.3. Board-Aware Flow Scripting Support

The following Tcl commands support the board-aware flow, including preset and board file management, and board and preset file export. For examples use of these commands, refer to Example 1 and Example 2.

Table 201. General Commands

Return	Command	Argument
No return value	`set_project_property`	`BOARD <board_name>`
String	`get_project_property`	`BOARD`

Table 202. Preset Commands

Return	Command	Argument
no return value	`save_component_preset`	`[<preset_file> <preset_name> <preset_description> <preset_version> <board_name> <category> <pin_file.tcl>]`
no return value	`apply_component_preset`	`<preset_name>`

Table 203. Board Commands

Return	Command	Argument
no return value	`add_board`	`<board_file> <board_name> <vendor> <device_family> <device_part> <version>`
String	`get_board_info`	`<board_name> <info_type>` <info_type> can be vendor, device_family, device_part, version
String	`get_boards`	`[<device_family> <device_part>]` Returns all board names.
no return value	`delete_board`	`<board_name>`

Table 204. Export Board and Presets from Current System Commands

Return	Command	Argument
no return value	`export_board_file`	*`<board_name> <board_file>`* `[<vendor> <version>]`
`String[] Instance_paths`	`get_quartus_instance_path_for_entity`	`[<entity_name>]` Returns a list of instance paths in hierarchical format. For example: • `quartus_top.qsys_instance_name` • `quartus_top.something.qsys_instance_name`
no return value	`load_pin_from_quartus_project`	*`<instance_path>`*
No return value	`export_system_preset`	*`[preset_directory_path]`* *Note:* By default the `preset_file_name` is `system_name.qprs`.

Example 1: Create System, Add Component, Apply Built-In Preset

```
# create the system
create_system test_system
set_project_property BOARD {Arria 10 SoC Development Kit}
set_project_property HIDE_FROM_IP_CATALOG {false}
set_use_testbench_naming_pattern 0 {}
# add component pio0
add_component pio_0 ip/test_system/test_system_pio_0.ip altera_avalon_pio pio_0
19.2.0
load_component pio_0

#assume this led4 preset file path is in the search path
apply_component_preset led4
save_component
```

Example 2: Create New Preset, Apply Preset with Pin Constraints File

Example of led4_pins.tcl file:

```
set_instance_assignment -to "led_export[0]" -name IO_STANDARD "1.8 V"
set_location_assignment -to "led_export[0]" "PIN_AR23"
set_instance_assignment -to "led_export[1]" -name IO_STANDARD "1.8 V"
set_location_assignment -to "led_export[1]" "PIN_AR22"
set_instance_assignment -to "led_export[2]" -name IO_STANDARD "1.8 V"
set_location_assignment -to "led_export[2]" "PIN_AM21"
set_instance_assignment -to "led_export[3]" -name IO_STANDARD "1.8 V"
set_location_assignment -to "led_export[3]" "PIN_AL20"
```

Script:

```
add_component pio_0 ip/sys/sys_pio_0.ip altera_avalon_pio pio_0 19.2.0
load_component pio_0
set_component_parameter_value bitClearingEdgeCapReg {0}
set_component_parameter_value bitModifyingOutReg {0}
set_component_parameter_value captureEdge {0}
set_component_parameter_value direction {Output}
set_component_parameter_value edgeType {RISING}
set_component_parameter_value generateIRQ {0}
set_component_parameter_value irqType {LEVEL}
set_component_parameter_value resetValue {0.0}
set_component_parameter_value simDoTestBenchWiring {0}
set_component_parameter_value simDrivenValue {0.0}
set_component_parameter_value width {4}
set_component_project_property HIDE_FROM_IP_CATALOG {false}
# export interface first, then only import the pin constraint tcl file
set_interface_property led EXPORT_OF pio_0.external_connection
save_component_preset led4_demo.qprs led4_demo "" ALL "Arria 10 SoC Development
Kit" "IO" led4_pins.tcl
apply_component_preset led4_demo
save_component
```

7.4. Display Available IP Components with ip-catalog

The `ip-catalog` command displays a list of available IP components relative to the current Quartus Prime project directory, as either text or XML.

You can use the following options with the `ip-catalog` utility:

Table 205. ip-catalog Command-Line Options

Option	Usage	Description
`--project-dir=<directory>`	Optional	Finds IP components relative to the Quartus Prime project directory. By default, Platform Designer uses `'.'` as the current directory. To exclude a project directory, leave the value empty.
`--type`	Optional	Provides a pattern to filter the type of available plug-ins. By default, Platform Designer shows only IP components. To look for a partial type string, surround with *, for instance, **connection**.
`--name=<value>`	Optional	Provides a pattern to filter the names of the IP components found. To show all IP components, use a * or `''`. By default, Platform Designer shows all IP components. The argument is not case sensitive. To look for a partial name, surround with *, for instance, **uart**
`--verbose`	Optional	Reports the progress of the command.
`--xml`	Optional	Generates the output in XML format, in place of colon-delimited format.
`--search-path=<value>`	Optional	If you omit this command, Platform Designer uses a standard default path. If you provide this command, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example, "`/extra/dir,$`".
<1st arg value>	Optional	Specifies the directory or name fragment.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that Platform Designer uses for when running `ip-catalog`. You specify the value as *<size><unit>*, where `unit` is m (or M) for multiples of megabytes or g (or G) for multiples of gigabytes. The default value is 512m.
`--help`	Optional	Displays help for the `ip-catalog` command.

7.5. Create an .ipx File with ip-make-ipx

The `ip-make-ipx` command creates an `.ipx` index file. This file provides a convenient way to include a collection of IP components from an arbitrary directory. You can edit the `.ipx` file to disable visibility of one or more IP components in the IP Catalog.

You can use the following options with the `ip-make-ipx` utility:

Table 206. ip-make-ipx Command-Line Options

Option	Usage	Description
`--source-directory=<directory>`	Optional	Specifies the directory containing your IP components. The default directory is `'.'`. You can provide a comma-separated list of directories.
`--output=<file>`	Optional	Specifies the name of the index file to generate. The default name is `/component.ipx`. Set as `--output=<"">` to print the output to the console.

continued...

Option	Usage	Description
`--relative-vars=<value>`	Optional	Causes the output file to include references relative to the specified variable or variables wherever possible. You can specify multiple variables as a comma-separated list.
`--thorough-descent`	Optional	If you set this option, Platform Designer searches all the component files, without skipping the sub-directories.
`--message-before=<value>`	Optional	Prints a log message at the start of reading an index file.
`--message-after=<value>`	Optional	Prints a log message at the end of reading an index file.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size Platform Designer uses when running `ipr-make-ipx`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Displays help for the `ip-make-ipx` command.

7.6. Generate Simulation Scripts

You can use the `ip-make-simscript` utility to generate simulation scripts for one or more simulators, given one or more **Simulation Package Descriptor** (`.spd`) files, `.qsys` files, and `.ip` files.

In Platform Designer, `ip-make-simscript` generates simulation scripts in a hierarchical structure instead of a flat view of the entire system. The `ip-make-simscript` utility uses `.spd` and system files according to the options you select:

- When targeting only `.spd` files (`ip-make-simscript --spd=<file>.spd`) the utility combines the contents of all input `.spd` files, and generates a common directory which contains a set of `<simulator>_files.tcl` files under the specified output directory.
- When targeting only system files (`ip-make-simscript --system-file=<file>`) such as `.qsys` and `.ip` files, the utility searches for instances of `<simulator>_files.tcl` files for each input system, and generates a combined simulation script which contains a list of references of `<simulator>_files.tcl`.
- When the utility uses both `--spd` and `--system-file` options, `ip-make-simscript` combines all input `.spd` files and generates a `common/<simulator>_files.tcl` in the specified output directory. The generated simulation script refers to the generated `common/<simulator>_files.tcl` first, followed by a list of Tcl files from each input system.

Table 207. ip-make-simscript Command-Line Options

Option	Usage	Description
`--spd[=<file>]`	Optional/Repeatable	The `.spd` files describe the list of HDL files for simulation, and memory models hierarchy. This argument can either be a single path to an `.spd` file or a comma-separated list of paths of `.spd` files. For instance, `--spd=ipcore_1.spd,ipcore_2.spd` The generated list is processed in the order of the input `.spd` files.

continued...

Option	Usage	Description
		Note: When this argument is used in combination with `--system-file`, the `.spd` files are parsed before the system files.
`--system-file[=<file>]`	Optional/Repeatable	Specifies the system files (`.qsys` or `.ip` files) used to generate the simulation scripts. This argument can contain either a single path to a Platform Designer system file or a comma-separated list of paths to Platform Designer system files. The simulation script is generated in the order the system files are listed. *Note:* When this argument is used in combination with `--spd`, the `.spd` files are parsed before the system files.
`--output-directory[=<directory>]`	Optional	Specifies the directory path for the location of output files. If you do not specify a directory, the output directory defaults to the directory from which `--ip-make-simscript` runs.
`--compile-to-work`	Optional	Compiles all design files to the default library - `work`.
`--use-relative-paths`	Optional	Uses relative paths whenever possible.
`--cache-file[=<file>]`	Optional	Generates cache file for managed flow.
`--quiet`	Optional	Quiet reporting mode. Does not report generated files.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size Platform Designer uses when running `ip-make-simscript`. You specify this value as `<size><unit>` where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--search-path=<value>`	Optional	Comma-separated list of search paths. If omitted, a default path including the current working directory is used. To include the standard path in your replacement, append the $ symbol, for example:`"/extra/dir,$"`
`--device-family=<value>`	Optional	Overrides the existing device family when used.
`--top-name=<value>`	Optional	Specify a top-level entity name used in generated simulation scripts.
`--help`	Optional	Displays help for `--ip-make-simscript`.

7.7. Generate a Platform Designer System with qsys-script

You can use the `qsys-script` utility to create and manipulate a Platform Designer system with Tcl scripting commands. If you specify a system, Platform Designer loads that system before executing any of the scripting commands.

Note: You must provide a package version for the `qsys-script`. If you do not specify the `--package-version=<value>` command, you must then provide a Tcl script and request the system scripting API directly with the `package require -exact qsys<version>` command.

Example 28. Platform Designer Command-Line Scripting

```
qsys-script --script=my_script.tcl \
--system-file=fancy.qsys
```

`my_script.tcl` contains:

```
package require -exact qsys 16.0
# get all instance names in the system and print one by one
set instances [ get_instances ]
foreach instance $instances {
    send_message Info "$instance"
}
```

You can use the following options with the `qsys-script` utility:

Table 208. qsys-script Command-Line Options

Option	Usage	Description
`--system-file=<file>`	Optional	Specifies the path to a `.qsys` file. Platform Designer loads the system before running scripting commands.
`--script=<file>`	Optional	A file that contains Tcl scripting commands that you can use to create or manipulate a Platform Designer system. If you specify both `--cmd` and `--script`, Platform Designer runs the `--cmd` commands before the script specified by `--script`.
`--cmd=<value>`	Optional	A string that contains Tcl scripting commands that you can use to create or manipulate a Platform Designer system. If you specify both `--cmd` and `--script`, Platform Designer runs the `--cmd` commands before the script specified by `--script`.
`--package-version=<value>`	Optional	Specifies which Tcl API scripting version to use and determines the functionality and behavior of the Tcl commands. The Quartus Prime software supports Tcl API scripting commands. The minimum supported version is 12.0. If you do not specify the version on the command-line, your script must request the scripting API directly with the `package require -exact qsys` <*version* > command.
`--search-path=<value>`	Optional	If you omit this command, a Platform Designer uses a standard default path. If you provide this command, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example, `/<directory path>/dir,$`. Separate multiple directory references with a comma.
`--quartus-project=<value>`	Optional	Specifies the path to a `.qpf` Quartus Prime project file. Utilizes the specified Quartus Prime project to add the file saved using `save_system` command. If you omit this command, Platform Designer uses the default revision as the project name.
`--new-quartus-project=<value>`	Optional	Specifies the name of the new Quartus Prime project. Creates a new Quartus Prime project at the specified path and adds the file saved using `save_system` command to the project. If you omit this command, Platform Designer uses the Quartus Prime project revision as the new Quartus Prime project name.
`--rev=<value>`	Optional	Allows you to specify the name of the Quartus Prime project revision.
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size that the `qsys-script` tool uses. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes.

continued...

Option	Usage	Description
`--metrics-log-file=[path_to_log_file]`	Optional	This switch creates a new file containing a set of metrics, at the path you specify. This switch must be used alongside the `--record-metrics` switch, or the switch does not function. The path to the log file must exist, or the `FileDoesNotExists` error appears. Platform Designer does not create any new directories listed in the path.
`--record-metrics[=<foobar>]`	Optional	Enables metric logging. This feature logs the time Platform Designer takes to perform various operations. The other information includes: • The operation performed. • The IP instance name and its `.ip` file location. • The IP component type and its `_hw.tcl` file location. The switch does not require a value, but it does accept a string value that has no effect on the metric logging. In the absence of the `--metrics-log-file` switch, Platform Designer creates the log file in the project directory with the name of `metrics_<top-level name>.txt` by default.
`--help`	Optional	Displays help for the `qsys-script` utility.

Related Information

Saving Platform Designer Systems on page 112

7.8. Parameterizing an Instantiated IP Core after save_system Command

When you call the `save_system` command in your Tcl script, Platform Designer converts all the instantiated IP cores in your system to generic components.

To modify these IP cores after saving your system, you must first load the actual component within the instantiated generic component. Re-parameterize an instantiated IP core using one of the following methods:

1. Load the component in the Platform Designer system, modify the component's parameter value, and save the component:

```
…
save_system kernel_system.qsys
…
load_component cra_root
set_component_parameter_value DATA_W 64
save_component
…
```

2. Load the `.ip` file specific to the component, modify the instance's parameter value, and save the `.ip` file:

```
…
save_system kernel_system.qsys
…
load_system cra_root.ip
set_instance_parameter_value cra_root DATA_W 64
save_system
…
```

Note: To directly modify an instance parameter value after the `save_system` command, you must load the `.ip` file corresponding to the IP component.

Related Information

- set_component_parameter_value on page 584
- load_component on page 581
- save_component on page 583
- save_system on page 471

7.9. Validate the Generic Components in a System with qsys-validate

Use the `qsys-validate` utility to run IP component footprint validation on the `.qsys` file for the system.

Table 209. qsys-validate Command-Line Options

Option	Usage	Description
1st arg file	Optional	The name of the `.qsys` system file to validate.
`--search-path[=<value>]`	Optional	If omitted, Platform Designer uses a standard default path. If provided, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example: `/extra/dir.$`.
`--strict`	Optional	Enables strict validation. All warnings are reported as errors
`--jvm-max-heap-size=<value>`	Optional	The maximum memory size Platform Designer uses for allocations when running `qsys-edit`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Display help for `qsys-validate`.

7.10. Generate an IP Component or Platform Designer System with quartus_ipgenerate

The `quartus_ipgenerate` command allows you to generate IP components or a Platform Designer system in your Quartus Prime project. Ensure that you include the IP component or the Platform Designer system you wish to generate in your Quartus Prime project.

To run the `quartus_ipgenerate` command from the Quartus Prime shell, type:

```
quartus_ipgenerate <project name> [<options>]
```

Use any of the following options with the `quartus_ipgenerate` utility:

Table 210. `quartus_ipgenerate` Command-Line Options

Option	Usage	Description
<1st arg file>	Required	Specifies the name of the Quartus Prime project file (`.qpf`). This option generates all the `.qsys` and `.ip` files in the specified Quartus Prime project (*<project name>*).
`-f [<argument file>]`	Optional	Specifies a file containing additional command-line arguments. Arguments that you specify after this option can conflict or override the options you specify in the argument file.
`--rev[=<revision name>]` or `-c[=<revision name>]`	Optional	Specifies the Quartus Prime project revision and the associated `.qsf` file to use. If you omit this option, Platform Designer uses the same revision name as your Quartus Prime project.

continued...

Option	Usage	Description
`--clear_ip_generation_dirs` or `--clean`	Optional	Clears the generation directories of all the `.qsys` or the `.ip` files in the specified Quartus Prime project. For example, to clear the generation directories in the project `test`, run the following command: `quartus_ipgenerate --clear_ip_generation_dirs test` or `quartus_ipgenerate --clean test`
`--generate_ip_file --ip_file[=<ip file name>]`	Optional	Generates the files for *<file name>*`.ip` file in the specified Quartus Prime project. Use the following optional flags with `--generate_ip_file`: • `-synthesis[=<value>]`—optional argument that specifies the synthesis target type. Specify the value as either *verilog* or *vhdl*. The default value is *verilog*. • `-simulation[=<value>]`—optional argument that specifies the simulation target type. Specify the value as either *verilog* or *vhdl*. If you omit this flag, Platform Designer does not generate any simulation files. • `--clear_ip_generation_dirs`—clears the preexisting generation directories before generation. If you omit this command, Platform Designer does not clear the generation directories. For example, to generate the files for a `test.qsys` file within the project, `test`: `quartus_ipgenerate --generate_ip_file --synthesis=vhdl --simulation=verilog --clear_ip_generation_dirs --ip_file=test.qsys test`
`--generate_project_ip_files [<project name>]`	Optional	Generates the files for all the `.qsys` and `.ip` files in the specified Quartus Prime project. Use any of the following optional flags with `--generate_project_ip_files`: • `-synthesis[=<value>]`—optional argument that specifies the synthesis target type. Specify the value as either *verilog* or *vhdl*. The default value is *verilog*. • `-simulation[=<value>]`—optional argument that specifies the simulation target type. Specify the value as either *verilog* or *vhdl*. If you omit this flag, Platform Designer does not generate any simulation files. • `--clear_ip_generation_dirs`—clears the preexisting generation directories before generation. If you omit this command, Platform Designer does not clear the generation directories. For example, to generate all the `.qsys` and `.ip` files within the project, `test`: `quartus_ipgenerate --generate_project_ip_files --synthesis=vhdl --simulation=verilog --clear_ip_generation_dirs test`
`--get_project_ip_files`	Optional	Returns a list of the `.qsys` or `.ip` files in the specified Quartus Prime project. This option displays each file in a separate Quartus Prime message line. For example, to get a list of `.qsys` files in the project `test`, and revision `rev`: `quartus_ipgenerate --get_project_ip_files test -c rev`
`--lower_priority`	Optional	Allows you to lower the priority of the current process. This option is useful if you use a single-processor computer, allowing you to use other applications more easily while the Quartus Prime software runs the command in the background.

7.11. Generate an IP Variation File with ip-deploy

Use the `ip-deploy` utility to generate an IP variation file (`.ip` file) in the specified location.

Table 211. ip-deploy Command-Line Options

Option	Usage	Description
`--component-name[=<value>]`	Required	The name of a component you instantiate.
`--output-name[=<value>]`	Optional	Name for the resulting component; defaults to the component's type name.
`--component-parameter[=<value>]`	Optional	Repeatable. A single value assignment, like `--component-param=WIDTH=11`. To assign multiple parameters, use this option several times.
`--preset[=<value>]`	Optional	Repeatable. The name of a saved preset to use in creating a variation of the IP component. Presets are additive and repeatable.
`--family[=<value>]`	Optional	Sets the device family
`--part[=<value>]`	Optional	Sets the device part number. You can also use this command to set the base device, device speed-grade, device family, and device feature's system information.
`--output-directory[=<value>]`	Optional	This directory contains the output IP variation file. Platform Designer automatically creates the directory if the directory does not exist. If you do not specify an output directory, the output directory is the current working directory.
`--search-path[=<value>]`	Optional	If you do not specify the search path, the command uses a standard default path. If you provide a search path, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "`$`", like `/extra/dir,$`.
`--jvm-max-heap-size[=<value>]`	Optional	The maximum memory size Platform Designer uses for allocations when running `qsys-edit`. You specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Displays help for `ip-deploy`

7.12. Archive and Extract Platform Designer Systems with qsys-archive

The `qsys-archive` command allows you to archive a system, extract an archived system, and retrieve information about the system's dependencies.

Table 212. `qsys-archive` Command-Line Options

Option	Usage	Description
<1st arg file>	Required	The filename of the root Platform Designer system, Platform Designer file archive, or the Quartus Prime project file.
`--search-path[=<value>]`	Optional	If you omit this option, Platform Designer uses a standard default path. If you specify this option, Platform Designer searches a comma-separated list of paths. To include the standard path in your replacement, use "$", for example: `/extra/dir,$`.
`--archive`	Optional	Creates a zip archive of the specified Platform Designer system or the Quartus Prime project.
`--report-file[=<value>]`	Optional	Lists the files that the Platform Designer system or the Quartus Prime project references, and writes the files list to the specified name in `.txt` format.
`--output-directory[=<file>]`	Optional	Specifies the output directory to save the archive.
`--extract`	Optional	Extracts all the files in the given archive.
`--output-name[=<value>]`	Optional	Specifies the output name to save the archive or report.
`collect-to-common-directory[=<true|false>]`	Optional	When archiving, collects all the `.qsys` files in the root directory of the archive and all `.ip` files in a single `ip` directory, and updates all the matching references. The default option is *true*.
`new-quartus-project[=<value>]`	Optional	Creates a new Quartus Prime project which contains all the `.ip` and system files referenced by the Platform Designer system or the Quartus Prime project.
`quartus-project[=<value>]`	Optional	When you use this command in combination with: • `--report-file`—adds all the referenced files to the Quartus Prime project. • `--extract`—adds all extracted files to the specified project. • `--archive`—archives all the system and `.ip` files referenced in the Quartus Prime project.
`--rev`	Optional	Specifies the name of the Quartus Prime project revision.
`--include-generated-files`	Optional	Includes all the generated files of the Platform Designer system.
`--force`	Optional	Forcefully creates the specified archive or report, overwriting any existing archives or reports.
`--jvm-max-heap-size=<value>`	Optional	Specifies the maximum memory size Platform Designer uses for allocations when running `qsys-edit`. Specify this value as `<size><unit>`, where unit is `m` (or `M`) for multiples of megabytes, or `g` (or `G`) for multiples of gigabytes. The default value is `512m`.
`--help`	Optional	Displays help for `qsys-archive`.

Alternatively, you can archive and restore your system using the Platform Designer GUI. For more information, refer to *Saving and Archiving Platform Designer Systems* section.

Related Information

Saving and Archiving Platform Designer Systems on page 112

7.13. Apply Presets to a New Board

The `get_presets` command allows you to determine if an instance has a preset that matches the current configuration. `apply_presets` allows you to apply presets to your board.

Table 213. Scripting Command to Apply Presets to New Board

Return	Command	Argument
No return value	`apply_presets`	Applies the presets that you specify to the Platform Designer system that your specify. `<systemName_ipName_presetName>` `systemName_ipName_presetName` consists of systems name, instance name, and preset name.
String[]	`get_presets`	Determines if the instance has a preset that matches the current configuration `<instance_name> <enable_board_filter> <enable_matched_preset>` `enable_matched_preset` indicates whether to perform filtering based on the module instance's parameters and/or pin assignments values.

Example 1: Change Board and Apply Built-in Preset to Components

```
$ qsys-script --script=board_migration.tcl --quartus-project=project \
--system-file=top_system.qsys
```

`board_migration.tcl` content for a single hierarchy system example:

```
package require -exact qsys 24.1

set_project_property BOARD {new board}
set_project_property DEVICE {new part}
set_project_property DEVICE_FAMILY {new family}

apply_presets {{top_system,top_ipinst0, top_ipinst0_preset0}}
```

`board_migration.tcl` content for a multiple hierarchy system example:

```
package require -exact qsys 24.1

set_project_property BOARD {new board}
set_project_property DEVICE {new part}
set_project_property DEVICE_FAMILY {new family}

apply_presets {{top_system,top_ipinst0, top_ipinst0_preset0} \
{top_system,top_ipinst1, top_ipinst1_preset1} {sub_system_name,subipinst0,
subip_preset0}}
```

Initially, you must change the board for the project in the Quartus Prime software (**Assignments ➤ Device ➤ Board**) before you can run the script.

Example 2: Determine If System Instances are Already Configured in the New Board

```
$ qsys-script --script=board_migration.tcl --quartus-project=project \
--system-file=top_system.qsys
```

`board_migration.tcl` content for a single hierarchy system example:

```
package require -exact qsys 24.1

set_project_property BOARD {new board}
```

```
set_project_property DEVICE {new part}
set_project_property DEVICE_FAMILY {new family}

set module_list [get_instances]

foreach module $module_list {
  set num_matched_preset [llength [get_presets $module true true]]
  if ($num_matched_preset > 0) {
    # do something when the instance configured
  }
}
```

`get_presets` supports the optional `enable_matched_preset` argument that allows you to determine if the instance has a preset that matches the current configuration.

7.14. Platform Designer Scripting Command Reference

Platform Designer system scripting provides Tcl commands to manipulate your system. The `qsys-script` provides a command-line alternative to the Platform Designer tool. Use the `qsys-script` commands to create and modify your system, as well as to create reports about the system.

To use the current version of the Tcl commands, include the following line at the top of your script:

```
package require -exact qsys <version>
```

For example, for the current release of the Quartus Prime software, include:

```
package require -exact qsys 18.0
```

The Platform Designer scripting commands fall under the following categories:

7.14.1. System

This section lists the commands that allow you to manipulate a Platform Designer system.

7.14.1.1. create_system

Description

Replaces the current system with a new system of the specified name.

Usage

`create_system [<name>]`

Returns

No return value.

Arguments

name (optional) The new system name.

Example

```
create_system my_new_system_name
```

Related Information

- load_system on page 470
- save_system on page 471

7.14.1.2. export_hw_tcl

Description

Allows you to save the currently open system as an `_hw.tcl` file in the project directory. The saved systems appears under the **System** category in the IP Catalog.

Usage

`export_hw_tcl`

Returns

No return value.

Arguments

No arguments

Example

```
export_hw_tcl
```

Related Information

- load_system on page 470
- save_system on page 471

7.14.1.3. get_device_families

Description

Returns the list of installed device families.

Usage

```
get_device_families
```

Returns

String[] The list of device families.

Arguments

No arguments

Example

```
get_device_families
```

Related Information

get_devices on page 465

7.14.1.4. get_devices

Description

Returns the list of installed devices for the specified family.

Usage

`get_devices <family>`

Returns

String[] The list of devices.

Arguments

family Specifies the family name to get the devices for.

Example

```
get_devices exampleFamily
```

Related Information

get_device_families on page 464

7.14.1.5. get_module_properties

Description

Returns the properties that you can manage for a top-level module of the Platform Designer system.

Usage

```
get_module_properties
```

Returns

The list of property names.

Arguments

No arguments.

Example

```
get_module_properties
```

Related Information

- get_module_property on page 467
- set_module_property on page 473

7.14.1.6. get_module_property

Description

Returns the value of a top-level system property.

Usage

`get_module_property` <*property*>

Returns

The property value.

Arguments

property The property name to query. Refer to *Module Properties*.

Example

```
get_module_property NAME
```

Related Information

- get_module_properties on page 466
- set_module_property on page 473

7.14.1.7. get_project_properties

Description

Returns the list of properties that you can query for properties pertaining to the Quartus Prime project.

Usage

```
get_project_properties
```

Returns

The list of project properties.

Arguments

No arguments

Example

```
get_project_properties
```

Related Information

- get_project_property on page 469
- set_project_property on page 474

7.14.1.8. get_project_property

Description

Returns the value of an Quartus Prime project property.

Usage

`get_project_property` <*property*>

Returns

The property value.

Arguments

property The project property name. Refer to *Project properties*.

Example

```
get_project_property DEVICE_FAMILY
```

Related Information

- get_module_properties on page 466
- get_module_property on page 467
- set_module_property on page 473
- Project Properties on page 651

7.14.1.9. load_system

Description

Loads the Platform Designer system from a file, and uses the system as the current system for scripting commands.

Usage

`load_system` <*file*>

Returns

No return value.

Arguments

file The path to the `.qsys` file.

Example

```
load_system example.qsys
```

Related Information

- create_system on page 462
- save_system on page 471

7.14.1.10. save_system

Description

Saves the current system to the specified file. If you do not specify the file, Platform Designer saves the system to the same file opened with the `load_system` command.

Usage

`save_system` <*file*>

Returns

No return value.

Arguments

file If available, the path of the `.qsys` file to save.

Example

```
save_system
```

```
save_system file.qsys
```

Related Information

- load_system on page 470
- create_system on page 462

7.14.1.11. set_design_id

Description

Specifies an alphanumeric string that identifies the system.

If you specify `set_design_id` when creating a system, the generated system includes a SOPCINFO file with a new `design_id` attribute in the EnsembleReport element:

```
<EnsembleReport
name="my_system"
kind="my_system"
design_id="my_design_id"
version="1.0"
fabric="QSYS">
```

Usage

`set_design_id` <*design-id*>

Returns

No return value.

Arguments

design-id (optional) An alphanumeric string that identifies the system.

Example

```
set_design_id foo121
```

7.14.1.12. set_module_property

Description

Specifies the Tcl procedure to evaluate changes in Platform Designer system instance parameters.

Usage

`set_module_property` <*property*> <*value*>

Returns

No return value.

Arguments

property The property name. Refer to *Module Properties*.

value The new value of the property.

Example

```
set_module_property COMPOSITION_CALLBACK "my_composition_callback"
```

Related Information

- get_module_properties on page 466
- get_module_property on page 467
- Module Properties on page 645

7.14.1.13. set_project_property

Description

Sets the project property value, such as the device family.

Usage

`set_project_property` <*property*> <*value*>

Returns

No return value.

Arguments

property The property name. Refer to *Project Properties*.

value The new property value.

Example

```
set_project_property DEVICE_FAMILY "Cyclone IV GX"
```

Related Information

- get_project_properties on page 468
- get_project_property on page 469
- Project Properties on page 651

7.14.2. Subsystems

This section lists the commands that allow you to obtain the connection and parameter information of instances in your Platform Designer subsystem.

7.14.2.1. get_composed_connections

Description

Returns the list of all connections in the subsystem for an instance that contains the subsystem of the Platform Designer system.

Usage

`get_composed_connections` <*instance*>

Returns

The list of connection names in the subsystem.

Arguments

instance The child instance containing the subsystem.

Example

```
get_composed_connections subsystem_0
```

Related Information

- get_composed_connection_parameter_value on page 477
- get_composed_connection_parameters on page 478

7.14.2.2. get_composed_connection_parameter_value

Description

Returns the parameter value of a connection in a child instance containing the subsystem.

Usage

`get_composed_connection_parameter_value` <*instance*> <*child_connection*> <*parameter*>

Returns

The parameter value.

Arguments

instance The child instance that contains the subsystem.

child_connection The connection name in the subsystem.

parameter The parameter name to query for the connection.

Example

```
get_composed_connection_parameter_value subsystem_0 cpu.data_host/memory.s0
baseAddress
```

Related Information

- get_composed_connection_parameters on page 478
- get_composed_connections on page 476

7.14.2.3. get_composed_connection_parameters

Description

Returns the list of parameters of a connection in the subsystem, for an instance that contains the subsystem.

Usage

`get_composed_connection_parameters` <*instance*> <*child_connection*>

Returns

The list of parameter names.

Arguments

instance The child instance containing the subsystem.

child_connection The name of the connection in the subsystem.

Example

```
get_composed_connection_parameters subsystem_0 cpu.data_host/memory.s0
```

Related Information

- get_composed_connection_parameter_value on page 477
- get_composed_connections on page 476

7.14.2.4. get_composed_instance_assignment

Description

Returns the assignment value of the child instance in the subsystem.

Usage

`get_composed_instance_assignment` <*instance*> <*child_instance*> <*assignment*>

Returns

The assignment value.

Arguments

instance The subsystem containing the child instance.

child_instance The child instance name in the subsystem.

assignment The assignment key.

Example

```
get_composed_instance_assignment subsystem_0 video_0
"embeddedsw.CMacro.colorSpace"
```

Related Information

- get_composed_instance_assignments on page 480
- get_composed_instances on page 483

7.14.2.5. get_composed_instance_assignments

Description

Returns the list of assignments of the child instance in the subsystem.

Usage

`get_composed_instance_assignments` <*instance*> <*child_instance*>

Returns

The list of assignment names.

Arguments

instance The subsystem containing the child instance.

child_instance The child instance name in the subsystem.

Example

```
get_composed_instance_assignments subsystem_0 cpu
```

Related Information

- get_composed_instance_assignment on page 479
- get_composed_instances on page 483

7.14.2.6. get_composed_instance_parameter_value

Description

Returns the parameter value of the child instance in the subsystem.

Usage

`get_composed_instance_parameter_value` <*instance*> <*child_instance*> <*parameter*>

Returns

The parameter value of the instance in the subsystem.

Arguments

instance The subsystem containing the child instance.

child_instance The child instance name in the subsystem.

parameter The parameter name to query on the child instance in the subsystem.

Example

```
get_composed_instance_parameter_value subsystem_0 cpu DATA_WIDTH
```

Related Information

- get_composed_instance_parameters on page 482
- get_composed_instances on page 483

7.14.2.7. get_composed_instance_parameters

Description

Returns the list of parameters of the child instance in the subsystem.

Usage

`get_composed_instance_parameters` <*instance*> <*child_instance*>

Returns

The list of parameter names.

Arguments

instance The subsystem containing the child instance.

child_instance The child instance name in the subsystem.

Example

```
get_composed_instance_parameters subsystem_0 cpu
```

Related Information

- get_composed_instance_parameter_value on page 481
- get_composed_instances on page 483

7.14.2.8. get_composed_instances

Description

Returns the list of child instances in the subsystem.

Usage

`get_composed_instances <instance>`

Returns

The list of instance names in the subsystem.

Arguments

instance The subsystem containing the child instance.

Example

```
get_composed_instances subsystem_0
```

Related Information

- get_composed_instance_assignment on page 479
- get_composed_instance_assignments on page 480
- get_composed_instance_parameter_value on page 481
- get_composed_instance_parameters on page 482

7.14.3. Domains and Interfaces

This section lists the commands that allow you to specify parameters for domains and interfaces in your system.

Related Information

Specifying Interconnect Parameters on page 71

7.14.3.1. set_domain_assignment

Description

Sets the assignment value to all connections on the given domain.

Usage

```
set_domain_assignment <element> <assignment> <value>
```

Arguments

element Connection or interface in the domain to set with assignment. If element name is $system, assigns to all the domains in the system.

assignment The name of the assignment.

value The value of the assignment.

7.14.3.2. get_domain_assignment

Description

Obtains the value for specified assignment in the given domain.

Usage

```
get_domain_assignment <element> <assignment>
```

Arguments

element Connection or interface in the domain for which you want the assignment value.

assignment The name of the assignment.

7.14.3.3. get_domain_assignments

Description

Obtains all domain assignments for the given domain as a list of strings. Each "group" of three elements in the list contains the element name, assignment name, and value (in that order). Element name in the output is the input element name. If the input element is $system, then the output element name is the connection point in the domain. For example, typical list contents appear like this:

```
[element0 name0 value0 element1 name1 value1 ... ]
```

In TCL, you'd loop over the list by writing a foreach loop:

```
foreach {element name value } $requirement_list \
     { puts " $element $name $value" }
```

Usage

```
get_domain_assignments <element>
```

Arguments

element Connection or interface in the domain for which you want to get assignments. If element is specified as $system, gives values of all the domains in the system.

7.14.3.4. set_interface_assignment

Description

Adds interconnect assignment to the interface.

Usage

```
set_interface_assignment <interface> <assignment> <value>
```

Arguments

interface Interface name.

assignment The name of the assignment.

value The value of the assignment.

7.14.3.5. get_interface_assignment

Description

Obtains the value of the named interface interconnect assignment on the specified interface.

Usage

```
get_interface_assignment <interface> <assignment>
```

Arguments

interface Interface name.

assignment The name of the assignment.

7.14.3.6. get_interface_assignments

Description

Obtains all interface interconnect assignments for the given domain as a list of strings. Each "group" of three elements in the list contains the interface name, assignment name, and value (in that order). For example, typical list contents might look like this:

```
[interface0 name0 value0 interface1 name1 value1 ... ]
```

In TCL, you'd loop over the list by writing a foreach loop:

```
foreach {interface name value } $requirement_list \
     { puts " $interface $name $value" }
```

Usage

```
get_interface_assignments <interface>
```

Arguments

interface Interface name that you want to get assignments for. If interface is specified as $system , it gives assignments of all the interfaces in the system.

assignment The name of the assignment.

7.14.3.7. set_postadaptation_assignment

Description

Adds a post adaptation interconnect assignment.

Usage

```
set_postadaptation_assignment <element> <assignment> <value>
```

Arguments

element Element name.

assignment The name of the assignment.

value The value of the assignment.

7.14.3.8. get_postadaptation_assignment

Description

Obtains the value of the named post adaptation interconnect assignment on the specified element.

Usage

```
get_postadaptation_assignment <element> <assignment>
```

Arguments

element Element name.

assignment The name of the assignment.

7.14.3.9. get_postadaptation_assignments

Description

Obtains all post adaptation interconnect assignments for the given domain as a list of strings. Each "group" of three elements in the list contains the element name, assignment name, and value (in that order). For example, typical list contents might look like this:

```
[element0 name0 value0 element1 name1 value1 ... ]
```

In TCL, you'd loop over the list by writing a foreach loop:

```
foreach {element name value } $requirement_list \
     { puts " $element $name $value" }
```

Usage

```
get_postadaptation_assignments <interface>
```

Arguments

interface Interface name that you want to get assignments for. If interface is specified as $system , it gives assignments of all the interfaces in the system.

7.14.4. Instances

This section lists the commands that allow you to manipulate the instances of IP components in your Platform Designer system.

7.14.4.1. add_instance

Description

Adds an instance of a component, referred to as a *child* or *child instance*, to the system.

Usage

`add_instance` <*name*> <*type*> [<*version*>]

Returns

No return value.

Arguments

name Specifies a unique local name that you can use to manipulate the instance. Platform Designer uses this name in the generated HDL to identify the instance.

type Refers to a kind of instance available in the IP Catalog, for example `altera_avalon_uart`.

version The required version of the specified instance type. This argument is required in Package version 19.1 and later. Before package version 19.1, when not specified the latest IP version is used.

Example

```
add_instance uart_0 altera_avalon_uart 22.2
```

Related Information

- get_instance_property on page 514
- get_instances on page 515
- remove_instance on page 517
- set_instance_parameter_value on page 518
- get_instance_parameter_value on page 509

7.14.4.2. apply_instance_preset

Description

Applies the settings in a preset to the specified instance.

Usage

`apply_instance_preset` <*preset_name*>

Returns

No return value.

Arguments

preset_name The preset name.

Example

```
apply_preset "Custom Debug Settings"
```

Related Information

set_instance_parameter_value on page 518

7.14.4.3. create_ip

Description

Creates a new IP Variation system with the given instance.

Usage

`create_ip` <*type*> [<*instance_name*> <*version*>]

Returns

No return value.

Arguments

type Kind of instance available in the IP catalog, for example, `altera_avalon_uart`.

instance_name (optional) A unique local name that you can use to manipulate the instance. If not specified, Platform Designer uses a default name.

version (optional) The required version of the specified instance type. If not specified, Platform Designer uses the latest version.

Example

```
create_ip altera_avalon_uart altera_avalon_uart_inst 17.0
```

Related Information

- add_component on page 492
- load_system on page 470
- save_system on page 471
- set_instance_parameter_value on page 518

7.14.4.4. add_component

Description

Adds a new IP Variation component to the system.

Usage

`add_component` <*instance_name*> <*file_name*> [<*component_type*> <*component_instance_name*> <*component_version*>]

Returns

No return value.

Arguments

instance_name A unique local name that you can use to manipulate the instance.

file_name The IP variation file name. If a path is not specified, Platform Designer saves the file in the `./ip/system/` sub-folder of your system.

component_type (optional) The kind of instance available in the IP catalog, for example `altera_avalon_uart`.

component_instance_name (optional) The instance name of the component in the IP variation file. If not specified, Platform Designer uses a default name.

component_version (optional) The required version of the specified instance type. If not specified, Platform Designer uses the latest version.

Example

```
add_component myuart_0 myuart.ip altera_avalon_uart altera_avalon_uart_inst 17.0
```

Related Information

save_system on page 471

7.14.4.5. duplicate_instance

Description

Creates a duplicate instance of the specified instance.

Usage

`duplicate_instance` <*instance*> [<*name*>]

Returns

String The new instance name.

Arguments

instance Specifies the instance name to duplicate.

name (*optional*) Specifies the name of the duplicate instance.

Example

```
duplicate_instance cpu cpu_0
```

Related Information

- add_instance on page 489
- remove_instance on page 517

7.14.4.6. enable_instance_parameter_update_callback

Description

Enables the update callback for instance parameters.

Usage

`enable_instance_parameter_update_callback [<value>]`

Returns

No return value.

Arguments

value (*optional*)	Specifies whether to enable/disable the instance parameters callback. Default option is "1".

Example

```
enabled_instance_parameter_update_callback
```

Related Information

- is_instance_parameter_update_callback_enabled on page 516
- set_instance_parameter_value on page 518

7.14.4.7. get_instance_assignment

Description

Returns the assignment value of a child instance. Platform Designer uses assignments to transfer information about hardware to embedded software tools and applications.

Usage

`get_instance_assignment` <*instance*> <*assignment*>

Returns

String The value of the specified assignment.

Arguments

instance The instance name.

assignment The assignment key to query.

Example

```
get_instance_assignment video_0 embeddedsw.CMacro.colorSpace
```

Related Information

get_instance_assignments on page 496

7.14.4.8. get_instance_assignments

Description

Returns the list of assignment keys for any defined assignments for the instance.

Usage

`get_instance_assignments <instance>`

Returns

String[] The list of assignment keys.

Arguments

instance The instance name.

Example

```
get_instance_assignments sdram
```

Related Information

get_instance_assignment on page 495

7.14.4.9. get_instance_documentation_links

Description

Returns the list of all documentation links provided by an instance.

Usage

`get_instance_documentation_links` <*instance*>

Returns

String[] The list of documentation links.

Arguments

instance The instance name.

Example

```
get_instance_documentation_links cpu_0
```

Notes

The list of documentation links includes titles and URLs for the links. For instance, a component with a single data sheet link may return:

```
{Data Sheet} {http://url/to/data/sheet}
```

7.14.4.10. get_instance_interface_assignment

Description

Returns the assignment value for an interface of a child instance. Platform Designer uses assignments to transfer information about hardware to embedded software tools and applications.

Usage

`get_instance_interface_assignment` <*instance*> <*interface*> <*assignment*>

Returns

String The value of the specified assignment.

Arguments

instance The child instance name.

interface The interface name.

assignment The assignment key to query.

Example

```
get_instance_interface_assignment sdram s1 embeddedsw.configuration.isFlash
```

Related Information

get_instance_interface_assignments on page 499

7.14.4.11. get_instance_interface_assignments

Description

Returns the list of assignment keys for any assignments defined for an interface of a child instance.

Usage

`get_instance_interface_assignments` <*instance*> <*interface*>

Returns

String[] The list of assignment keys.

Arguments

instance The child instance name.

interface The interface name.

Example

```
get_instance_interface_assignments sdram s1
```

Related Information

get_instance_interface_assignment on page 498

7.14.4.12. get_instance_interface_parameter_property

Description

Returns the property value for a parameter in an interface of an instance. Parameter properties are metadata about how Platform Designer uses the parameter.

Usage

`get_instance_interface_parameter_property` <*instance*> <*interface*> <*parameter*> <*property*>

Returns

various The parameter property value.

Arguments

instance The child instance name.

interface The interface name.

parameter The parameter name for the interface.

property The property name for the parameter. Refer to *Parameter Properties*.

Example

```
get_instance_interface_parameter_property uart_0 s0 setupTime ENABLED
```

Related Information

- get_instance_interfaces on page 502
- get_instance_interfaces on page 507
- get_parameter_properties on page 627
- Parameter Properties on page 646

7.14.4.13. get_instance_interface_parameter_value

Description

Returns the parameter value of an interface in an instance.

Usage

`get_instance_interface_parameter_value` <*instance*> <*interface*> <*parameter*>

Returns

various The parameter value.

Arguments

instance The child instance name.

interface The interface name.

parameter The parameter name for the interface.

Example

```
get_instance_interface_parameter_value uart_0 s0 setupTime
```

Related Information

- get_instance_interface_parameters on page 502
- get_instance_interfaces on page 507

7.14.4.14. get_instance_interface_parameters

Description

Returns the list of parameters for an interface in an instance.

Usage

`get_instance_interface_parameters` <*instance*> <*interface*>

Returns

String[] The list of parameter names for parameters in the interface.

Arguments

instance The child instance name.

interface The interface name.

Example

```
get_instance_interface_parameters uart_0 s0
```

Related Information

- get_instance_interface_parameter_value on page 501
- get_instance_interfaces on page 507

7.14.4.15. get_instance_interface_port_property

Description

Returns the property value of a port in the interface of a child instance.

Usage

`get_instance_interface_port_property` <*instance*> <*interface*> <*port*> <*property*>

Returns

various The port property value.

Arguments

instance The child instance name.

interface The interface name.

port The port name.

property The property name of the port. Refer to *Port Properties*.

Example

```
get_instance_interface_port_property uart_0 exports tx WIDTH
```

Related Information

- get_instance_interface_ports on page 504
- get_port_properties on page 607
- Port Properties on page 650

7.14.4.16. get_instance_interface_ports

Description

Returns the list of ports in an interface of an instance.

Usage

`get_instance_interface_ports` <*instance*> <*interface*>

Returns

String[] The list of port names in the interface.

Arguments

instance The instance name.

interface The interface name.

Example

```
get_instance_interface_ports uart_0 s0
```

Related Information

- get_instance_interface_port_property on page 503
- get_instance_interfaces on page 507

7.14.4.17. get_instance_interface_properties

Description

Returns the list of properties that you can query for an interface in an instance.

Usage

```
get_instance_interface_properties
```

Returns

String[] The list of property names.

Arguments

No arguments.

Example

```
get_instance_interface_properties
```

Related Information

- get_instance_interface_property on page 506
- get_instance_interfaces on page 507

7.14.4.18. get_instance_interface_property

Description

Returns the property value for an interface in a child instance.

Usage

`get_instance_interface_property` <*instance*> <*interface*> <*property*>

Returns

String The property value.

Arguments

instance The child instance name.

interface The interface name.

property The property name. Refer to *Element Properties*.

Example

```
get_instance_interface_property uart_0 s0 DESCRIPTION
```

Related Information

- get_instance_interface_properties on page 505
- get_instance_interfaces on page 507
- Element Properties on page 641

7.14.4.19. get_instance_interfaces

Description

Returns the list of interfaces in an instance.

Usage

`get_instance_interfaces <instance>`

Returns

String[] The list of interface names.

Arguments

instance The instance name.

Example

```
get_instance_interfaces uart_0
```

Related Information

- get_instance_interface_ports on page 504
- get_instance_interface_properties on page 505
- get_instance_interface_property on page 506

7.14.4.20. get_instance_parameter_property

Description

Returns the property value of a parameter in an instance. Parameter properties are metadata about how Platform Designer uses the parameter.

Usage

`get_instance_parameter_property` <*instance*> <*parameter*> <*property*>

Returns

various The parameter property value.

Arguments

instance The instance name.

parameter The parameter name.

property The property name of the parameter. Refer to *Parameter Properties*.

Example

```
get_instance_parameter_property uart_0 baudRate ENABLED
```

Related Information

- get_instance_parameters on page 511
- get_parameter_properties on page 627
- Parameter Properties on page 646

7.14.4.21. get_instance_parameter_value

Description

Returns the parameter value in a child instance.

Usage

`get_instance_parameter_value` <*instance*> <*parameter*>

Returns

various The parameter value.

Arguments

instance The instance name.

parameter The parameter name.

Example

```
get_instance_parameter_value pixel_converter input_DPI
```

Related Information

- get_instance_parameters on page 511
- set_instance_parameter_value on page 518

7.14.4.22. get_instance_parameter_values

Description

Returns a list of the parameters' values in a child instance.

Usage

`get_instance_parameter_values` <*instance*> <*parameters*>

Returns

String[] A list of the parameters' value.

Arguments

instance The child instance name.

parameter A list of parameter names in the instance.

Example

```
get_instance_parameter_value uart_0 [list param1 param2]
```

Related Information

- get_instance_parameters on page 511
- set_instance_parameter_value on page 518
- set_instance_parameter_values on page 519

7.14.4.23. get_instance_parameters

Description

Returns the names of all parameters for a child instance that the parent can manipulate. This command omits derived parameters and parameters that have the SYSTEM_INFO parameter property set.

Usage

`get_instance_parameters` <*instance*>

Returns

instance The list of parameters in the instance.

Arguments

instance The instance name.

Example

```
get_instance_parameters uart_0
```

Related Information

- get_instance_parameter_property on page 508
- get_instance_parameter_value on page 509
- set_instance_parameter_value on page 518

7.14.4.24. get_instance_port_property

Description

Returns the property value of a port contained by an interface in a child instance.

Usage

`get_instance_port_property` <*instance*> <*port*> <*property*>

Returns

various The property value for the port.

Arguments

instance The child instance name.

port The port name.

property The property name. Refer to *Port Properties*.

Example

```
get_instance_port_property uart_0 tx WIDTH
```

Related Information

- get_instance_interface_ports on page 504
- get_port_properties on page 607
- Port Properties on page 650

7.14.4.25. get_instance_properties

Description

Returns the list of properties for a child instance.

Usage

```
get_instance_properties
```

Returns

String[] The list of property names for the child instance.

Arguments

No arguments.

Example

```
get_instance_properties
```

Related Information

get_instance_property on page 514

7.14.4.26. get_instance_property

Description

Returns the property value for a child instance.

Usage

`get_instance_property` <*instance*> <*property*>

Returns

String The property value.

Arguments

instance The child instance name.

property The property name. Refer to *Element Properties*.

Example

```
get_instance_property uart_0 ENABLED
```

Related Information

- get_instance_properties on page 513
- Element Properties on page 641

7.14.4.27. get_instances

Description

Returns the list of the instance names for all the instances in the system.

Usage

```
get_instances
```

Returns

String[] The list of child instance names.

Arguments

No arguments.

Example

```
get_instances
```

Related Information

- add_instance on page 489
- remove_instance on page 517

7.14.4.28. is_instance_parameter_update_callback_enabled

Description

Returns true if you enable the update callback for instance parameters.

Usage

```
is_instance_parameter_update_callback_enabled
```

Returns

boolean 1 if you enable the callback; 0 if you disable the callback.

Arguments

No arguments

Example

```
is_instance_parameter_update_callback_enabled
```

Related Information

enable_instance_parameter_update_callback on page 494

7.14.4.29. remove_instance

Description

Removes an instance from the system.

Usage

`remove_instance <instance>`

Returns

No return value.

Arguments

instance The child instance name to remove.

Example

```
remove_instance cpu
```

Related Information

- add_instance on page 489
- get_instances on page 515

7.14.4.30. set_instance_parameter_value

Description

Sets the parameter value for a child instance. You cannot set derived parameters and SYSTEM_INFO parameters for the child instance with this command.

Usage

`set_instance_parameter_value` <*instance*> <*parameter*> <*value*>

Returns

No return value.

Arguments

instance The child instance name.

parameter The parameter name.

value The parameter value.

Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

Related Information

- get_instance_parameter_value on page 509
- get_instance_parameter_property on page 508

7.14.4.31. set_instance_parameter_values

Description

Sets a list of parameter values for a child instance. You cannot set derived parameters and SYSTEM_INFO parameters for the child instance with this command.

Usage

`set_instance_parameter_value` <*instance*> <*parameter_value_pairs*>

Returns

No return value.

Arguments

instance The child instance name.

parameter_value_pairs The pairs of parameter name and value to set.

Example

```
set_instance_parameter_value uart_0 [list baudRate 9600 parity odd]
```

Related Information

- get_instance_parameter_value on page 509
- get_instance_parameter_values on page 510
- get_instance_parameters on page 511

7.14.4.32. set_instance_property

Description

Sets the property value of a child instance. Most instance properties are read-only and can only be set by the instance itself. The primary use for this command is to update the ENABLED parameter, which includes or excludes a child instance when generating Platform Designer interconnect.

Usage

`set_instance_property` <*instance*> <*property*> <*value*>

Returns

No return value.

Arguments

instance The child instance name.

property The property name. Refer to *Instance Properties*.

value The property value.

Example

```
set_instance_property cpu ENABLED false
```

Related Information

- get_instance_parameters on page 511
- get_instance_property on page 514
- Instance Properties on page 642

7.14.5. Instantiations

This section lists the commands that allow you to manipulate the loaded instantiations in a Platform Designer system.

7.14.5.1. add_instantiation_hdl_file

Description

Adds an HDL file to the loaded instantiation.

Usage

`add_instantiation_hdl_file <file> [<kind>]`

Returns

No return value.

Arguments

file Specifies the HDL file name.

kind(optional) Indicates the file set kind to add the file to. If you do not specify this option, the command adds the file to all the file sets. Refer to *File Set Kind*.

Example

```
add_instantiation_hdl_file my_nios2_gen2.vhdl quartus_synth
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- File Set Kind on page 657

7.14.5.2. add_instantiation_interface

Description

Adds an interface to the loaded instantiation.

Usage

`add_instantiation_interface` <*interface*> <*type*> <*direction*>

Returns

No return value.

Arguments

interface Specifies the interface name.

type Specifies the interface type.

direction Specifies the interface direction. Refer to *Interface Direction*.

Example

```
add_instantiation_interface clk_0 clock OUTPUT
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- Interface Direction on page 656

7.14.5.3. add_instantiation_interface_port

Description

Adds a port to a loaded instantiation's interface.

Usage

`add_instantiation_interface_port` *<interface> <port> <role> <width> <vhdl_type><direction>*

Returns

No return value.

Arguments

interface Specifies the interface name.

port Specifies the port name.

role Specifies the port role.

width Specifies the port width.

vhdl_type Specifies the VHDL type of the port. Refer to *VHDL Type*.

direction Specifies the port direction. Refer to *Direction Properties*.

Example

```
add_instantiation_interface_port avs_s0 avs_s0_address address 8 {standard logic
vector} input
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- VHDL Type on page 664
- Direction Properties on page 640

7.14.5.4. copy_instance_interface_to_instantiation

Description

Adds an interface to a loaded instantiation by copying the specified interface of another instance.

Usage

`copy_instance_interface_to_instantiation` <*instance*> <*interface*> <*type*>

Returns

String The name of the newly added interface.

Arguments

instance Specifies the name of the instance to copy the interface from.

interface Specifies the name of the interface to copy.

type Specifies the type of copy to make. Refer to *Instantiation Interface Duplicate Type*.

Example

```
copy_instance_interface_to_instantiation cpu_0 data_host CLONE
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- Instantiation Interface Duplicate Type on page 660

7.14.5.5. get_instantiation_assignment_value

Description

Gets the assignment value on the loaded instantiation.

Usage

`get_instantiation_assignment_value <name>`

Returns

String The assignment value.

Arguments

name Specifies the name of the assignment to get the value of.

Example

```
get_instantiation_assignment_value embeddedsw.configuration.exceptionOffset
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.6. get_instantiation_assignments

Description

Gets the assignment names in the loaded instantiation.

Usage

```
get_instantiation_assignments
```

Returns

String[] The list of assignment names.

Arguments

No arguments

Example

```
get_instantiation_assignments
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.7. get_instantiation_hdl_file_properties

Description

Returns the list of properties in an HDL file associated with an instantiation.

Usage

```
get_instantiation_hdl_file_properties
```

Returns

String[] The list of property names.

Arguments

No arguments

Example

```
get_instantiation_hdl_file_properties
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.8. get_instantiation_hdl_file_property

Description

Returns the property value of an HDL file associated with the loaded instantiation.

Usage

`get_instantiation_hdl_file_property` <*file*> <*property*>

Returns

various The property value.

Arguments

file Specifies the HDL file name.

property Specifies the property name. Refer to *Instantiation Hdl File Properties*.

Example

```
get_instantiation_hdl_file_property my_nios2_gen2.vhdl OUTPUT_PATH
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- Instantiation HDL File Properties on page 659

7.14.5.9. get_instantiation_hdl_files

Description

Returns the list of HDL files of the loaded instantiation.

Usage

`get_instantiation_hdl_files` [<*kind*>]

Returns

String[] The list of HDL file names.

Arguments

kind (*optional*) Specifies the file set kind to get the files of. If you do not specify this option, the command gets the QUARTUS_SYNTH files. Refer to *File Set Kind*.

Example

```
get_instantiation_hdl_files quartus_synth
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- File Set Kind on page 657

7.14.5.10. get_instantiation_interface_assignment_value

Description

Gets the assignment value of the loaded instantiation's interface.

Usage

`get_instantiation_interface_assignment_value` <*interface*> <*name*>

Returns

String The assignment value

Arguments

interface Specifies the interface name.

name Specifies the assignment name to get the value of.

Example

```
get_instantiation_interface_assignment_value avs_s0
embeddedsw.configuration.exceptionOffset
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.11. get_instantiation_interface_assignments

Description

Gets the assignment names of the loaded instantiation's interface.

Usage

`get_instantiation_interface_assignments` <*interface*>

Returns

String[] The list of assignment names.

Arguments

interface Specifies the interface name.

Example

```
get_instantiation_interface_assignments avs_s0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.12. get_instantiation_interface_parameter_value

Description

Returns the parameter value of a loaded instantiation's interface.

Usage

`get_instantiation_interface_parameter_value` <*interface*> <*parameter*>

Returns

String The parameter value.

Arguments

interface Specifies the interface name.

parameter Specifies the parameter name.

Example

```
get_instantiation_interface_parameter_value avs_s0 associatedClock
```

Related Information

- get_instantiation_interface_parameters on page 535
- set_instantiation_interface_parameter_value on page 556
- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.13. get_instantiation_interface_parameters

Description

Returns the list of parameters of an instantiation's interface.

Usage

`get_instantiation_interface_parameters` <*interface*>

Returns

String[] The list of parameter names.

Arguments

interface Specifies the interface name.

Example

```
get_instantiation_interface_parameters avs_s0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- get_instantiation_interface_parameter_value on page 534
- set_instantiation_interface_parameter_value on page 556

7.14.5.14. get_instantiation_interface_port_properties

Description

Returns the list of port properties of an instantiation's interface.

Usage

```
get_instantiation_interface_port_properties
```

Returns

String[] The list of port properties.

Arguments

No arguments

Example

```
get_instantiation_interface_port_properties
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.15. get_instantiation_interface_port_property

Description

Returns the port property value of a loaded instantiation's interface.

Usage

`get_instantiation_interface_port_property` <*interface*> <*port*> <*property*>

Returns

various The property value.

Arguments

interface Specifies the interface name.

port Specifies the port name.

property Specifies the property name. Refer to *Port Properties*.

Example

```
get_instantiation_interface_port_property avs_s0 avs_s0_address WIDTH
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- Port Properties on page 663

7.14.5.16. get_instantiation_interface_ports

Description

Returns the list of ports of the loaded instantiation's interface.

Usage

`get_instantiation_interface_ports <interface>`

Returns

String[] The list of port names.

Arguments

interface Specifies the interface name.

Example

```
get_instantiation_interface_ports avs_s0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.17. get_instantiation_interface_property

Description

Returns the value of a single interface property from the specified instantiation interface.

Usage

`get_instantiation_interface_property` <*interface*> <*property*>

Returns

various The property value.

Arguments

interface The interface name on the currently loaded interface.

property The property name. Refer to *Instantiation Interface Properties*.

Example

```
get_instantiation_interface_property in_clk TYPE
```

Related Information

- get_instantiation_interface_properties on page 540
- load_instantiation on page 548
- Instantiation Interface Properties on page 661

7.14.5.18. get_instantiation_interface_properties

Description

Returns the names of all the available instantiation interface properties, common to all interface types.

Usage

```
get_instantiation_interface_properties
```

Returns

String[] A list of instantiation interface properties.

Arguments

No arguments.

Example

```
get_instantiation_interface_properties
```

Related Information

get_instantiation_interface_property on page 539

7.14.5.19. get_instantiation_interface_sysinfo_parameter_value

Description

Gets the system info parameter value for a loaded instantiation's interface.

Usage

`get_instantiation_interface_sysinfo_parameter_value` <*interface*> <*parameter*>

Returns

various The system info property value.

Arguments

interface Specifies the interface name.

parameter Specifies the system info parameter name. Refer to *System Info Type*.

Example

```
get_instantiation_interface_sysinfo_parameter_value debug_mem_agent
max_agent_data_width
```

Related Information

- get_instantiation_interface_sysinfo_parameters on page 542
- set_instantiation_interface_sysinfo_parameter_value on page 558
- System Info Type Properties on page 652

7.14.5.20. get_instantiation_interface_sysinfo_parameters

Description

Returns the list of system info parameters for the loaded instantiation's interface.

Usage

`get_instantiation_interface_sysinfo_parameters` <*interface*> [<*type*>]

Returns

String[] The list of system info parameter names.

Arguments

interface Specifies the interface name.

type (*optional*) Specifies the parameters type to return. If you do not specify this option, the command returns all the parameters. Refer to *Access Type*.

Example

```
get_instantiation_interface_sysinfo_parameters debug_mem_agent
```

Related Information

- get_instantiation_interface_sysinfo_parameter_value on page 541
- set_instantiation_interface_sysinfo_parameter_value on page 558
- Access Type on page 658

7.14.5.21. get_instantiation_interfaces

Description

Returns the list of interfaces for the loaded instantiation.

Usage

```
get_instantiation_interfaces
```

Returns

String[] The list of interface names.

Arguments

No arguments.

Example

```
get_instantiation_interfaces
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.22. get_instantiation_properties

Description

Returns the list of properties for the loaded instantiation.

Usage

```
get_instantiation_properties
```

Returns

String[] The list of property names.

Arguments

No arguments.

Example

```
get_instantiation_properties
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.23. get_instantiation_property

Description

Returns the value of the specified property for the loaded instantiation.

Usage

`get_instantiation_property` <*property*>

Returns

various The value of an instantiation property.

Arguments

property Specifies the property name to get the value of. Refer to *Instantiation Properties*.

Example

```
get_instantiation_property HDL_ENTITY_NAME
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- Instantiation Properties on page 662

7.14.5.24. get_loaded_instantiation

Description

Returns the instance name of the loaded instantiation.

Usage

```
get_loaded_instantiation
```

Returns

String The instance name.

Arguments

No arguments

Example

```
get_loaded_instantiation
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.25. import_instantiation_interfaces

Description

Sets the interfaces of a loaded instantiation by importing the interfaces from the specified file.

Usage

`import_instantiation_interfaces <file>`

Returns

No return value

Arguments

file Specifies the IP or IP-XACT file to import the interfaces from.

Example

```
import_instantiation_interfaces ip/my_system/my_system_nios2_gen2_0.ip
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.26. load_instantiation

Description

Loads the instantiation of an instance, so that you can modify the instantiation if necessary.

Usage

`load_instantiation` <*instance*>

Returns

boolean 1 if successful; 0 if unsuccessful.

Arguments

instance Specifies the instance name.

Example

```
load_instantiation cpu
```

Related Information

save_instantiation on page 552

7.14.5.27. remove_instantiation_hdl_file

Description

Removes an HDL file from the loaded instantiation.

Usage

`remove_instantiation_hdl_file` <*file*> [<*kind*>]

Returns

No return value.

Arguments

file Specifies the HDL file name.

kind (*optional*) Specifies the kind of file set to remove the file from. If you do not specify this option, the command removes the file from all the file sets. Refer to *File Set Kind*.

Example

```
remove_instantiation_hdl_file my_counter.vhdl quartus_synth
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- File Set Kind on page 657

7.14.5.28. remove_instantiation_interface

Description

Removes an interface from a loaded instantiation.

Usage

`remove_instantiation_interface` <*interface*>

Returns

No return value

Arguments

interface Specifies the interface name.

Example

```
remove_instantiation_interface avs_s0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.29. remove_instantiation_interface_port

Description

Removes a port from a loaded instantiation's interface.

Usage

`remove_instantiation_interface_port` <*interface*> <*port*>

Returns

No return value

Arguments

interface Specifies the interface name.

port Specifies the port name.

Example

```
remove_instantiation_interface_port avs_s0 avs_s0_address
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.5.30. save_instantiation

Description

Saves the loaded instantiation.

Usage

```
save_instantiation
```

Returns

No return value

Arguments

No arguments

Example

```
save_instantiation
```

Related Information

load_instantiation on page 548

7.14.5.31. set_instantiation_assignment_value

Description

Sets the assignment value for the loaded instantiation.

Usage

`set_instantiation_assignment_value` <*name*> [<*value*>]

Returns

No return value

Arguments

instance Specifies the assignment name to set value for.

value (*optional*) Specifies the assignment value. If you do not specify this option, the command removes the assignment.

Example

```
set_instantiation_assignment_value embeddedsw.configuration.exceptionOffset 32
```

Related Information

get_instantiation_assignment_value on page 527

7.14.5.32. set_instantiation_hdl_file_property

Description

Sets the property value for an HDL file associated with a loaded instantiation.

Usage

`set_instantiation_hdl_file_property<file> <property> <value>`

Returns

No return value

Arguments

file Specifies the HDL file name.

property Specifies the property name. Refer to *Instantiation Hdl File Properties*.

value Specifies the property value.

Example

```
set_instantiation_hdl_file_property my_nios2_gen2.vhdl OUTPUT_PATH
my_nios2_gen2.vhdl
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- Instantiation HDL File Properties on page 659

7.14.5.33. set_instantiation_interface_assignment_value

Description

Sets the assignment value for the loaded instantiation's interface.

Usage

`set_instantiation_interface_assignment_value` <*interface*> <*name*> [<*value*>]

Returns

No return value

Arguments

interface Specifies the interface name.

name Specifies the assignment name to set the value of.

value (*optional*) Specifies the new assignment value. If you do not specify this value, the command removes the assignment.

Example

```
set_instantiation_interface_assignment_value
embeddedsw.configuration.exceptionOffset 32
```

Related Information

get_instantiation_assignment_value on page 527

7.14.5.34. set_instantiation_interface_parameter_value

Description

Sets the parameter value for the loaded instantiation's interface.

Usage

`set_instantiation_interface_parameter_value` <*interface*> <*parameter*> <*value*>

Returns

No return value

Arguments

instance Specifies the interface name.

parameter Specifies the parameter name.

value Specifies the parameter value.

Example

```
set_instantiation_interface_parameter avs_s0 associatedClock clk
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- get_instantiation_interface_parameter_value on page 534
- get_instantiation_interface_parameters on page 535

7.14.5.35. set_instantiation_interface_port_property

Description

Sets the port property value on a loaded instantiation's interface.

Usage

`set_instantiation_interface_port_property` <*interface*> <*port*> <*property*> <*value*>

Returns

No return value

Arguments

interface Specifies the interface name.

port Specifies the port name.

property Specifies the property name. Refer to *Port Properties*.

value Specifies the property value.

Example

```
set_instantiation_interface_port_property avs_s0 avs_s0_address WIDTH 1
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- Port Properties on page 663

7.14.5.36. set_instantiation_interface_sysinfo_parameter_value

Description

Sets the system info parameter value for the loaded instantiation's interface.

Usage

`set_instantiation_interface_sysinfo_parameter_value` <*interface*> <*parameter*> <*value*>

Returns

No return value

Arguments

interface Specifies the interface name.

parameter Specifies the system info parameter name. Refer to *System Info Type*.

value Specifies the system info parameter value.

Example

```
set_instantiation_interface_sysinfo_parameter_value debug_mem_agent
max_agent_data_width 64
```

Related Information

- get_instantiation_interface_sysinfo_parameter_value on page 541
- get_instantiation_interface_sysinfo_parameters on page 542
- System Info Type Properties on page 652

7.14.5.37. set_instantiation_property

Description

Sets the property value for the loaded instantiation.

Usage

`set_instantiation_property` <*property*> <*value*>

Returns

No return value

Arguments

property Specifies the property name. Refer to *Instantiation Properties*.

value Specifies the value to set.

Example

```
set_instantiation_property HDL_ENTITY_NAME my_system_nios2_gen2_0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- Instantiation Properties on page 662

7.14.6. Components

This section lists the commands that allow you to manipulate the IP components loaded in a Platform Designer system.

7.14.6.1. apply_component_preset

Description

Applies the settings in a preset to the loaded component.

Usage

`apply_component_preset<`*preset_name*`>`

Returns

No return value

Arguments

preset_name Specifies the preset name.

Example

```
apply_component_preset "Custom Debug Settings"
```

Related Information

- load_component on page 581
- set_component_parameter_value on page 584

7.14.6.2. get_component_assignment

Description

Returns the assignment value for the loaded component.

Usage

`get_component_assignment <assignment>`

Returns

String The specified assignment value.

Arguments

assignment Specifies the assignment key value to query.

Example

```
get_component_assignment embeddedsw.CMacro.colorSpace
```

Related Information

- load_component on page 581
- get_component_assignments on page 563

7.14.6.3. get_component_assignments

Description

Returns the list of assignment keys for the loaded component.

Usage

```
get_component_assignments
```

Returns

String[] The list of assignment keys.

Arguments

No arguments

Example

```
get_component_assignments
```

Related Information

- get_instance_assignment on page 495
- load_component on page 581

7.14.6.4. get_component_documentation_links

Description

Returns the list of all documentation links that the loaded component provides.

Usage

```
get_component_documentation_links
```

Returns

String[] The list of documentation links.

Arguments

No arguments

Example

```
get_component_documentation_links
```

Related Information

load_component on page 581

7.14.6.5. get_component_interface_assignment

Description

Returns the assignment value of an interface of the loaded component.

Usage

`get_component_interface_assignment` <*interface*> <*assignment*>

Returns

String The specified assignment value.

Arguments

interface Specifies the interface name.

assignment Specifies the assignment key to the query.

Example

```
get_component_interface_assignment s1 embeddedsw.configuration.isFlash
```

Related Information

- get_component_interface_assignments on page 566
- load_component on page 581

7.14.6.6. get_component_interface_assignments

Description

Returns the list of assignment keys for any assignments that you define for an interface on the loaded component.

Usage

`get_component_interface_assignments` <*interface*>

Returns

String[] The list of assignment keys.

Arguments

interface Specifies the interface name.

Example

```
get_component_interface_assignments s1
```

Related Information

- get_component_interface_assignment on page 565
- load_component on page 581

7.14.6.7. get_component_interface_parameter_property

Description

Returns the property value of a parameter in a loaded component's interface. Parameter properties are metadata about how the Quartus Prime uses the parameters.

Usage

`get_component_interface_parameter_property` <*interface*> <*parameter*> <*property*>

Returns

various The parameter property value.

Arguments

interface Specifies the interface name.

parameter Specifies the parameter name.

property Specifies the parameter property. Refer to *Parameter Properties*.

Example

```
get_component_interface_parameter_property s0 setupTime ENABLED
```

Related Information

- get_component_interface_parameters on page 569
- get_component_interfaces on page 573
- load_component on page 581
- Parameter Properties on page 646
- get_parameter_properties on page 627

7.14.6.8. get_component_interface_parameter_value

Description

Returns the parameter value of an interface of the loaded component.

Usage

`get_component_interface_parameter_value` <*interface*> <*parameter*>

Returns

various The parameter value.

Arguments

interface Specifies the interface name.

parameter Specifies the parameter name.

Example

```
get_component_interface_parameter_value s0 setupTime
```

Related Information

- get_component_interface_parameters on page 569
- get_component_interfaces on page 573
- load_component on page 581

7.14.6.9. get_component_interface_parameters

Description

Returns the list of parameters for an interface of the loaded component.

Usage

`get_component_interface_parameters` <*interface*>

Returns

String[] The list of parameter names.

Arguments

interface Specifies the interface name.

Example

```
get_component_interface_parameters s0
```

Related Information

- get_component_interface_parameter_value on page 568
- get_component_interfaces on page 573
- load_component on page 581

7.14.6.10. get_component_interface_port_property

Description

Returns the property value of a port in the interface of the loaded component.

Usage

`get_component_interface_port_property` <*interface*> <*port*> <*property*>

Returns

various The port property value

Arguments

interface Specifies the interface name.

port Specifies the port name of the interface.

property Specifies the property name of the port. Refer to *Port Properties*.

Example

```
get_component_interface_port_property exports tx WIDTH
```

Related Information

- get_component_interface_ports on page 571
- load_component on page 581
- Port Properties on page 663
- get_port_properties on page 607

7.14.6.11. get_component_interface_ports

Description

Returns the list of interface ports of the loaded component.

Usage

`get_component_interface_ports` <*interface*>

Returns

String[] The list of port names

Arguments

interface Specifies the interface name.

Example

```
get_component_interface_ports s0
```

Related Information

- get_component_interface_port_property on page 570
- get_component_interfaces on page 573
- load_component on page 581

7.14.6.12. get_component_interface_property

Description

Returns the value of a single property from the specified interface for the loaded component.

Usage

`get_component_interface_property` <*interface*> <*property*>

Returns

String The property value.

Arguments

interface Specifies the interface name.

property Specifies the property name. Refer to *Element Properties*.

Example

```
get_interface_property clk_in DISPLAY_NAME
```

Related Information

- load_component on page 581
- Element Properties on page 641
- get_interface_properties on page 604

7.14.6.13. get_component_interfaces

Description

Returns the list of interfaces in the loaded component.

Usage

```
get_component_interfaces
```

Returns

String[] The list of interface names.

Arguments

No arguments

Example

```
get_component_interfaces
```

Related Information

- get_component_interface_ports on page 571
- get_component_interface_property on page 572
- load_component on page 581

7.14.6.14. get_component_parameter_property

Description

Returns the property value of a parameter in the loaded component.

Usage

`get_component_parameter_property` <*parameter*> <*property*>

Returns

Various The parameter property value.

Arguments

parameter Specifies the parameter name in the component.

property Specifies the property name of the parameter. Refer to *Parameter Properties*.

Example

```
get_component_parameter_property baudRate ENABLED
```

Related Information

- get_component_parameters on page 576
- get_parameter_properties on page 627
- load_component on page 581
- Parameter Properties on page 646

7.14.6.15. get_component_parameter_value

Description

Returns the parameter value in the loaded component.

Usage

`get_component_parameter_value <parameter>`

Returns

various The parameter value

Arguments

parameter Specifies the parameter name in the component.

Example

```
get_component_parameter_value baudRate
```

Related Information

- get_component_parameters on page 576
- load_component on page 581
- set_component_parameter_value on page 584

7.14.6.16. get_component_parameters

Description

Returns the list of parameters in the loaded component.

Usage

```
get_component_parameters
```

Returns

String[] The list of parameters in the component.

Arguments

No arguments

Example

```
get_instance_parameters
```

Related Information

- get_component_parameter_property on page 574
- get_component_parameter_value on page 575
- load_component on page 581
- set_component_parameter_value on page 584

7.14.6.17. get_component_project_properties

Description

Returns the list of properties that you query about the loaded component's Quartus Prime project.

Usage

```
get_component_project_properties
```

Returns

String[] The list of project properties.

Arguments

No arguments

Example

```
get_component_project_properties
```

Related Information

- get_component_project_property on page 578
- load_component on page 581
- set_component_project_property on page 585

7.14.6.18. get_component_project_property

Description

Returns the project property value of the loaded component. Only select project properties are available.

Usage

`get_component_project_property` <*property*>

Returns

String The property value.

Arguments

property Specifies the project property name. Refer to *Project Properties*.

Example

```
get_component_project_property HIDE_FROM_IP_CATALOG
```

Related Information

- get_component_project_properties on page 577
- load_component on page 581
- set_component_project_property on page 585
- Project Properties on page 651

7.14.6.19. get_component_property

Description

Returns the property value of the loaded component.

Usage

`get_component_property <property>`

Returns

String The property value.

Arguments

property The property name on the loaded component. Refer to *Element Properties*.

Example

```
get_component_property CLASS_NAME
```

Related Information

- load_component on page 581
- get_instance_properties on page 513
- Element Properties on page 641

7.14.6.20. get_loaded_component

Description

Returns the instance name associated with the loaded component.

Usage

```
get_loaded_component
```

Returns

String The instance name.

Arguments

No arguments

Example

```
get_loaded_component
```

Related Information

- load_component on page 581
- save_component on page 583

7.14.6.21. load_component

Description

Loads the actual component inside of a generic component, so that you can modify the component parameters.

Usage

`load_component <instance>`

Returns

boolean 1 if successful; 0 if unsuccessful.

Arguments

instance Specifies the instance name.

Example

```
load_component cpu
```

Related Information

- get_loaded_component on page 580
- save_component on page 583

7.14.6.22. reload_component_footprint

Description

Validates the footprint of a specified child instance, and updates the footprint of the instance in case of issues.

Usage

`reload_component_footprint [<instance>]`

Returns

String[] A list of validation messages.

Arguments

instance (*optional*)	Specifies the child instance name to validate. If you do not specify this option, the command validates all the generic components in the system.

Example

```
reload_component_footprint cpu_0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- validate_component_footprint on page 621

7.14.6.23. save_component

Description

Saves the loaded component.

Usage

```
save_component
```

Returns

No return value

Arguments

No arguments

Example

```
save_component
```

Related Information

- get_loaded_component on page 580
- load_component on page 581

7.14.6.24. set_component_parameter_value

Description

Sets the parameter value for the loaded component.

Usage

`set_component_parameter_value` <*parameter*> <*value*>

Returns

No return value

Arguments

parameter Specifies the parameter name.

parameter Specifies the new parameter value.

Example

```
set_component_parameter_value baudRate 9600
```

Related Information

- get_component_parameter_value on page 575
- get_component_parameters on page 576
- load_component on page 581

7.14.6.25. set_component_project_property

Description

Sets the project property value of the loaded component, such as hiding from the IP catalog.

Usage

`set_component_project_property` <*property*> <*value*>

Returns

No return value

Arguments

property Specifies the property name. Refer to *Project Properties*.

value Specifies the new property value.

Example

```
set_component_project_property HIDE_FROM_IP_CATALOG false
```

Related Information

- get_component_project_properties on page 577
- get_component_project_property on page 578
- load_component on page 581
- Project Properties on page 651

7.14.7. Connections

This section lists the commands that allow you to manipulate the interface connections in your Platform Designer system.

7.14.7.1. add_connection

Description

Connects the named interfaces using an appropriate connection type. Both interface names consist of an instance name, followed by the interface name that the module provides.

Usage

`add_connection <start> [<end>]`

Returns

No return value.

Arguments

start The start interface that you connect, in `<instance_name>.<interface_name>` format. If you do not specify the end argument, the connection must be of the form `<instance1>.<interface>/<instance2>.<interface>`.

end (optional) The end interface that you connect, in `<instance_name>.<interface_name>` format.

Example

```
add_connection dma.read_host sdram.s1
```

Related Information

- get_connection_parameter_value on page 590
- get_connection_property on page 593
- get_connections on page 594
- remove_connection on page 595
- set_connection_parameter_value on page 597

7.14.7.2. auto_connect

Description

Creates connections from an instance or instance interface to matching interfaces of other instances in the system. For example, Avalon memory mapped agents connect to Avalon memory mapped hosts.

Usage

`auto_connect <element>`

Returns

No return value.

Arguments

element The instance interface name, or the instance name.

Example

```
auto_connect sdram
auto_connect uart_0.s1
```

Related Information

add_connection on page 587

7.14.7.3. get_connection_parameter_property

Description

Returns the property value of a parameter in a connection. Parameter properties are metadata about how Platform Designer uses the parameter.

Usage

`get_connection_parameter_property` <*connection*> <*parameter*> <*property*>

Returns

various The parameter property value.

Arguments

connection The connection to query.

parameter The parameter name.

property The property of the connection. Refer to *Parameter Properties*.

Example

```
get_connection_parameter_property cpu.data_host/dma0.csr baseAddress UNITS
```

Related Information

- get_connection_parameter_value on page 590
- get_connection_property on page 593
- get_connections on page 594
- get_parameter_properties on page 627
- Parameter Properties on page 646

7.14.7.4. get_connection_parameter_value

Description

Returns the parameter value of the connection. Parameters represent aspects of the connection that you can modify, such as the base address for an Avalon memory mapped connection.

Usage

`get_connection_parameter_value` <*connection*> <*parameter*>

Returns

various The parameter value.

Arguments

connection The connection to query.

parameter The parameter name.

Example

```
get_connection_parameter_value cpu.data_host/dma0.csr baseAddress
```

Related Information

- get_connection_parameters on page 591
- get_connections on page 594
- set_connection_parameter_value on page 597

7.14.7.5. get_connection_parameters

Description

Returns the list of parameters of a connection.

Usage

`get_connection_parameters` <*connection*>

Returns

String[] The list of parameter names.

Arguments

connection The connection to query.

Example

```
get_connection_parameters cpu.data_host/dma0.csr
```

Related Information

- get_connection_parameter_property on page 589
- get_connection_parameter_value on page 590
- get_connection_property on page 593

7.14.7.6. get_connection_properties

Description

Returns the properties list of a connection.

Usage

```
get_connection_properties
```

Returns

String[] The list of connection properties.

Arguments

No arguments.

Example

```
get_connection_properties
```

Related Information

- get_connection_property on page 593
- get_connections on page 594

7.14.7.7. get_connection_property

Description

Returns the property value of a connection. Properties represent aspects of the connection that you can modify, such as the connection type.

Usage

`get_connection_property` <*connection*> <*property*>

Returns

String The connection property value.

Arguments

connection The connection to query.

property The connection property name. Refer to *Connection Properties*.

Example

```
get_connection_property cpu.data_host/dma0.csr TYPE
```

Related Information

- get_connection_properties on page 592
- Connection Properties on page 638

7.14.7.8. get_connections

Description

Returns the list of all connections in the system if you do not specify any element. If you specify a child instance, for example `cpu`, Platform Designer returns all connections to any interface on the instance. If you specify an interface of a child instance, for example `cpu.instruction_host`, Platform Designer returns all connections to that interface.

Usage

`get_connections` [<*element*>]

Returns

String[] The list of connections.

Arguments

element (optional) The child instance name, or the qualified interface name on a child instance.

Example

```
get_connections
get_connections cpu
get_connections cpu.instruction_host
```

Related Information

- add_connection on page 587
- remove_connection on page 595

7.14.7.9. remove_connection

Description

Removes a connection from the system.

Usage

`remove_connection` <*connection*>

Returns

No return value.

Arguments

connection The connection name to remove.

Example

```
remove_connection cpu.data_host/sdram.s0
```

Related Information

- add_connection on page 587
- get_connections on page 594

7.14.7.10. remove_dangling_connections

Description

Removes connections where both end points of the connection no longer exist in the system.

Usage

```
remove_dangling_connections
```

Returns

No return value.

Arguments

No arguments.

Example

```
remove_dangling_connections
```

Related Information

- add_connection on page 587
- get_connections on page 594
- remove_connection on page 595

7.14.7.11. set_connection_parameter_value

Description

Sets the parameter value for a connection.

Usage

`set_connection_parameter_value` <*connection*> <*parameter*> <*value*>

Returns

No return value.

Arguments

connection The connection name.

parameter The parameter name.

value The new parameter value.

Example

```
set_connection_parameter_value cpu.data_host/dma0.csr baseAddress "0x000a0000"
```

Related Information

- get_connection_parameter_value on page 590
- get_connection_parameters on page 591

7.14.8. Top-level Exports

This section lists the commands that allow you to manipulate the exported interfaces in your Platform Designer system.

7.14.8.1. add_interface

Description

Adds an interface to your system, which Platform Designer uses to export an interface from within the system. You specify the exported internal interface with `set_interface_property <interface> EXPORT_OF instance.interface`.

Usage

`add_interface <name> <type> <direction>`.

Returns

No return value.

Arguments

name The name of the interface that Platform Designer exports from the system.

type The type of interface.

direction The interface direction.

Example

```
add_interface my_export conduit end
set_interface_property my_export EXPORT_OF uart_0.external_connection
```

Related Information

- get_interface_ports on page 603
- get_interface_properties on page 604
- get_interface_property on page 605
- set_interface_property on page 611

7.14.8.2. get_exported_interface_sysinfo_parameter_value

Description

Gets the value of a system info parameter for an exported interface.

Usage

`get_exported_interface_sysinfo_parameter_value` <*interface*> <*parameter*>

Returns

various The system info parameter value.

Arguments

interface Specifies the name of the exported interface.

parameter Specifies the name of the system info parameter. Refer to *System Info Type*.

Example

```
get_exported_interface_sysinfo_parameter_value clk clock_rate
```

Related Information

- get_exported_interface_sysinfo_parameters on page 601
- set_exported_interface_sysinfo_parameter_value on page 609
- System Info Type Properties on page 652

7.14.8.3. get_exported_interface_sysinfo_parameters

Description

Returns the list of system info parameters for an exported interface.

Usage

`get_exported_interface_sysinfo_parameters` <*interface*> [<*type*>]

Returns

String[] The list of system info parameter names.

Arguments

interface Specifies the name of the exported interface.

type (*optional*) Specifies the parameters type to return. If you do not specify this option, the command returns all the parameters. Refer to *Access Type*.

Example

```
get_exported_interface_sysinfo_parameters clk
```

Related Information

- get_exported_interface_sysinfo_parameter_value on page 600
- set_exported_interface_sysinfo_parameter_value on page 609
- Access Type on page 658

7.14.8.4. get_interface_port_property

Description

Returns the value of a property of a port contained by one of the top-level exported interfaces.

Usage

`get_interface_port_property` <*interface*> <*port*> <*property*>

Returns

various The property value.

Arguments

interface The name of a top-level interface of the system.

port The port name in the interface.

property The property name on the port. Refer to *Port Properties*.

Example

```
get_interface_port_property uart_exports tx DIRECTION
```

Related Information

- get_interface_ports on page 603
- get_port_properties on page 607
- Port Properties on page 650

7.14.8.5. get_interface_ports

Description

Returns the names of all the added ports to a given interface.

Usage

`get_interface_ports` <*interface*>

Returns

String[] The list of port names.

Arguments

interface The top-level interface name of the system.

Example

```
get_interface_ports export_clk_out
```

Related Information

- get_interface_port_property on page 602
- get_interfaces on page 606

7.14.8.6. get_interface_properties

Description

Returns the names of all the available interface properties common to all interface types.

Usage

```
get_interface_properties
```

Returns

String[] The list of interface properties.

Arguments

No arguments.

Example

```
get_interface_properties
```

Related Information

- get_interface_property on page 605
- set_interface_property on page 611

7.14.8.7. get_interface_property

Description

Returns the value of a single interface property from the specified interface.

Usage

`get_interface_property` <*interface*> <*property*>

Returns

various The property value.

Arguments

interface The name of a top-level interface of the system.

property The name of the property. Refer to *Interface Properties*.

Example

```
get_interface_property export_clk_out EXPORT_OF
```

Related Information

- get_interface_properties on page 604
- set_interface_property on page 611
- Interface Properties on page 643

7.14.8.8. get_interfaces

Description

Returns the list of top-level interfaces in the system.

Usage

```
get_interfaces
```

Returns

String[] The list of the top-level interfaces exported from the system.

Arguments

No arguments.

Example

```
get_interfaces
```

Related Information

- add_interface on page 599
- get_interface_ports on page 603
- get_interface_property on page 605
- remove_interface on page 608
- set_interface_property on page 611

7.14.8.9. get_port_properties

Description

Returns the list of properties that you can query for ports.

Usage

```
get_port_properties
```

Returns

String[] The list of port properties.

Arguments

No arguments.

Example

```
get_port_properties
```

Related Information

- get_instance_interface_port_property on page 503
- get_instance_interface_ports on page 504
- get_instance_port_property on page 512
- get_interface_port_property on page 602
- get_interface_ports on page 603

7.14.8.10. remove_interface

Description

Removes an exported top-level interface from the system.

Usage

`remove_interface` <*interface*>

Returns

No return value.

Arguments

interface The name of the exported top-level interface.

Example

```
remove_interface clk_out
```

Related Information

- add_interface on page 599
- get_interfaces on page 606

7.14.8.11. set_exported_interface_sysinfo_parameter_value

Description

Sets the system info parameter value for an exported interface.

Usage

`set_exported_interface_sysinfo_parameter_value` <*interface*> <*parameter*> <*value*>

Returns

No return value

Arguments

interface Specifies the name of the exported interface.

parameter Specifies the name of the system info parameter. Refer to *System Info Type*.

value Specifies the system info parameter value.

Example

```
set_exported_interface_sysinfo_parameter_value clk clock_rate 5000000
```

Related Information

- get_exported_interface_sysinfo_parameter_value on page 600
- get_exported_interface_sysinfo_parameters on page 601
- System Info Type Properties on page 652

7.14.8.12. set_interface_port_property

Description

Sets the port property in a top-level interface of the system.

Usage

`set_interface_port_property` <*interface*> <*port*> <*property*> <*value*>

Returns

No return value

Arguments

interface Specifies the top-level interface name of the system.

port Specifies the port name in a top-level interface of the system.

property Specifies the property name of the port. Refer to *Port Properties*.

value Specifies the property value.

Example

```
set_interface_port_property clk clk_clk NAME my_clk
```

Related Information

- Port Properties on page 663
- get_interface_ports on page 603
- get_interfaces on page 606
- get_port_properties on page 607

7.14.8.13. set_interface_property

Description

Sets the value of a property on an exported top-level interface. You use this command to set the EXPORT_OF property to specify which interface of a child instance is exported via this top-level interface.

Usage

`set_interface_property` <*interface*> <*property*> <*value*>

Returns

No return value.

Arguments

interface The name of an exported top-level interface.

property The name of the property. Refer to *Interface Properties*.

value The property value.

Example

```
set_interface_property clk_out EXPORT_OF clk.clk_out
```

Related Information

- add_interface on page 599
- get_interface_properties on page 604
- get_interface_property on page 605
- Interface Properties on page 643

7.14.9. Validation

This section lists the commands that allow you to validate the components, instances, interfaces and connections in a Platform Designer system.

7.14.9.1. set_validation_property

Description

Sets a property that affects how and when validation is run. To disable system validation after each scripting command, set `AUTOMATIC_VALIDATION` to `False`.

Usage

`set_validation_property` <*property*> <*value*>

Returns

No return value.

Arguments

property The name of the property. Refer to *Validation Properties*.

value The new property value.

Example

```
set_validation_property AUTOMATIC_VALIDATION false
```

Related Information

- validate_system on page 620
- Validation Properties on page 655

7.14.9.2. sync_sysinfo_parameters

Description

Updates the system info parameters of the specified generic component.

Usage

`sync_sysinfo_parameters [<instance> ]`

Returns

String[] A list of update messages.

Arguments

instance (*optional*)	Specifies the name of the instance to sync. If you do not specify this option, the command synchronizes all the generic components in the system.

Example

```
sync_sysinfo_parameters cpu_0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.9.3. validate_component

Description

Validates the loaded component.

Usage

```
validate_component
```

Returns

String[] A list of validation messages.

Arguments

No arguments

Example

```
validate_component
```

Related Information

- validate_component_interface on page 616
- load_component on page 581

7.14.9.4. validate_component_interface

Description

Validates an interface of the loaded component.

Usage

`validate_component_interface <interface>`

Returns

String[] List of validation messages

Arguments

instance Specifies the name of the instance for the loaded component.

Example

```
validate_instance_interface data_host
```

Related Information

- load_component on page 581
- validate_component on page 615

7.14.9.5. validate_connection

Description

Validates the specified connection and returns validation messages.

Usage

`validate_connection <connection>`

Returns

A list of validation messages.

Arguments

connection The connection name to validate.

Example

```
validate_connection cpu.data_host/sdram.s1
```

Related Information

- validate_instance on page 618
- validate_instance_interface on page 619
- validate_system on page 620

7.14.9.6. validate_instance

Description

Validates the specified child instance and returns validation messages.

Usage

`validate_instance` <*instance*>

Returns

A list of validation messages.

Arguments

instance The child instance name to validate.

Example

```
validate_instance cpu
```

Related Information

- validate_connection on page 617
- validate_instance_interface on page 619
- validate_system on page 620

7.14.9.7. validate_instance_interface

Description

Validates an interface of an instance and returns validation messages.

Usage

`validate_instance_interface` <*instance*> <*interface*>

Returns

A list of validation messages.

Arguments

instance The child instance name.

interface The interface to validate.

Example

```
validate_instance_interface cpu data_host
```

Related Information

- validate_connection on page 617
- validate_instance on page 618
- validate_system on page 620

7.14.9.8. validate_system

Description

Validates the system and returns validation messages.

Usage

```
validate_system
```

Returns

A list of validation messages.

Arguments

No arguments.

Example

```
validate_system
```

Related Information

- validate_connection on page 617
- validate_instance on page 618
- validate_instance_interface on page 619

7.14.9.9. validate_component_footprint

Description

Validates the footprint of the specified child instance.

Usage

`validate_component_footprint` <*instance*>

Returns

String[] List of validation messages.

Arguments

instance (*optional*)	Specifies the child instance name. If you omit this option, the command validates all generic components in the system.

Example

```
validate_component_footprint cpu_0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552

7.14.9.10. reload_component_footprint

Description

Validates the footprint of a specified child instance, and updates the footprint of the instance in case of issues.

Usage

`reload_component_footprint [<instance>]`

Returns

String[] A list of validation messages.

Arguments

instance (*optional*)	Specifies the child instance name to validate. If you do not specify this option, the command validates all the generic components in the system.

Example

```
reload_component_footprint cpu_0
```

Related Information

- load_instantiation on page 548
- save_instantiation on page 552
- validate_component_footprint on page 621

7.14.10. Miscellaneous

This section lists the miscellaneous commands that you can use for your Platform Designer systems.

7.14.10.1. auto_assign_base_addresses

Description

Assigns base addresses to all memory-mapped interfaces of an instance in the system. Instance interfaces that are locked with `lock_avalon_base_address` keep their addresses during address auto-assignment.

Usage

`auto_assign_base_addresses` <*instance*>

Returns

No return value.

Arguments

instance The name of the instance with memory-mapped interfaces.

Example

```
auto_assign_base_addresses sdram
```

Related Information

- auto_assign_system_base_addresses on page 626
- lock_avalon_base_address on page 628
- unlock_avalon_base_address on page 631

7.14.10.2. auto_assign_irqs

Description

Assigns interrupt numbers to all connected interrupt senders of an instance in the system.

Usage

`auto_assign_irqs <`*instance*`>`

Returns

No return value.

Arguments

instance The name of the instance with an interrupt sender.

Example

```
auto_assign_irqs uart_0
```

7.14.10.3. auto_assign_system_base_addresses

Description

Assigns legal base addresses to all memory-mapped interfaces of all instances in the system. Instance interfaces that are locked with `lock_avalon_base_address` keep their addresses during address auto-assignment.

Usage

`auto_assign_system_base_addresses`

Returns

No return value.

Arguments

No arguments.

Example

```
auto_assign_system_base_addresses
```

Related Information

- auto_assign_base_addresses on page 624
- lock_avalon_base_address on page 628
- unlock_avalon_base_address on page 631

7.14.10.4. get_parameter_properties

Description

Returns the list of properties that you can query for any parameters, for example parameters of instances, interfaces, instance interfaces, and connections.

Usage

```
get_parameter_properties
```

Returns

String[] The list of parameter properties.

Arguments

No arguments.

Example

```
get_parameter_properties
```

Related Information

- get_connection_parameter_property on page 589
- get_instance_interface_parameter_property on page 500
- get_instance_parameter_property on page 508

7.14.10.5. lock_avalon_base_address

Description

Prevents the memory-mapped base address from being changed for connections to the specified interface of an instance when Platform Designer runs the `auto_assign_base_addresses` or `auto_assign_system_base_addresses` commands.

Usage

`lock_avalon_base_address` <*instance.interface*>

Returns

No return value.

Arguments

instance.interface	The qualified name of the interface of an instance, in <*instance*>.<*interface*> format.

Example

```
lock_avalon_base_address sdram.s1
```

Related Information

- auto_assign_base_addresses on page 624
- auto_assign_system_base_addresses on page 626
- unlock_avalon_base_address on page 631

7.14.10.6. send_message

Description

Sends a message to the user of the component. The message text is normally HTML. You can use the <*b*> element to provide emphasis. If you do not want the message text to be HTML, then pass a list like `{ Info Text }` as the message level,

Usage

`send_message` <*level*> <*message*>

Returns

No return value.

Arguments

level Quartus Prime supports the following message levels:

- `ERROR`—provides an error message.
- `WARNING`—provides a warning message.
- `INFO`—provides an informational message.
- `PROGRESS`—provides a progress message.
- `DEBUG`—provides a debug message when debug mode is enabled.

message The text of the message.

Example

```
send_message ERROR "The system is down!"
send_message { Info Text } "The system is up!"
```

7.14.10.7. set_use_testbench_naming_pattern

Description

Use this command to create testbench systems so that the generated file names for the test system match the system's original generated file names. Without setting this command, the generated file names for the test system receive the top-level testbench system name.

Usage

`set_use_testbench_naming_pattern` <*value*>

Returns

No return value.

Arguments

value True or false.

Example

```
set_use_testbench_naming_pattern true
```

Notes

Use this command only to create testbench systems.

7.14.10.8. unlock_avalon_base_address

Description

Allows the memory-mapped base address to change for connections to the specified interface of an instance when Platform Designer runs the `auto_assign_base_addresses` or `auto_assign_system_base_addresses` commands.

Usage

`unlock_avalon_base_address` <*instance.interface*>

Returns

No return value.

Arguments

instance.interface	The qualified name of the interface of an instance, in <*instance*>.<*interface*> format.

Example

```
unlock_avalon_base_address sdram.s1
```

Related Information

- auto_assign_base_addresses on page 624
- auto_assign_system_base_addresses on page 626
- lock_avalon_base_address on page 628

7.14.10.9. get_testbench_dutname

Description

Returns the currently set dutname for the test-bench systems. Use this command only when creating test-bench systems.

Usage

```
get_testbench_dutname
```

Returns

String The currently set dutname. Returns NULL if empty.

Arguments

No arguments.

Example

```
get_testbench_dutname
```

Related Information

- get_use_testbench_naming_pattern on page 633
- set_use_testbench_naming_pattern on page 630

7.14.10.10. get_use_testbench_naming_pattern

Description

Verifies if the test-bench naming pattern is set to be used. Use this command only when creating test-bench systems.

Usage

```
get_use_testbench_naming_pattern
```

Returns

boolean True, if the test-bench naming pattern is set to be used.

Arguments

No arguments.

Example

```
get_use_testbench_naming_pattern
```

Related Information

- get_testbench_dutname on page 632
- set_use_testbench_naming_pattern on page 630

7.14.11. Wire-Level Connection Commands

Wire-level commands accept optional input ports and wire-level expressions as arguments for the `qsys-script` utility and in `_hw.tcl` files.

You can use wire-level commands to:

- Apply a wire-level expression to a port with `set_wirelevel_expression`.
- Retrieve a list of expressions from a port, instance, or all expressions in the current level of system hierarchy with `get_wirelevel_expression`.
- Remove a list of expressions from a port, instance, or all expressions in the current level of system hierarchy with `remove_wirelevel_expression`.

Note: The following restrictions apply when using wire-level commands `_hw.tcl` files:

- Wire-level commands are only valid in a composition callback.
- Wire-level expressions can only be applied to instances created by `add_instance`.

Related Information

- Scripting Wire-Level Expressions on page 71
- Create a Composed Component or Subsystem on page 196

7.14.11.1. set_wirelevel_expression

Description

Applies a wire-level expression to an optional input port or instance in the system.

Usage

`set_wirelevel_expression` <*instance_or_port_bitselection*> <*expression*>

Returns

No return value.

Arguments

instance_or_port_bitselection	Specify the instance or port to which the wire-level expression using the <*instance_name*>.<*port_name*>[<*bit_selection*>] format. The *bit selection* can be a bit-select, for example `[0]`, or a partial range defined in descending order, for example `[7:0]`. If no *bit selection* is specified, the full range of the port is selected.
expression	The expression to be applied to an optional input port.

Examples

```
set_wirelevel_expression {module0.portA[7:0]} "8'b0"
set_wirelevel_expression module0.portA "8'b0"
set_wirelevel_expression {module0.portA[0]} "1'b0"
```

Related Information

Scripting Wire-Level Expressions on page 71

7.14.11.2. get_wirelevel_expressions

Description

Retrieve a list of wire-level expressions from an optional input port, instance, or all expressions in the current level of system hierarchy. If the port *bit selection* is specified as an argument, the range must be identical to what was used in the `set_wirelevel_expression` statement.

Usage

`get_wirelevel_expressions` <*instance_or_port_bitselection*>

Returns

String[]	A flattened list of wire-level expressions. Every item in the list consists of right- and left-hand clauses of a wire-level expression. You can loop over the returned list using `foreach{port expr} $return_list{}`.

Arguments

instance_or_port_bitselection	Specifies which instance or port from which a list of wire-level expressions are retrieved using the <*instance_name*>.<*port_name*>[<*bit_selection*>] format. • If no <*port_name*>[<*bit_selection*>] is specified, the command causes the return of all expressions from the specified instance. • If no argument is present, the command causes the return of all expressions from the current level of system hierarchy. The *bit selection* can be a bit-select, for example `[0]`, or a partial range defined in descending order, for example `[7:0]`. If no *bit selection* is specified, the full range of the port is selected.

Example

```
get_wirelevel_expressions
get_wirelevel_expressions module0
get_wirelevel_expressions {module0.portA[7:0]}
```

Related Information

Scripting Wire-Level Expressions on page 71

7.14.11.3. remove_wirelevel_expressions

Description

Remove a list of wire-level expressions from an optional input port, instance, or all expressions in the current level of system hierarchy. If the port *bit selection* is specified as an argument, the range must be identical to what was used in the `set_wirelevel_expressions` statement.

Usage

`remove_wirelevel_expressions` <*instance_or_port_bitselection*>

Returns

No return value.

Arguments

instance_or_port_bitselection	Specifies which instance or port from which a list of wire-level expressions are removed using the <*instance_name*>.<*port_name*>[<*bit_selection*>] format.

- If no *<port_name>[<bit_selection>]* is specified, the command causes the removal of all expressions from the specified instance.
- If no argument is present, the command causes the return of all expressions from the current level of system hierarchy.

The *bit selection* can be a bit-select, for example `[0]`, or a partial range defined in descending order, for example `[7:0]`. If no *bit selection* is specified, the full range of the port is selected.

Examples

```
remove_wirelevel_expressions
remove_wirelevel_expressions module0
remove_wirelevel_expressions {module0.portA[7:0]}
```

Related Information

Scripting Wire-Level Expressions on page 71

7.15. Platform Designer Scripting Property Reference

Interface properties work differently for **_hw.tcl** scripting than with Platform Designer scripting. In **_hw.tcl**, interfaces do not distinguish between properties and parameters. In Platform Designer scripting, the properties and parameters are unique.

The following are the Platform Designer scripting properties:

7.15.1. Connection Properties

Type	Name	Description
string	END	Indicates the end interface of the connection.
string	NAME	Indicates the name of the connection.
string	START	Indicates the start interface of the connection.
String	TYPE	The type of the connection.

7.15.2. Design Environment Type Properties

Description

IP cores use the design environment to identify the most appropriate interfaces to connect to the parent system.

Name	Description
NATIVE	Supports native IP interfaces.
QSYS	Supports standard Platform Designer interfaces.

7.15.3. Direction Properties

Name	Description
BIDIR	Indicates the direction for a bidirectional signal.
INOUT	Indicates the direction for an input signal.
OUTPUT	Indicates the direction for an output signal.

7.15.4. Element Properties

Description

Element properties are, with the exception of ENABLED and NAME, read-only properties of the types of instances, interfaces, and connections. These read-only properties represent metadata that does not vary between copies of the same type. ENABLED and NAME properties are specific to particular instances, interfaces, or connections.

Type	Name	Description
String	AUTHOR	The author of the component or interface.
Boolean	AUTO_EXPORT	Indicates whether unconnected interfaces on the instance are automatically exported.
String	CLASS_NAME	The type of the instance, interface or connection, for example, altera_nios2.
String	DESCRIPTION	The description of the instance, interface or connection type.
String	DISPLAY_NAME	The display name for referencing the type of instance, interface or connection.
Boolean	EDITABLE	Indicates whether you can edit the component in the Platform Designer Component Editor.
Boolean	ENABLED	Indicates whether the instance is enabled.
String	GROUP	The IP Catalog category.
Boolean	INTERNAL	Hides internal IP components or sub-components from the IP Catalog..
String	NAME	The name of the instance, interface or connection.
String	VERSION	The version number of the instance, interface or connection, for example, 16.1.

7.15.5. Instance Properties

Type	Name	Description
String	AUTO_EXPORT	Indicates whether Platform Designer automatically exports the unconnected interfaces on the instance.
Boolean	ENABLED	If true, Platform Designer includes this instance in the generated system.
String	NAME	The name of the system, which Platform Designer uses as the name of the top-level module in the generated HDL.

7.15.6. Interface Properties

Type	Name	Description
String	EXPORT_OF	Indicates which interface of a child instance to export through the top-level interface. Before using this command, you must create the top-level interface using the `add_interface` command. You must use the format: <*instanceName.interfaceName*>. For example: `set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port`

7.15.7. Message Levels Properties

Name	Description
COMPONENT_INFO	Reports an informational message only during component editing.
DEBUG	Provides messages when debug mode is enabled.
ERROR	Provides an error message.
INFO	Provides an informational message.
PROGRESS	Reports progress during generation.
TODOERROR	Provides an error message that indicates the system is incomplete.
WARNING	Provides a warning message.

7.15.8. Module Properties

Type	Name	Description
String	GENERATION_ID	The generation ID for the system.
String	NAME	The name of the instance.

7.15.9. Parameter Properties

Type	Name	Description
Boolean	AFFECTS_ELABORATION	Set AFFECTS_ELABORATION to false for parameters that do not affect the external interface of the module. An example of a parameter that does not affect the external interface is isNonVolatileStorage. An example of a parameter that does affect the external interface is width. When the value of a parameter changes and AFFECTS_ELABORATION is false, the elaboration phase does not repeat and improves performance. When AFFECTS_ELABORATION is set to true, the default value, Platform Designer reanalyzes the HDL file to determine the port widths and configuration each time a parameter changes.
Boolean	AFFECTS_GENERATION	The default value of AFFECTS_GENERATION is false if you provide a top-level HDL module. The default value is true if you provide a fileset callback. Set AFFECTS_GENERATION to false if the value of a parameter does not change the results of fileset generation.
Boolean	AFFECTS_VALIDATION	The AFFECTS_VALIDATION property determines whether a parameter's value sets derived parameters, and whether the value affects validation messages. Setting this property to false may improve response time in the parameter editor when the value changes.
String[]	ALLOWED_RANGES	Indicates the range or ranges of the parameter. For integers, each range is a single value, or a range of values defined by a start and end value, and delimited by a colon, for example, 11:15. This property also specifies the legal values and description strings for integers, for example, {0:None 1:Monophonic 2:Stereo 4:Quadrophonic}, where 0, 1, 2, and 4 are the legal values. You can assign description strings in the parameter editor for string variables. For example, `ALLOWED_RANGES {"dev1:Cyclone IV GX""dev2:Stratix V GT"}`
String	DEFAULT_VALUE	The default value.
Boolean	DERIVED	When True, indicates that the parameter value is set by the component and cannot be set by the user. Derived parameters are not saved as part of an instance's parameter values. The default value is False.
String	DESCRIPTION	A short user-visible description of the parameter, suitable for a tooltip description in the parameter editor.
String[]	DISPLAY_HINT	Provides a hint about how to display a property. • boolean--For integer parameters whose value are 0 or 1. The parameter displays as an option that you can turn on or off. • radio—displays a parameter with a list of values as radio buttons. • hexadecimal—for integer parameters, displays and interprets the value as a hexadecimal number, for example: 0x00000010 instead of 16. • fixed_size—for string_list and integer_list parameters, the fixed_size DISPLAY_HINT eliminates the **Add** and **Remove** buttons from tables. • file--displays a file selection button that allows the user to enter a path to a file for the parameter.
String	DISPLAY_NAME	The GUI label that appears to the left of this parameter.
String	DISPLAY_UNITS	The GUI label that appears to the right of the parameter.

Type	Name	Description
Boolean	ENABLED	When False, the parameter is disabled. The parameter displays in the parameter editor but is grayed out, indicating that you cannot edit this parameter.
String	GROUP	Controls the layout of parameters in the GUI.
Boolean	HDL_PARAMETER	When True, Platform Designer passes the parameter to the HDL component description. The default value is False.
String	LONG_DESCRIPTION	A user-visible description of the parameter. Similar to DESCRIPTION, but allows a more detailed explanation.
String	NEW_INSTANCE_VALUE	Changes the default value of a parameter without affecting older components that do not explicitly set a parameter value, and use the DEFAULT_VALUE property. Oder instances continue to use DEFAULT_VALUE for the parameter and new instances use the value assigned by NEW_INSTANCE_VALUE.
String[]	SYSTEM_INFO	Allows you to assign information about the instantiating system to a parameter that you define. SYSTEM_INFO requires an argument specifying the type of information. For example: `SYSTEM_INFO <info-type>`
String	SYSTEM_INFO_ARG	Defines an argument to pass to SYSTEM_INFO. For example, the name of a reset interface.
(various)	SYSTEM_INFO_TYPE	Specifies the types of system information that you can query. Refer to *System Info Type Properties*.
(various)	TYPE	Specifies the type of the parameter. Refer to *Parameter Type Properties*.
(various)	UNITS	Sets the units of the parameter. Refer to *Units Properties*.
Boolean	VISIBLE	Indicates whether or not to display the parameter in the parameter editor.
String	WIDTH	Indicates the width of the logic vector for the STD_LOGIC_VECTOR parameter.

Related Information

- System Info Type Properties on page 652
- Parameter Type Properties on page 649
- Units Properties on page 654

7.15.10. Parameter Status Properties

Type	Name	Description
Boolean	ACTIVE	Indicates that this parameter is an active parameter.
Boolean	DEPRECATED	Indicates that this parameter exists only for backwards compatibility, and may not have any effect.
Boolean	EXPERIMENTAL	Indicates that this parameter is experimental and not exposed in the design flow.

7.15.11. Parameter Type Properties

Name	Description
BOOLEAN	A boolean parameter set to `true` or `false`.
FLOAT	A signed 32-bit floating point parameter. (Not supported for HDL parameters.)
INTEGER	A signed 32-bit integer parameter.
INTEGER_LIST	A parameter that contains a list of 32-bit integers. (Not supported for HDL parameters.)
LONG	A signed 64-bit integer parameter. (Not supported for HDL parameters.)
NATURAL	A 32-bit number that contains values `0` to `2147483647` (`0x7fffffff`).
POSITIVE	A 32-bit number that contains values `1` to `2147483647` (`0x7fffffff`).
STD_LOGIC	A single bit parameter set to `0` or `1`.
STD_LOGIC_VECTOR	An arbitrary-width number. The parameter property `WIDTH` determines the size of the logic vector.
STRING	A string parameter.
STRING_LIST	A parameter that contains a list of strings. (Not supported for HDL parameters.)

7.15.12. Port Properties

Type	Name	Description
(various)	DIRECTION	The direction of the signal. Refer to *Direction Properties*.
String	ROLE	The type of the signal. Each interface type defines a set of interface types for its ports.
Integer	WIDTH	The width of the signal in bits.

Related Information

Direction Properties on page 640

7.15.13. Project Properties

Type	Name	Description
String	DEVICE	The device part number in the Quartus Prime project that contains the Platform Designer system.
String	DEVICE_FAMILY	The device family name in the Quartus Prime project that contains the Platform Designer system.

7.15.14. System Info Type Properties

Type	Name	Description
String	ADDRESS_MAP	An XML-formatted string that describes the address map for the interface specified in the SYSTEM_INFO parameter property.
Integer	ADDRESS_WIDTH	The number of address bits that Platform Designer requires to address memory-mapped agents connected to the specified memory-mapped host in this instance.
String	AVALON_SPEC	The version of the Platform Designer interconnect. Refer to *Avalon Interface Specifications*.
Integer	CLOCK_DOMAIN	An integer that represents the clock domain for the interface specified in the SYSTEM_INFO parameter property. If this instance has interfaces on multiple clock domains, you can use this property to determine which interfaces are on each clock domain. The absolute value of the integer is arbitrary.
Long, Integer	CLOCK_RATE	The rate of the clock connected to the clock input specified in the SYSTEM_INFO parameter property. If zero, the clock rate is currently unknown.
String	CLOCK_RESET_INFO	The name of this instance's primary clock or reset sink interface. You use this property to determine the reset sink for global reset when you use Platform Designer interconnect that conforms to *Avalon Interface Specifications*.
String	CUSTOM_INSTRUCTION_SLAVES	Provides agent information, including the name, base address, address span, and clock cycle type.
String	DESIGN_ENVIRONMENT	A string that identifies the current design environment. Refer to *Design Environment Type Properties*.
String	DEVICE	The device part number of the selected device.
String	DEVICE_FAMILY	The family name of the selected device.
String	DEVICE_FEATURES	A list of key/value pairs delimited by spaces that indicate whether a device feature is available in the selected device family. The format of the list is suitable for passing to the `array` command. The keys are device features. The values are 1 if the feature is present, and 0 if the feature is absent.
String	DEVICE_SPEEDGRADE	The speed grade of the selected device.
Integer	GENERATION_ID	An integer that stores a hash of the generation time that Platform Designer uses as a unique ID for a generation run.
BigInteger, Long	INTERRUPTS_USED	A mask indicating which bits of an interrupt receiver are connected to interrupt senders. The interrupt receiver is specified in the system info argument.
Integer	MAX_SLAVE_DATA_WIDTH	The data width of the widest agent connected to the specified memory-mapped host.
String, Boolean, Integer	QUARTUS_INI	The value of the `quartus.ini` setting specified in the system info argument.
Integer	RESET_DOMAIN	An integer representing the reset domain for the interface specified in the SYSTEM_INFO parameter property If this instance has interfaces on multiple reset

Type	Name	Description
		domains, you can use this property to determine which interfaces are on each reset domain. The absolute value of the integer is arbitrary.
String	TRISTATECONDUIT_INFO	An XML description of the tri-state conduit hosts connected to a tri-state conduit agent. The agent is specified as the SYSTEM_INFO parameter property. The value contains information about the agent, connected host instance and interface names, and signal names, directions, and widths.
String	TRISTATECONDUIT_MASTERS	The names of the instance's interfaces that are tri-state conduit agents.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

Related Information

- Design Environment Type Properties on page 639
- Avalon Interface Specifications
- Platform Designer Interconnect on page 251

7.15.15. Units Properties

Name	Description
ADDRESS	A memory-mapped address.
BITS	Memory size in bits.
BITSPERSECOND	Rate in bits per second.
BYTES	Memory size in bytes.
CYCLES	A latency or count in clock cycles.
GIGABITSPERSECOND	Rate in gigabits per second.
GIGABYTES	Memory size in gigabytes.
GIGAHERTZ	Frequency in GHz.
HERTZ	Frequency in Hz.
KILOBITSPERSECOND	Rate in kilobits per second.
KILOBYTES	Memory size in kilobytes.
KILOHERTZ	Frequency in kHz.
MEGABITSPERSECOND	Rate, in megabits per second.
MEGABYTES	Memory size in megabytes.
MEGAHERTZ	Frequency in MHz.
MICROSECONDS	Time in microseconds.
MILLISECONDS	Time in milliseconds.
NANOSECONDS	Time in nanoseconds.
NONE	Unspecified units.
PERCENT	A percentage.
PICOSECONDS	Time in picoseconds.
SECONDS	Time in seconds.

7.15.16. Validation Properties

Type	Name	Description
Boolean	AUTOMATIC_VALIDATION	When `true`, Platform Designer runs system validation and elaboration after each scripting command. When `false`, Platform Designer runs system validation with validation scripting commands. Some queries affected by system elaboration may be incorrect if automatic validation is disabled. You can disable validation to make a system script run faster.

7.15.17. Interface Direction

Type	Name	Description
String	INPUT	Indicates that the interface is an agent (input, transmitter, sink, or end).
String	OUTPUT	Indicates that the interface is a host (output, receiver, source, or start).

7.15.18. File Set Kind

Name	Description
EXAMPLE_DESIGN	This file-set contains example design files.
QUARTUS_SYNTH	This file-set contains files that Platform Designer uses for Quartus Prime Synthesis
SIM_VERILOG	This file-set contains files that Platform Designer uses for Verilog HDL Simulation.
SIM_VHDL	This file-set contains files that Platform Designer uses for VHDL Simulation.

7.15.19. Access Type

Name	Type	Description
String	READ_ONLY	Indicates that the parameter can be only read-only.
String	WRITABLE	Indicates that the parameter has read/write properties.

7.15.20. Instantiation HDL File Properties

Name	Type	Description
Boolean	CONTAINS_INLINE_CONFIGURATION	Returns *True* if the HDL file contains inline configuration.
Boolean	IS_CONFIGURATION_PACKAGE	Returns *True* if the HDL file is a configuration package.
Boolean	IS_TOP_LEVEL	Returns *True* if the HDL file is the top-level HDL file.
String	OUTPUT_PATH	Specifies the output path of the HDL file.
String	TYPE	Specifies the HDL file type of the HDL file.

7.15.21. Instantiation Interface Duplicate Type

Type	Name	Description
String	CLONE	Creates a copy of an interface and all the interface ports.
String	MIRROR	Creates a copy of an interface with all the port roles and directions reversed.

7.15.22. Instantiation Interface Properties

Name	Type	Description
String	`DIRECTION`	The direction of the interface.
String	`TYPE`	The type of the interface.

7.15.23. Instantiation Properties

Name	Type	Description
String	HDL_COMPILATION_LIBRARY	Indicates the HDL compilation library name of the generic component.
String	HDL_ENTITY_NAME	Indicates the HDL entity name of the Generic Component.
String	IP_FILE	Indicates the .ip file path that implements the generic component.

7.15.24. Port Properties

Name	Type	Description
String	`DIRECTION`	Specifies the direction of the signal
String	`NAME`	Renames a top-level port. Only use with `set_interface_port_property`
String	`ROLE`	Specifies the type of the signal. Each interface type defines a set of interface types for its ports.
String	`VHDL_TYPE`	Specifies the VHDL type of the signal. Can be either *STANDARD_LOGIC*, or *STANDARD_LOGIC_VECTOR*.
Integer	`WIDTH`	Specifies the width of the signal in bits.

Related Information

Direction Properties on page 640

7.15.25. VHDL Type

Name	Description
STD_LOGIC	Represents the value of a digital signal in a wire.
STD_LOGIC_VECTOR	Represents an array of digital signals and variables.

7.16. Platform Designer Command-Line Utilities Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout. • Added new *Apply Presets to a New Board* topic. • Added missing file option to *Parameter Properties* topic.
2023.12.20	23.4	• Updated *Run the Platform Designer Editor with qsys-edit* topic for new metrics logging switches. • Updated *qsys-generate Command-Line Options* topic for new metrics logging switches. • Updated *Generate a Platform Designer System with qsys-script* topic for new metrics logging switches.
2023.04.03	23.1	• Updated *Board-Aware Flow Scripting Support* topic for new get_quartus_instance_path_for_entity Tcl API. • The product family name is updated to "Intel Agilex 7" to reflect the different family members.
2022.12.12	22.4	• Added new *Board-Aware Flow Scripting Support* topic.
2022.06.20	22.2	• Revised `add_instance` topic to indicate that the `version` argument is required.
2021.03.29	21.1	• Revised *Archive and Extract Platform Designer Systems with qsys-archive* title and link to GUI topic. • Converted to "host" and "agent" inclusive terminology for Avalon memory mapped interface descriptions and related GUI elements throughout.
2020.06.22	20.1	• Added set_design_id command.
2019.11.11	19.1	• Added statement to `qsys-generate` topic indicating that `graybox` option is only for individual Intel FPGA IP cores, and not for complete Platform Designer systems. • Added statement to `qsys-generate` topic indicating that `upgrade-ip-cores` has no impact on subsystems.
2019.04.01	19.1	• Added new Domains command-line reference and deprecated `get_interconnect_requirement`, `get_interconnect_requirements`, and `set_interconnect_requirement assignments`.
2018.12.15	18.1	First release as separate chapter.
2016.10.31	16.1	• Added command-line options for qsys-archive. • Added command-line options for quartus_ipgenerate. • Updated the Qsys Pro scripting commands.

continued...

Document Version	Quartus Prime Version	Changes
2016.05.03	16.0	• Qsys Command-Line Utilities updated with latest supported command-line options.
June 2012	12.0	• Added command-line utilities, and scripts.
December 2010	10.1	Initial release of content.

8. Component Interface Tcl Reference

Tcl commands allow you to perform a wide range of functions in Platform Designer. Command descriptions contain the Platform Designer phases where you can use the command, for example, main program, elaboration, composition, or fileset callback. This reference denotes optional command arguments in brackets [ ].

Note: Intel now refers to Qsys Pro as Platform Designer.

Platform Designer supports Avalon, AMBA 3 AXI (version 1.0), AMBA 4 AXI (version 2.0), AMBA 4 AXI-Lite (version 2.0), AMBA 4 AXI-Stream (version 1.0), and AMBA 3 APB (version 1.0) interface specifications.

For more information about procedures for creating IP component `_hw.tcl` files in the Platform Designer Component Editor, and supported interface standards, refer to *Creating Platform Designer Components* and *Platform Designer Interconnect*.

If you are developing an IP component to work with the Nios II processor, refer to *Publishing Component Information to Embedded Software* in section 3 of the *Nios II Software Developer's Handbook*, which describes how to publish hardware IP component information for embedded software tools, such as a C compiler and a Board Support Package (BSP) generator.

Related Information

- Avalon Interface Specifications
- Creating Platform Designer Components on page 153
- Platform Designer Interconnect on page 251

8.1. Platform Designer _hw.tcl Command Reference

8.1.1. Interfaces and Ports

Related Information

8.1.1.1. add_interface

Description

Adds an interface to your module. An interface represents a collection of related signals that are managed together in the parent system. These signals are implemented in the IP component's HDL, or exported from an interface from a child instance. As the IP component author, you choose the name of the interface.

Availability

Discovery, Main Program, Elaboration, Composition

Usage

`add_interface` <*name*> <*type*> <*direction*> [<*associated_clock*>]

Returns

No returns value.

Arguments

name A name you choose to identify an interface.

type The type of interface.

direction The interface direction.

associated_clock (optional) (deprecated) For interfaces requiring associated clocks, use: `set_interface_property` <*interface*> `associatedClock` <*clockInterface*> For interfaces requiring associated resets, use: `set_interface_property` <*interface*> `associatedReset` <*resetInterface*>

Example

```
add_interface mm_agent avalon agent

add_interface my_export conduit end
set_interface_property my_export EXPORT_OF uart_0.external_connection
```

Notes

By default, interfaces are enabled. You can set the interface property ENABLED to `false` to disable an interface. If an interface is disabled, it is hidden and its ports are automatically terminated to their default values. Active high signals are terminated to 0. Active low signals are terminated to 1.

If the IP component is composed of child instances, the top-level interface is associated with a child instance's interface with `set_interface_property` *`interface`* `EXPORT_OF` *`child_instance.interface`*.

The following direction rules apply to Platform Designer-supported interfaces.

Interface Type	Direction
avalon	host, agent
axi	manager, subordinate
tristate_conduit	host, agent
avalon_streaming	source, sink
interrupt	sender, receiver
conduit	end
clock	source, sink
reset	source, sink
nios_custom_instruction	slave

Related Information

- add_interface_port on page 670
- get_interface_assignments on page 674
- get_interface_properties on page 676
- get_interfaces on page 672

8.1.1.2. add_interface_port

Description

Adds a port to an interface on your module. The name must match the name of a signal on the top-level module in the HDL of your IP component. The port width and direction must be set before the end of the elaboration phase. You can set the port width as follows:

- In the Main program, you can set the port width to a fixed value or a width expression.
- If the port width is set to a fixed value in the Main program, you can update the width in the elaboration callback.

Availability

Main Program, Elaboration

Usage

`add_interface_port` *<interface>* *<port>* [*<signal_type>* *<direction>* *<width_expression>*]

Returns

Arguments

interface The name of the interface to which this port belongs.

port The name of the port. This name must match a signal in your top-level HDL for this IP component.

signal_type (optional) The type of signal for this port, which must be unique. Refer to the *Avalon Interface Specifications* for the signal types available for each interface type.

direction (optional) The direction of the signal. Refer to *Direction Properties*.

width_expression (optional) The width of the port, in bits. The width may be a fixed value, or a simple arithmetic expression of parameter values.

Example

```
fixed width:
add_interface_port mm_agent s0_rdata readdata output 32

width expression:
add_parameter DATA_WIDTH INTEGER 32
add_interface_port s0 rdata readdata output "DATA_WIDTH/2"
```

Related Information

- add_interface on page 668
- get_port_properties on page 678

- get_port_property on page 679
- get_port_property on page 679
- Direction Properties on page 774
- Avalon Interface Specifications

8.1.1.3. get_interfaces

Description

Returns a list of top-level interfaces.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_interfaces
```

Returns

A list of the top-level interfaces exported from the system.

Arguments

No arguments.

Example

```
get_interfaces
```

Related Information

add_interface on page 668

8.1.1.4. get_interface_assignment

Description

Returns the value of the specified assignment for the specified interface

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_interface_assignment` <*interface*> <*assignment*>

Returns

The value of the assignment.

Arguments

interface The name of a top-level interface.

assignment The name of an assignment.

Example

```
get_interface_assignment s1 embeddedsw.configuration.isFlash
```

Related Information

- add_interface on page 668
- get_interface_assignments on page 674
- get_interfaces on page 672

8.1.1.5. get_interface_assignments

Description

Returns the value of all interface assignments for the specified interface.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_interface_assignments` <*interface*>

Returns

A list of assignment keys.

Arguments

interface The name of the top-level interface whose assignment is being retrieved.

Example

```
get_interface_assignments s1
```

Related Information

- add_interface on page 668
- get_interface_assignment on page 673
- get_interfaces on page 672

8.1.1.6. get_interface_ports

Description

Returns the names of all of the ports that have been added to a given interface. If the interface name is omitted, all ports for all interfaces are returned.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_interface_ports` [<*interface*>]

Returns

A list of port names.

Arguments

interface (optional) The name of a top-level interface.

Example

```
get_interface_ports mm_agent
```

Related Information

- add_interface_port on page 670
- get_port_property on page 679
- set_port_property on page 683

8.1.1.7. get_interface_properties

Description

Returns the names of all the interface properties for the specified interface as a space separated list

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_interface_properties` <*interface*>

Returns

A list of properties for the interface.

Arguments

interface The name of an interface.

Example

```
get_interface_properties interface
```

Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

Related Information

- get_interface_property on page 677
- set_interface_property on page 682
- Avalon Interface Specifications

8.1.1.8. get_interface_property

Description

Returns the value of a single interface property from the specified interface.

Availability

Discovery, Main Program, Elaboration, Composition, Fileset Generation

Usage

`get_interface_property` <*interface*> <*property*>

Returns

Arguments

interface The name of an interface.

property The name of the property whose value you want to retrieve. Refer to *Interface Properties*.

Example

```
get_interface_property mm_agent linewrapBursts
```

Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

Related Information

- get_interface_properties on page 676
- set_interface_property on page 682
- Avalon Interface Specifications

8.1.1.9. get_port_properties

Description

Returns a list of port properties.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_port_properties
```

Returns

A list of port properties. Refer to *Port Properties*.

Arguments

No arguments.

Example

```
get_port_properties
```

Related Information

- add_interface_port on page 670
- get_port_property on page 679
- set_port_property on page 683
- Port Properties on page 772

8.1.1.10. get_port_property

Description

Returns the value of a property for the specified port.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_port_property` <*port*> <*property*>

Returns

The value of the property.

Arguments

port The name of the port.

property The name of a port property. Refer to *Port Properties*.

Example

```
get_port_property rdata WIDTH_VALUE
```

Related Information

- add_interface_port on page 670
- get_port_properties on page 678
- set_port_property on page 683
- Port Properties on page 772

8.1.1.11. set_interface_assignment

Description

Sets the value of the specified assignment for the specified interface.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`set_interface_assignment` <*interface*> <*assignment*> [<*value*>]

Returns

No return value.

Arguments

interface The name of the top-level interface whose assignment is being set.

assignment The assignment whose value is being set.

value (optional) The new assignment value.

Example

```
set_interface_assignment s1 embeddedsw.configuration.isFlash 1
```

Notes

Assignments for Nios II Software Build Tools

Interface assignments provide extra data for the Nios II Software Build Tools working with the generated system.

Assignments for Platform Designer Tools

There are several assignments that guide behavior in the Platform Designer tools.

qsys.ui.export_name:	If present, this interface should always be exported when an instance is added to a Platform Designer system. The value is the requested name of the exported interface in the parent system.
qsys.ui.connect:	If present, this interface should be auto-connected when an instance is added to a Platform Designer system. The value is a comma-separated list of other interfaces on the same instance that should be connected with this interface.

ui.blockdiagram.direction:	If present, the direction of this interface in the block diagram is set by the user. The value is either "output" or "input".

Related Information

- add_interface on page 668
- get_interface_assignment on page 673
- get_interface_assignments on page 674

8.1.1.12. set_interface_property

Description

Sets the value of a property on an exported top-level interface. You can use this command to set the EXPORT_OF property to specify which interface of a child instance is exported via this top-level interface.

Availability

Main Program, Elaboration, Composition

Usage

`set_interface_property` <*interface*> <*property*> <*value*>

Returns

No return value.

Arguments

interface The name of an exported top-level interface.

property The name of the property Refer to *Interface Properties*.

value The new property value.

Example

```
set_interface_property clk_out EXPORT_OF clk.clk_out
set_interface_property mm_agent linewrapBursts false
```

Notes

The properties for each interface type are different. Refer to the *Avalon Interface Specifications* for more information about interface properties.

Related Information

- get_interface_properties on page 676
- get_interface_property on page 677
- Avalon Interface Specifications

8.1.1.13. set_port_property

Description

Sets a port property.

Availability

Elaboration

Usage

`set_port_property` <*port*> <*property*> [<*value*>]

Returns

The new value.

Arguments

port The name of the port.

property One of the supported properties. Refer to *Port Properties*.

value (optional) The value to set.

Example

```
set_port_property rdata WIDTH 32
```

Related Information

- add_interface_port on page 670
- get_port_properties on page 678
- set_port_property on page 683

8.1.1.14. set_interface_upgrade_map

Description

Maps the interface name of an older version of an IP core to the interface name of the current IP core. The interface type must be the same between the older and newer versions of the IP cores. This allows system connections and properties to maintain proper functionality. By default, if the older and newer versions of IP core have the same name and type, then Platform Designer maintains all properties and connections automatically.

Availability

Parameter Upgrade

Usage

```
set_interface_upgrade_map { <old_interface_name> <new_interface_name>
<old_interface_name_2> <new_interface_name_2> … }
```

Returns

No return value.

Arguments

{ <old_interface_name> <new_interface_name>}	List of mappings between names of older and newer interfaces.

Example

```
set_interface_upgrade_map { avalon_host_interface new_avalon_host_interface }
```

8.1.2. Parameters

8.1.2.1. add_parameter

Description

Adds a parameter to your IP component.

Availability

Main Program

Usage

`add_parameter <name> <type> [<default_value> <description>]`

Returns

Arguments

name The name of the parameter.

type The data type of the parameter Refer to *Parameter Type Properties*.

default_value (optional) The initial value of the parameter in a new instance of the IP component.

description (optional) Explains the use of the parameter.

Example

```
add_parameter seed INTEGER 17 "The seed to use for data generation."
```

Notes

Most parameter types have a single GUI element for editing the parameter value. `string_list` and `integer_list` parameters are different, because they are edited as tables. A multi-column table can be created by grouping multiple into a single table. To edit multiple list parameters in a single table, the display items for the parameters must be added to a group with a TABLE hint:

```
add_parameter coefficients INTEGER_LIST add_parameter positions
INTEGER_LIST add_display_item "" "Table Group" GROUP TABLE
add_display_item "Table Group" coefficients PARAMETER
add_display_item "Table Group" positions PARAMETER
```

Related Information

- get_parameter_properties on page 688
- get_parameter_property on page 689
- get_parameter_value on page 690
- set_parameter_property on page 693
- set_parameter_value on page 694
- Parameter Type Properties on page 770

8.1.2.2. get_parameters

Description

Returns the names of all the parameters in the IP component.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_parameters
```

Returns

A list of parameter names

Arguments

No arguments.

Example

```
get_parameters
```

Related Information

- add_parameter on page 686
- get_parameter_property on page 689
- get_parameter_value on page 690
- get_parameters on page 687
- set_parameter_property on page 693

8.1.2.3. get_parameter_properties

Description

Returns a list of all the parameter properties as a list of strings. The `get_parameter_property` and `set_parameter_property` commands are used to get and set the values of these properties, respectively.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_parameter_properties
```

Returns

A list of parameter property names. Refer to *Parameter Properties*.

Arguments

No arguments.

Example

```
set property_summary [ get_parameter_properties ]
```

Related Information

- add_parameter on page 686
- get_parameter_property on page 689
- get_parameter_value on page 690
- get_parameters on page 687
- set_parameter_property on page 693
- Parameter Properties on page 768

8.1.2.4. get_parameter_property

Description

Returns the value of a property of a parameter.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_parameter_property` <*parameter*> <*property*>

Returns

The value of the property.

Arguments

parameter The name of the parameter whose property value is being retrieved.

property The name of the property. Refer to *Parameter Properties*.

Example

```
set enabled [ get_parameter_property parameter1 ENABLED ]
```

Related Information

- add_parameter on page 686
- get_parameter_properties on page 688
- get_parameter_value on page 690
- get_parameters on page 687
- set_parameter_property on page 693
- set_parameter_value on page 694
- Parameter Properties on page 768

8.1.2.5. get_parameter_value

Description

Returns the current value of a parameter defined previously with the `add_parameter` command.

Availability

Discovery, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_parameter_value` <*parameter*>

Returns

The value of the parameter.

Arguments

parameter The name of the parameter whose value is being retrieved.

Example

```
set width [ get_parameter_value fifo_width ]
```

Notes

If `AFFECTS_ELABORATION` is `false` for a given parameter, `get_parameter_value` is not available for that parameter from the elaboration callback. If `AFFECTS_GENERATION` is `false` then it is not available from the generation callback.

Related Information

- add_parameter on page 686
- get_parameter_property on page 689
- get_parameters on page 687
- set_parameter_property on page 693
- set_parameter_value on page 694

8.1.2.6. get_string

Description

Returns the value of an externalized string previously loaded by the `load_strings` command.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_string` <*identifier*>

Returns

The externalized string.

Arguments

identifier The string identifier.

Example

```
hw.tcl:
load_strings test.properties
set_module_property NAME test
set_module_property VERSION [get_string VERSION]
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
add_parameter firepower INTEGER 0 ""
set_parameter_property firepower DISPLAY_NAME [get_string PARAM_DISPLAY_NAME]
set_parameter_property firepower TYPE INTEGER
set_parameter_property firepower DESCRIPTION [get_string PARAM_DESCRIPTION]

test.properties:
DISPLAY_NAME = Trogdor!
VERSION = 1.0
PARAM_DISPLAY_NAME = Firepower
PARAM_DESCRIPTION = The amount of force to use when breathing fire.
```

Notes

Use uppercase words separated with underscores to name string identifiers. If you are externalizing module properties, use the module property name for the string identifier:

```
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
```

If you are externalizing a parameter property, qualify the parameter property with the parameter name, with uppercase format, if needed:

```
set_parameter_property my_param DISPLAY_NAME [get_string MY_PARAM_DISPLAY_NAME]
```

If you use a string to describe a string format, end the identifier with _FORMAT.

```
set formatted_string [ format  [ get_string TWO_ARGUMENT_MESSAGE_FORMAT ] "arg1"
"arg2" ]
```

Related Information

load_strings on page 692

8.1.2.7. load_strings

Description

Loads strings from an external `.properties` file.

Availability

Discovery, Main Program

Usage

`load_strings` <*path*>

Returns

No return value.

Arguments

path The path to the properties file.

Example

```
hw.tcl:
load_strings test.properties
set_module_property NAME test
set_module_property VERSION [get_string VERSION]
set_module_property DISPLAY_NAME [get_string DISPLAY_NAME]
add_parameter firepower INTEGER 0 ""
set_parameter_property firepower DISPLAY_NAME [get_string PARAM_DISPLAY_NAME]
set_parameter_property firepower TYPE INTEGER
set_parameter_property firepower DESCRIPTION [get_string PARAM_DESCRIPTION]

test.properties:
DISPLAY_NAME = Trogdor!
VERSION = 1.0
PARAM_DISPLAY_NAME = Firepower
PARAM_DESCRIPTION = The amount of force to use when breathing fire.
```

Notes

Refer to the *Java Properties File* for properties file format. A `.properties` file is a text file with *KEY*=*value* pairs. For externalized strings, the *KEY* is a string identifier and the *value* is the externalized string.
For example:

```
TROGDOR = A dragon with a big beefy arm
```

Related Information

- get_string on page 691
- Java Properties File

8.1.2.8. set_parameter_property

Description

Sets a single parameter property.

Availability

Main Program, Edit, Elaboration, Validation, Composition

Usage

`set_parameter_property` <*parameter*> <*property*> <*value*>

Returns

Arguments

parameter The name of the parameter that is being set.

property The name of the property. Refer to *Parameter Properties*.

value The new value for the property.

Example

```
set_parameter_property BAUD_RATE ALLOWED_RANGES {9600 19200 38400}
```

Related Information

- add_parameter on page 686
- get_parameter_properties on page 688
- set_parameter_property on page 693
- Parameter Properties on page 768

8.1.2.9. set_parameter_value

Description

Sets a parameter value. The value of a derived parameter can be updated by the IP component in the elaboration callback or the edit callback. Any changes to the value of a derived parameter in the edit callback is not preserved.

Availability

Edit, Elaboration, Validation, Composition, Parameter Upgrade

Usage

`set_parameter_value` <*parameter*> <*value*>

Returns

No return value.

Arguments

parameter The name of the parameter that is being set.

value Specifies the new parameter value.

Example

```
set_parameter_value half_clock_rate [ expr { [ get_parameter_value
clock_rate ] / 2 } ]
```

8.1.3. Interconnect Parameters

8.1.3.1. set_domain_assignment

Description

Sets the assignment value to all connections on the given domain.

Availability

Composition

Usage

```
set_domain_assignment <element> <assignment> <value>
```

Arguments

element Connection or interface in the domain to which you want to set the assignment. If the element name is `$system`, the assignment applies to all the domains in the system.

assignment The name of the assignment.

value The value of the assignment.

8.1.3.2. get_domain_assignment

Description

Returns the value for the specified assignment in the given domain.

Availability

Composition

Usage

```
get_domain_assignment <element> <assignment>
```

Arguments

element Connection or interface in the domain for which you want to get the assignment value.

assignment The name of the assignment.

8.1.3.3. get_domain_assignments

Description

Returns all domain assignments for the given domain as a list of strings. Each "group" of three elements in the list contains the element name, assignment name, and value, in that order. Element name in the output is the input element name. If the input element is `$system`, then the output element name is the connection point in the domain. The Returns section shows a typical list.

Returns

```
[element0 name0 value0 element1 name1 value1 ... ]
```

In TCL, you'd loop over the list by writing a foreach loop:

```
foreach {element name value } \
    $requirement_list { puts " $element $name $value" }
```

Availability

Composition

Usage

```
get_domain_assignments <element>
```

Arguments

element Connection or interface in the domains for which you want to get the assignments value. If you specify $system as the element, the command returns values of all the domains in the system.

8.1.3.4. set_postadaptation_assignment

Description

Adds an post adaptation interconnect assignment.

Availability

Composition

Usage

```
set_postadaptation_assignment <element> <assignment> <value>
```

Arguments

element Connection or interface in the domain to which you want to set the assignment.

assignment The name of the assignment.

value The value of the assignment.

8.1.3.5. get_postadaptation_assignment

Description

Returns the value of the named post adaptation interconnect assignment on the specified element.

Availability

Composition

Usage

```
get_postadaptation_assignment <element> <assignment>
```

Arguments

element Connection or interface in the domain for which you want to get the assignment value.

assignment The name of the assignment.

8.1.3.6. get_postadaptation_assignments

Description

Returns all post adaptation interconnect assignments for the given domain as a list of strings. Each "group" of three elements in the list contains the element name, assignment name and value in that order. The Returns section shows a typical list.

Returns

```
[element0 name0 value0 element1 name1 value1 ... ]
```

In Tcl, you loop over the list by writing a foreach loop:

```
foreach {element name value } $requirement_list \
     { puts " $element $name $value" }
```

Availability

Composition

Usage

```
get_postadaptation_assignments <element>
```

Arguments

element Connection or interface in the domain to which you want to set the assignment.

8.1.4. Display Items

8.1.4.1. add_display_item

Description

Specifies the following aspects of the IP component display:

- Creates logical groups for an IP component's parameters. For example, to create separate groups for the IP component's timing, size, and simulation parameters. An IP component displays the groups and parameters in the order that you specify the display items in the `_hw.tcl` file.
- Groups a list of parameters to create multi-column tables.
- Specifies an image to provide representation of a parameter or parameter group.
- Creates a button by adding a display item of type `action`. The display item includes the name of the callback to run.

Availability

Main Program

Usage

`add_display_item` <*parent_group*> <*id*> <*type*> [<*args*>]

Returns

Arguments

parent_group Specifies the group to which a display item belongs

id The identifier for the display item. If the item being added is a parameter, this is the parameter name. If the item is a group, this is the group name.

type The type of the display item. Refer to *Display Item Kind Properties*.

args (optional) Provides extra information required for display items.

Example

```
add_display_item "Timing" read_latency PARAMETER
add_display_item "Sounds" speaker_image_id ICON speaker.jpg
```

Notes

The following examples illustrate further illustrate the use of arguments:

- `add_display_item groupName id icon` *`path-to-image-file`*
- `add_display_item groupName parameterName parameter`
- `add_display_item groupName id text "your-text"`

  The your-text argument is a block of text that is displayed in the GUI. Some simple HTML formatting is allowed, such as <b> and <i>, if the text starts with `<html>`.
- `add_display_item parentGroupName childGroupName group [tab]`

  The tab is an optional parameter. If present, the group appears in separate tab in the GUI for the instance.
- `add_display_item parentGroupName actionName action buttonClickCallbackProc`

Related Information

- get_display_item_properties on page 702
- get_display_item_property on page 703
- get_display_items on page 701
- set_display_item_property on page 704
- Display Item Kind Properties on page 776

8.1.4.2. get_display_items

Description

Returns a list of all items to be displayed as part of the parameterization GUI.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_display_items
```

Returns

List of display item IDs.

Arguments

No arguments.

Example

```
get_display_items
```

Related Information

- add_display_item on page 699
- get_display_item_properties on page 702
- get_display_item_property on page 703
- set_display_item_property on page 704

8.1.4.3. get_display_item_properties

Description

Returns a list of names of the properties of display items that are part of the parameterization GUI.

Availability

Main Program

Usage

```
get_display_item_properties
```

Returns

A list of display item property names. Refer to *Display Item Properties*.

Arguments

No arguments.

Example

```
get_display_item_properties
```

Related Information

- add_display_item on page 699
- get_display_item_property on page 703
- set_display_item_property on page 704
- Display Item Properties on page 775

8.1.4.4. get_display_item_property

Description

Returns the value of a specific property of a display item that is part of the parameterization GUI.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_display_item_property` <*display_item*> <*property*>

Returns

The value of a display item property.

Arguments

display_item The id of the display item.

property The name of the property. Refer to *Display Item Properties*.

Example

```
set my_label [get_display_item_property my_action DISPLAY_NAME]
```

Related Information

- add_display_item on page 699
- get_display_item_properties on page 702
- get_display_items on page 701
- set_display_item_property on page 704
- Display Item Properties on page 775

8.1.4.5. set_display_item_property

Description

Sets the value of specific property of a display item that is part of the parameterization GUI.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition

Usage

`set_display_item_property` <*display_item*> <*property*> <*value*>

Returns

No return value.

Arguments

display_item The name of the display item whose property value is being set.

property The property that is being set. Refer to *Display Item Properties*.

value The value to set.

Example

```
set_display_item_property my_action DISPLAY_NAME "Click Me"
set_display_item_property my_action DESCRIPTION "clicking this button runs the
click_me_callback proc in the hw.tcl file"
```

Related Information

- add_display_item on page 699
- get_display_item_properties on page 702
- get_display_item_property on page 703
- Display Item Properties on page 775

8.1.5. Module Definition

8.1.5.1. add_documentation_link

Description

Allows you to link to documentation for your IP component.

Availability

Discovery, Main Program

Usage

`add_documentation_link` <*title*> <*path*>

Returns

No return value.

Arguments

title The title of the document for use on menus and buttons.

path A path to the IP component documentation, using a syntax that provides the entire URL, not a relative path. For example: `http://www.mydomain.com/my_memory_controller.html` or `file:///datasheet.txt`

Example

```
add_documentation_link "Avalon Verification IP Suite User Guide" http://
www.altera.com/literature/ug/ug_avalon_verification_ip.pdf
```

8.1.5.2. get_module_assignment

Description

This command returns the value of an assignment. You can use the `get_module_assignment` and `set_module_assignment` and the `get_interface_assignment` and `set_interface_assignment` commands to provide information about the IP component to embedded software tools and applications.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_module_assignment` <*assignment*>

Returns

The value of the assignment

Arguments

assignment The name of the assignment whose value is being retrieved

Example

```
get_module_assignment embeddedsw.CMacro.colorSpace
```

Related Information

- get_module_assignments on page 708
- set_module_assignment on page 713

8.1.5.3. get_module_assignments

Description

Returns the names of the module assignments.

Availability

Main Program, Elaboration, Validation, Composition

Usage

```
get_module_assignments
```

Returns

A list of assignment names.

Arguments

No arguments.

Example

```
get_module_assignments
```

Related Information

- get_module_assignment on page 707
- set_module_assignment on page 713

8.1.5.4. get_module_ports

Description

Returns a list of the names of all the ports which are currently defined.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_module_ports
```

Returns

A list of port names.

Arguments

No arguments.

Example

```
get_module_ports
```

Related Information

- add_interface on page 668
- add_interface_port on page 670

8.1.5.5. get_module_properties

Description

Returns the names of all the module properties as a list of strings. You can use the `get_module_property` and `set_module_property` commands to get and set values of individual properties. The value returned by this command is always the same for a particular version of Platform Designer

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_module_properties
```

Returns

List of strings. Refer to *Module Properties*.

Arguments

No arguments.

Example

```
get_module_properties
```

Related Information

- get_module_property on page 711
- set_module_property on page 714
- Module Properties on page 778

8.1.5.6. get_module_property

Description

Returns the value of a single module property.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_module_property` <*property*>

Returns

Various.

Arguments

property The name of the property, Refer to *Module Properties*.

Example

```
set my_name [ get_module_property NAME ]
```

Related Information

- get_module_properties on page 710
- set_module_property on page 714
- Module Properties on page 778

8.1.5.7. send_message

Description

Sends a message to the user of the IP component. The message text is normally interpreted as HTML. You can use the <b> element to provide emphasis. If you do not want the message text to be interpreted as HTML, then pass a list as the message level, for example, `{ Info Text }`.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

`send_message` <*level*> <*message*>

Returns

No return value .

Arguments

level The following message levels are supported:

- **ERROR**--Provides an error message. The Platform Designer system cannot be generated with existing error messages.
- **WARNING**--Provides a warning message.
- **INFO**--Provides an informational message. The **INFO** level is not available in the Main Program.
- **PROGRESS**--Reports progress during generation.
- **DEBUG**--Provides a debug message when debug mode is enabled.

message The text of the message.

Example

```
send_message ERROR "The system is down!"
send_message { Info Text } "The system is up!"
```

8.1.5.8. set_module_assignment

Description

Sets the value of the specified assignment.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`set_module_assignment` <*assignment*> [<*value*>]

Returns

No return value.

Arguments

assignment The assignment whose value is being set

value (optional) The value of the assignment

Example

```
set_module_assignment embeddedsw.CMacro.colorSpace CMYK
```

Related Information

- get_module_assignment on page 707
- get_module_assignments on page 708

8.1.5.9. set_module_property

Description

Allows you to set the values for module properties.

Availability

Discovery, Main Program

Usage

`set_module_property` <*property*> <*value*>

Returns

No return value.

Arguments

property The name of the property. Refer to *Module Properties*.

value The new value of the property.

Example

```
set_module_property VERSION 10.0
```

Related Information

- get_module_properties on page 710
- get_module_property on page 711
- Module Properties on page 778

8.1.5.10. add_hdl_instance

Description

Adds an instance of a predefined module, referred to as a *child* or *child instance*. The HDL entity generated from this instance can be instantiated and connected within this IP component's HDL.

Availability

Main Program, Elaboration, Composition

Usage

`add_hdl_instance` <*entity_name*> <*ip_type*> [<*version*>]

Returns

The entity name of the added instance.

Arguments

entity_name Specifies a unique local name that you can use to manipulate the instance. This name is used in the generated HDL to identify the instance.

ip_type The type refers to a kind of instance available in the IP Catalog, for example `altera_avalon_uart`.

version (optional) The required version of the specified instance type. If no version is specified, the latest version is used.

Example

```
add_hdl_instance my_uart altera_avalon_uart
```

Related Information

- get_instance_parameter_value on page 733
- get_instance_parameters on page 731
- get_instances on page 723
- set_instance_parameter_value on page 736

8.1.5.11. package

Description

Allows you to specify a particular version of the Platform Designer software to avoid software compatibility issues, and to determine which version of the **_hw.tcl** API to use for the IP component. You must use the package command at the beginning of your **_hw.tcl** file.

Availability

Main Program

Usage

```
package require -exact qsys <version>
```

Returns

No return value

Arguments

version The version of Platform Designer that you require, such as 14.1.

Example

```
package require -exact qsys 14.1
```

8.1.6. Composition

8.1.6.1. add_instance

Description

Adds an instance of an IP component, referred to as a child or child instance to the subsystem. You can use this command to create IP components that are composed of other IP component instances. The HDL for this subsystem generates; There is no need to write custom HDL for the IP component.

Availability

Main Program, Composition

Usage

`add_instance` <*name*> <*type*> [<*version*>]

Returns

No return value.

Arguments

name Specifies a unique local name that you can use to manipulate the instance. This name is used in the generated HDL to identify the instance.

type The type refers to a type available in the IP Catalog, for example `altera_avalon_uart`.

version The required version of the specified instance type. This argument is required in Package version 19.1 and later. Before package version 19.1, when not specified the latest IP version is used.

Example

```
add_instance my_uart altera_avalon_uart
add_instance my_uart altera_avalon_uart 22.1
```

Related Information

- add_connection on page 719
- get_instance_interface_property on page 730
- get_instance_parameter_value on page 733
- get_instance_parameters on page 731
- get_instance_property on page 727
- get_instances on page 723
- set_instance_parameter_value on page 736

8.1.6.2. add_connection

Description

Connects the named interfaces on child instances together using an appropriate connection type. Both interface names consist of a child instance name, followed by the name of an interface provided by that module. For example, `mux0.out` is the interface named `out` on the instance named `mux0`. Be careful to connect the start to the end, and not the other way around.

Availability

Main Program, Composition

Usage

`add_connection` <*start*> [<*end*> <*kind*> <*name*>]

Returns

The name of the newly added connection in `start.point/end.point` format.

Arguments

start The start interface to be connected, in `<instance_name>.<interface_name>` format.

end (optional) The end interface to be connected, `<instance_name>.<interface_name>`.

kind (optional) The type of connection, such as `avalon` or `clock`.

name (optional) A custom name for the connection. If unspecified, the name will be `<start_instance>.<interface>.<end_instance><interface>`

Example

```
add_connection dma.read_host sdram.s1 avalon
```

Related Information

- add_instance on page 718
- get_instance_interfaces on page 724

8.1.6.3. get_connections

Description

Returns a list of all connections in the composed subsystem.

Availability

Main Program, Composition

Usage

```
get_connections
```

Returns

A list of connections.

Arguments

No arguments.

Example

```
set all_connections [ get_connections ]
```

Related Information

add_connection on page 719

8.1.6.4. get_connection_parameters

Description

Returns a list of parameters found on a connection.

Availability

Main Program, Composition

Usage

`get_connection_parameters` <*connection*>

Returns

A list of parameter names

Arguments

connection The connection to query.

Example

```
get_connection_parameters cpu.data_host/dma0.csr
```

Related Information

- add_connection on page 719
- get_connection_parameter_value on page 722

8.1.6.5. get_connection_parameter_value

Description

Returns the value of a parameter on the connection. Parameters represent aspects of the connection that can be modified once the connection is created, such as the base address for an Avalon Memory Mapped connection.

Availability

Composition

Usage

`get_connection_parameter_value` <*connection*> <*parameter*>

Returns

The value of the parameter.

Arguments

connection The connection to query.

parameter The name of the parameter.

Example

```
get_connection_parameter_value cpu.data_host/dma0.csr baseAddress
```

Related Information

- add_connection on page 719
- get_connection_parameters on page 721

8.1.6.6. get_instances

Description

Returns a list of the instance names for all child instances in the system.

Availability

Main Program, Elaboration, Validation, Composition

Usage

```
get_instances
```

Returns

A list of child instance names.

Arguments

No arguments.

Example

```
get_instances
```

Notes

This command can be used with instances created by either `add_instance` or `add_hdl_instance`.

Related Information

- add_hdl_instance on page 715
- add_instance on page 718
- get_instance_parameter_value on page 733
- get_instance_parameters on page 731
- set_instance_parameter_value on page 736

8.1.6.7. get_instance_interfaces

Description

Returns a list of interfaces found in a child instance. The list of interfaces can change if the parameterization of the instance changes.

Availability

Validation, Composition

Usage

`get_instance_interfaces` <*instance*>

Returns

A list of interface names.

Arguments

instance The name of the child instance.

Example

```
get_instance_interfaces pixel_converter
```

Related Information

- add_instance on page 718
- get_instance_interface_ports on page 725
- get_instance_interfaces on page 724

8.1.6.8. get_instance_interface_ports

Description

Returns a list of ports found in an interface of a child instance.

Availability

Validation, Composition, Fileset Generation

Usage

`get_instance_interface_ports` <*instance*> <*interface*>

Returns

A list of port names found in the interface.

Arguments

instance The name of the child instance.

interface The name of an interface on the child instance.

Example

```
set port_names [ get_instance_interface_ports cpu data_host ]
```

Related Information

- add_instance on page 718
- get_instance_interfaces on page 724
- get_instance_port_property on page 734

8.1.6.9. get_instance_interface_properties

Description

Returns the names of all of the properties of the specified interface

Availability

Validation, Composition

Usage

`get_instance_interface_properties` <*instance*> <*interface*>

Returns

List of property names.

Arguments

instance The name of the child instance.

interface The name of an interface on the instance.

Example

```
set properties [ get_instance_interface_properties cpu data_host ]
```

Related Information

- add_instance on page 718
- get_instance_interface_property on page 730
- get_instance_interfaces on page 724

8.1.6.10. get_instance_property

Description

Returns the value of a single instance property.

Availability

Main Program, Elaboration, Validation, Composition, Fileset Generation

Usage

`get_instance_property` <*instance*> <*property*>

Returns

Various.

Arguments

instance The name of the instance.

property The name of the property. Refer to *Instance Properties*.

Example

```
set my_name [ get_instance_property myinstance NAME ]
```

Related Information

- add_instance on page 718
- get_instance_properties on page 729
- set_instance_property on page 728
- Instance Properties on page 767

8.1.6.11. set_instance_property

Description

Allows a user to set the properties of a child instance.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`set_instance_property` <*instance*> <*property*> <*value*>

Returns

Arguments

instance The name of the instance.

property The name of the property to set. Refer to *Instance Properties*.

value The new property value.

Example

```
set_instance_property myinstance SUPRESS_ALL_WARNINGS true
```

Related Information

- add_instance on page 718
- get_instance_properties on page 729
- get_instance_property on page 727
- Instance Properties on page 767

8.1.6.12. get_instance_properties

Description

Returns the names of all the instance properties as a list of strings. You can use the `get_instance_property` and `set_instance_property` commands to get and set values of individual properties. The value returned by this command is always the same for a particular version of Platform Designer

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_instance_properties
```

Returns

List of strings. Refer to *Instance Properties*.

Arguments

No arguments.

Example

```
get_instance_properties
```

Related Information

- add_instance on page 718
- get_instance_property on page 727
- set_instance_property on page 728
- Instance Properties on page 767

8.1.6.13. get_instance_interface_property

Description

Returns the value of a property for an interface in a child instance.

Availability

Validation, Composition

Usage

`get_instance_interface_property` <*instance*> <*interface*> <*property*>

Returns

The value of the property.

Arguments

instance The name of the child instance.

interface The name of an interface on the child instance.

property The name of the property of the interface.

Example

```
set value [ get_instance_interface_property cpu data_host setupTime ]
```

Related Information

- add_instance on page 718
- get_instance_interfaces on page 724

8.1.6.14. get_instance_parameters

Description

Returns a list of names of the parameters on a child instance that can be set using `set_instance_parameter_value`. It omits parameters that are derived and those that have the `SYSTEM_INFO` parameter property set.

Availability

Main Program, Elaboration, Validation, Composition

Usage

`get_instance_parameters` <*instance*>

Returns

A list of parameters in the instance.

Arguments

instance The name of the child instance.

Example

```
set parameters [ get_instance_parameters instance ]
```

Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

Related Information

- add_hdl_instance on page 715
- add_instance on page 718
- get_instance_parameter_value on page 733
- get_instances on page 723
- set_instance_parameter_value on page 736

8.1.6.15. get_instance_parameter_property

Description

Returns the value of a property on a parameter in a child instance. Parameter properties are metadata that describe how the Platform Designer tools use the parameter.

Availability

Validation, Composition

Usage

`get_instance_parameter_property` <*instance*> <*parameter*> <*property*>

Returns

The value of the parameter property.

Arguments

instance The name of the child instance.

parameter The name of the parameter in the instance.

property The name of the property of the parameter. Refer to *Parameter Properties*.

Example

```
get_instance_parameter_property instance parameter property
```

Related Information

- add_instance on page 718
- Parameter Properties on page 768

8.1.6.16. get_instance_parameter_value

Description

Returns the value of a parameter in a child instance. You cannot use this command to get the value of parameters whose values are derived or those that are defined using the SYSTEM_INFO parameter property.

Availability

Elaboration, Validation, Composition

Usage

`get_instance_parameter_value` <*instance*> <*parameter*>

Returns

The value of the parameter.

Arguments

instance The name of the child instance.

parameter Specifies the parameter whose value is being retrieved.

Example

```
set dpi [ get_instance_parameter_value pixel_converter input_DPI ]
```

Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

Related Information

- add_hdl_instance on page 715
- add_instance on page 718
- get_instance_parameters on page 731
- get_instances on page 723
- set_instance_parameter_value on page 736

8.1.6.17. get_instance_port_property

Description

Returns the value of a property of a port contained by an interface in a child instance.

Availability

Validation, Composition, Fileset Generation

Usage

`get_instance_port_property` <*instance*> <*port*> <*property*>

Returns

The value of the property for the port.

Arguments

instance The name of the child instance.

port The name of a port in one of the interfaces on the child instance.

property The property whose value is being retrieved. Only the following port properties can be queried on ports of child instances: `ROLE`, `DIRECTION`, `WIDTH`, `WIDTH_EXPR` and `VHDL_TYPE`. Refer to *Port Properties*.

Example

```
get_instance_port_property instance port property
```

Related Information

- add_instance on page 718
- get_instance_interface_ports on page 725
- Port Properties on page 772

8.1.6.18. set_connection_parameter_value

Description

Sets the value of a parameter of the connection. The start and end are each interface names of the format `<instance>.<interface>`. Connection parameters depend on the type of connection, for Avalon memory mapped they include base addresses and arbitration priorities.

Availability

Main Program, Composition

Usage

`set_connection_parameter_value` <*connection*> <*parameter*> <*value*>

Returns

No return value.

Arguments

connection Specifies the name of the connection as returned by the `add_conection` command. It is of the form `start.point/end.point`.

parameter The name of the parameter.

value The new parameter value.

Example

```
set_connection_parameter_value cpu.data_host/dma0.csr baseAddress "0x000a0000"
```

Related Information

- add_connection on page 719
- get_connection_parameter_value on page 722

8.1.6.19. set_instance_parameter_value

Description

Sets the value of a parameter for a child instance. Derived parameters and SYSTEM_INFO parameters for the child instance cannot be set with this command.

Availability

Main Program, Elaboration, Composition

Usage

`set_instance_parameter_value` <*instance*> <*parameter*> <*value*>

Returns

Vo return value.

Arguments

instance Specifies the name of the child instance.

parameter Specifies the parameter that is being set.

value Specifies the new parameter value.

Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

Notes

You can use this command with instances created by either `add_instance` or `add_hdl_instance`.

Related Information

- add_hdl_instance on page 715
- add_instance on page 718
- get_instance_parameter_value on page 733
- get_instances on page 723

8.1.7. Fileset Generation

8.1.7.1. add_fileset

Description

Adds a generation fileset for a particular target as specified by the `kind`. Platform Designer calls the target (`SIM_VHDL`, `SIM_VERILOG`, `QUARTUS_SYNTH`, or `EXAMPLE_DESIGN`) when the specified generation target is requested. You can define multiple filesets for each kind of fileset. Platform Designer passes a single argument to the specified callback procedure. The value of the argument is a generated name, which you must use in the top-level module or entity declaration of your IP component. To override this generated name, you can set the fileset property `TOP_LEVEL`.

Availability

Main Program

Usage

`add_fileset` <*name*> <*kind*> [<*callback_proc*> <*display_name*>]

Returns

No return value.

Arguments

name The name of the fileset.

kind The kind of fileset. Refer to *Fileset Properties*.

callback_proc (optional) A string identifying the name of the callback procedure. If you add files in the global section, you can then specify a blank callback procedure.

display_name (optional) A display string to identify the fileset.

Example

```
add_fileset my_synthesis_fileset QUARTUS_SYNTH mySynthCallbackProc "My Synthesis"
proc mySynthCallbackProc { topLevelName } { ... }
```

Notes

If using the `TOP_LEVEL` fileset property, all parameterizations of the component must use identical HDL.

Related Information

- add_fileset_file on page 739
- get_fileset_property on page 744
- Fileset Properties on page 780

8.1.7.2. add_fileset_file

Description

Adds a file to the generation directory. You can specify source file locations with either an absolute path, or a path relative to the IP component's `_hw.tcl` file. When you use the `add_fileset_file` command in a fileset callback, the Quartus Prime software compiles the files in the order that they are added.

Availability

Main Program, Fileset Generation

Usage

`add_fileset_file` <*output_file*> <*file_type*> <*file_source*> <*path_or_contents*> [<*attributes*>]

Returns

No return value.

Arguments

output_file Specifies the location to store the file after Platform Designer generation

file_type The kind of file. Refer to *File Kind Properties*.

file_source Specifies whether the file is being added by path, or by file contents. Refer to *File Source Properties*.

path_or_contents When the `file_source` is PATH, specifies the file to be copied to `output_file`. When the `file_source` is TEXT, specifies the text contents to be stored in the file.

attributes (optional) An optional list of file attributes. Typically used to specify that a file is intended for use only in a particular simulator. Refer to *File Attribute Properties*.

Example

```
add_fileset_file "./implementation/rx_pma.sv" SYSTEM_VERILOG PATH synth_rx_pma.sv
add_fileset_file gui.sv SYSTEM_VERILOG TEXT "Customize your IP core"
```

Related Information

- add_fileset on page 738
- get_fileset_file_attribute on page 741
- File Kind Properties on page 784
- File Source Properties on page 785
- File Attribute Properties on page 783

8.1.7.3. set_fileset_property

Description

Allows you to set the properties of a fileset.

Availability

Main Program, Elaboration, Fileset Generation

Usage

`set_fileset_property` <*fileset*> <*property*> <*value*>

Returns

No return value.

Arguments

fileset The name of the fileset.

property The name of the property to set. Refer to *Fileset Properties*.

value The new property value.

Example

```
set_fileset_property mySynthFileset TOP_LEVEL simple_uart
```

Notes

When a fileset callback is called, the callback procedure is passed a single argument. The value of this argument is a generated name which must be used in the top-level module or entity declaration of your IP component. If set, the TOP_LEVEL specifies a fixed name for the top-level name of your IP component.

The TOP_LEVEL property must be set in the global section. It cannot be set in a fileset callback.

If using the TOP_LEVEL fileset property, all parameterizations of the IP component must use identical HDL.

Related Information

- add_fileset on page 738
- Fileset Properties on page 780

8.1.7.4. get_fileset_file_attribute

Description

Returns the attribute of a fileset file.

Availability

Main Program, Fileset Generation

Usage

`get_fileset_file_attribute` <*output_file*> <*attribute*>

Returns

Value of the fileset File attribute.

Arguments

output_file Location of the output file.

attribute Specifies the name of the attribute Refer to *File Attribute Properties*.

Example

```
get_fileset_file_attribute my_file.sv ALDEC_SPECIFIC
```

Related Information

- add_fileset on page 738
- add_fileset_file on page 739
- get_fileset_file_attribute on page 741
- File Attribute Properties on page 783
- add_fileset on page 738
- add_fileset_file on page 739
- get_fileset_file_attribute on page 741
- File Attribute Properties on page 783

8.1.7.5. set_fileset_file_attribute

Description

Sets the attribute of a fileset file.

Availability

Main Program, Fileset Generation

Usage

`set_fileset_file_attribute` <*output_file*> <*attribute*> <*value*>

Returns

The attribute value if it was set.

Arguments

output_file Location of the output file.

attribute Specifies the name of the attribute Refer to *File Attribute Properties*.

value Value to set the attribute to.

Example

```
set_fileset_file_attribute my_file_pkg.sv COMMON_SYSTEMVERILOG_PACKAGE
my_file_package
```

8.1.7.6. get_fileset_properties

Description

Returns a list of properties that can be set on a fileset.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Generation, Composition, Fileset Generation, Parameter Upgrade

Usage

```
get_fileset_properties
```

Returns

A list of property names. Refer to *Fileset Properties*.

Arguments

No arguments.

Example

```
get_fileset_properties
```

Related Information

- add_fileset on page 738
- get_fileset_properties on page 743
- set_fileset_property on page 740
- Fileset Properties on page 780

8.1.7.7. get_fileset_property

Description

Returns the value of a fileset property for a fileset.

Availability

Main Program, Elaboration, Fileset Generation

Usage

`get_fileset_property` <*fileset*> <*property*>

Returns

The value of the property.

Arguments

fileset The name of the fileset.

property The name of the property to query. Refer to *Fileset Properties*.

Example

```
get_fileset_property fileset property
```

Related Information

Fileset Properties on page 780

8.1.7.8. get_fileset_sim_properties

Description

Returns simulator properties for a fileset.

Availability

Main Program, Fileset Generation

Usage

`get_fileset_sim_properties` <*fileset*> <*platform*> <*property*>

Returns

The fileset simulator properties.

Arguments

fileset The name of the fileset.

platform The operating system that applies to the property. Refer to *Operating System Properties*.

property Specifies the name of the property to set. Refer to *Simulator Properties*.

Example

```
get_fileset_sim_properties my_fileset LINUX64 OPT_CADENCE_64BIT
```

Related Information

- add_fileset on page 738
- set_fileset_sim_properties on page 746
- Operating System Properties on page 792
- Simulator Properties on page 786

8.1.7.9. set_fileset_sim_properties

Description

Sets simulator properties for a given fileset

Availability

Main Program, Fileset Generation

Usage

`set_fileset_sim_properties` <*fileset*> <*platform*> <*property*> <*value*>

Returns

The fileset simulator properties if they were set.

Arguments

fileset The name of the fileset.

platform The operating system that applies to the property. Refer to *Operating System Properties*.

property Specifies the name of the property to set. Refer to *Simulator Properties*.

value Specifies the value of the property.

Example

```
set_fileset_sim_properties my_fileset LINUX64 OPT_MENTOR_PLI "{libA} {libB}"
```

Related Information

- get_fileset_sim_properties on page 745
- Operating System Properties on page 792
- Simulator Properties on page 786

8.1.7.10. create_temp_file

Description

Creates a temporary file, which you can use inside the fileset callbacks of a `_hw.tcl` file. This temporary file is included in the generation output if it is added using the `add_fileset_file` command.

Availability

Fileset Generation

Usage

`create_temp_file` <*path*>

Returns

The path to the temporary file.

Arguments

path The name of the temporary file.

Example

```
set filelocation [create_temp_file "./hdl/compute_frequency.v" ]
add_fileset_file compute_frequency.v VERILOG PATH ${filelocation}
```

Related Information

- add_fileset on page 738
- add_fileset_file on page 739

8.1.8. Miscellaneous

8.1.8.1. check_device_family_equivalence

Description

Returns 1 if the device family is equivalent to one of the families in the device families list. Returns 0 if the device family is not equivalent to any families. This command ignores differences in capitalization and spaces.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Fileset Generation, Parameter Upgrade

Usage

`check_device_family_equivalence` <*device_family*> <*device_family_list*>

Returns

1 if equivalent, 0 if not equivalent.

Arguments

device_family The device family name that is being checked.

device_family_list The list of device family names to check against.

Example

```
check_device_family_equivalence "CYLCONE III LS" { "stratixv" "Cyclone IV"
"cycloneiiils" }
```

Related Information

get_device_family_displayname on page 750

8.1.8.2. get_device_family_displayname

Description

Returns the display name of a given device family.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Fileset Generation, Parameter Upgrade

Usage

`get_device_family_displayname <device_family>`

Returns

The preferred display name for the device family.

Arguments

device_family A device family name.

Example

```
get_device_family_displayname cycloneiiils ( returns: "Cyclone IV LS" )
```

Related Information

check_device_family_equivalence on page 749

8.1.8.3. get_qip_strings

Description

Returns a Tcl list of QIP strings for the IP component.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Parameter Upgrade

Usage

```
get_qip_strings
```

Returns

A Tcl list of qip strings set by this IP component.

Arguments

No arguments.

Example

```
set strings [ get_qip_strings ]
```

Related Information

set_qip_strings on page 752

8.1.8.4. set_qip_strings

Description

Places strings in the Quartus Prime IP File (**.qip**) file, which Platform Designer passes to the command as a Tcl list. You add the **.qip** file to your Quartus Prime project on the **Files** page, in the **Settings** dialog box. Successive calls to `set_qip_strings` are not additive and replace the previously declared value.

Availability

Discovery, Main Program, Edit, Elaboration, Validation, Composition, Parameter Upgrade

Usage

`set_qip_strings` <*qip_strings*>

Returns

The Tcl list which was set.

Arguments

qip_strings A space-delimited Tcl list.

Example

```
set_qip_strings {"QIP Entry 1" "QIP Entry 2"}
```

Notes

You can use the following macros in your QIP strings entry:

%entityName%	The generated name of the entity replaces this macro when the string is written to the **.qip** file.
%libraryName%	The compilation library this IP component was compiled into is inserted in place of this macro inside the **.qip** file.
%instanceName%	The name of the instance is inserted in place of this macro inside the `.qip` file.

Related Information

get_qip_strings on page 751

8.1.9. SystemVerilog Interface Commands

8.1.9.1. add_sv_interface

Description

Adds a SystemVerilog interface to the IP component.

Availability

Elaboration, Global

Usage

`add_sv_interface` <*sv_interface_name*> <*sv_interface_type*>

Returns

No return value.

Arguments

sv_interface_name	The name of the SystemVerilog interface in the IP component.
sv_interface_type	The type of the SystemVerilog interface used by the IP component.

Example

```
add_sv_interface my_sv_interface my_sv_interface_type
```

8.1.9.2. get_sv_interfaces

Description

Returns the list of SystemVerilog interfaces in the IP component.

Availability

Elaboration, Global

Usage

```
get_sv_interfaces
```

Returns

String[] Returns the list of SystemVerilog interfaces defined in the IP component.

Arguments

No arguments.

Example

```
get_sv_interfaces
```

8.1.9.3. get_sv_interface_property

Description

Returns the value of a single SystemVerilog interface property from the specified interface.

Availability

Elaboration, Global

Usage

`get_sv_interface_property` <*sv_interface_name*> <*sv_interface_property*>

Returns

various The property value.

Arguments

sv_interface_name The name of a SystemVerilog interface of the system.

sv_interface_property The name of the property. Refer to *System Verilog Interface Properties*.

Example

```
get_sv_interface_property my_sv_interface USE_ALL_PORTS
```

8.1.9.4. get_sv_interface_properties

Description

Returns the names of all the available SystemVerilog interface properties common to all interface types.

Availability

Elaboration, Global

Usage

```
get_sv_interface_properties
```

Returns

String[] The list of SystemVerilog interface properties.

Arguments

No arguments.

Example

```
get_sv_interface_properties
```

8.1.9.5. set_sv_interface_property

Description

Sets the value of a property on a SystemVerilog interface.

Availability

Elaboration, Global

Usage

`set_sv_interface_property` <*sv_interface_name*> <*sv_interface_property*> <*value*>

Returns

No return value.

Arguments

interface The name of a SystemVerilog interface.

sv_interface_property The name of the property. Refer to *SystemVerilog Interface Properties*.

value The property value.

Example

```
set_sv_interface_property my_sv_interface USE_ALL_PORTS True
```

8.1.10. Wire-Level Expression Commands

8.1.10.1. set_wirelevel_expression

Description

Applies a wire-level expression to an optional input port or instance in the system.

Usage

`set_wirelevel_expression` <*instance_or_port_bitselection*> <*expression*>

Returns

No return value.

Arguments

instance_or_port_bitselection	Specify the instance or port to which the wire-level expression using the <*instance_name*>.<*port_name*>[<*bit_selection*>] format. The *bit selection* can be a bit-select, for example `[0]`, or a partial range defined in descending order, for example `[7:0]`. If no *bit selection* is specified, the full range of the port is selected.
expression	The expression to be applied to an optional input port.

Examples

```
set_wirelevel_expression {module0.portA[7:0]} "8'b0"
set_wirelevel_expression module0.portA "8'b0"
set_wirelevel_expression {module0.portA[0]} "1'b0"
```

Related Information

Scripting Wire-Level Expressions on page 71

8.1.10.2. get_wirelevel_expressions

Description

Retrieve a list of wire-level expressions from an optional input port, instance, or all expressions in the current level of system hierarchy. If the port *bit selection* is specified as an argument, the range must be identical to what was used in the `set_wirelevel_expression` statement.

Usage

`get_wirelevel_expressions` <*instance_or_port_bitselection*>

Returns

String[] A flattened list of wire-level expressions. Every item in the list consists of right- and left-hand clauses of a wire-level expression. You can loop over the returned list using `foreach{port expr} $return_list{}`.

Arguments

instance_or_port_bitselection Specifies which instance or port from which a list of wire-level expressions are retrieved using the <*instance_name*>.<*port_name*>[<*bit_selection*>] format.

- If no <*port_name*>*[<bit_selection>]* is specified, the command causes the return of all expressions from the specified instance.
- If no argument is present, the command causes the return of all expressions from the current level of system hierarchy.

The *bit selection* can be a bit-select, for example `[0]`, or a partial range defined in descending order, for example `[7:0]`. If no *bit selection* is specified, the full range of the port is selected.

Example

```
get_wirelevel_expressions
get_wirelevel_expressions module0
get_wirelevel_expressions {module0.portA[7:0]}
```

Related Information

Scripting Wire-Level Expressions on page 71

8.1.10.3. remove_wirelevel_expressions

Description

Remove a list of wire-level expressions from an optional input port, instance, or all expressions in the current level of system hierarchy. If the port *bit selection* is specified as an argument, the range must be identical to what was used in the `set_wirelevel_expressions` statement.

Usage

`remove_wirelevel_expressions` <*instance_or_port_bitselection*>

Returns

No return value.

Arguments

instance_or_port_bitselection	Specifies which instance or port from which a list of wire-level expressions are removed using the <*instance_name*>.<*port_name*>[<*bit_selection*>] format. • If no <*port_name*>*[<bit_selection>]* is specified, the command causes the removal of all expressions from the specified instance. • If no argument is present, the command causes the return of all expressions from the current level of system hierarchy. The *bit selection* can be a bit-select, for example `[0]`, or a partial range defined in descending order, for example `[7:0]`. If no *bit selection* is specified, the full range of the port is selected.

Examples

```
remove_wirelevel_expressions
remove_wirelevel_expressions module0
remove_wirelevel_expressions {module0.portA[7:0]}
```

Related Information

Scripting Wire-Level Expressions on page 71

8.2. Platform Designer _hw.tcl Property Reference

8.2.1. Script Language Properties

Name	Description
TCL	Implements the script in Tcl.

8.2.2. Interface Properties

Name	Description
CMSIS_SVD_FILE	Specifies the connection point's associated CMSIS file.
CMSIS_SVD_VARIABLES	Defines the variables inside a .svd file.
ENABLED	Specifies whether or not interface is enabled.
EXPORT_OF	For composed `_hwl.tcl` files, the EXPORT_OF property indicates which interface of a child instance is to be exported through this interface. Before using this command, you must have created the border interface using `add_interface`. The interface to be exported is of the form <*instanceName.interfaceName*>. Example: `set_interface_property CSC_input EXPORT_OF my_colorSpaceConverter.input_port`
PORT_NAME_MAP	A map of external port names to internal port names, formatted as a Tcl list. Example: `set_interface_property <interface name> PORT_NAME_MAP "<new port name> <old port name> <new port name 2> <old port name 2>"`
SVD_ADDRESS_GROUP	Generates a CMSIS SVD file. Hosts in the same SVD address group write register data of their connected agents into the same SVD file
SVD_ADDRESS_OFFSET	Generates a CMSIS SVD file. Agents connected to this host have their base address offset by this amount in the SVD file.
SV_INTERFACE	When SV_INTERFACE is set, all the ports in the given interface are part of the SystemVerilog interface. Example: `set_interface_property my_qsys_interface SV_INTERFACE my_sv_interface`
IPXACT_REGISTER_MAP	Specifies the connection point's associated IP-XACT register map file. Platform Designer supports register map files in IP-XACT 2009 or 2014 format. Example: `set_interface_property my_qsys_interface IPXACT_REGISTER_MAP <path_to_ipxact_reg_file>`
IPXACT_REGISTER_MAP_VARIABLES	For macro substitution inside the IP-XACT register map file. Specifies a list of key value pairs, where key is the macro name and value is the replacement text that substitutes the macros in the IP-XACT register map.

Related Information

Interfaces and Ports on page 667

8.2.3. SystemVerilog Interface Properties

Name	Description
SV_INTERFACE_TYPE	Set the interface type of the SystemVerilog interface.

Name	Description
USE_ALL_PORTS	When USE_ALL_PORTS is set to true, all the ports defined in the Module, are declared in this SystemVerilog interface. USE_ALL_PORTS must be set to true only if the module has one SystemVerilog interface and the SystemVerilog interface signal names match with the port names declared for Platform Designer interface. When USE_ALL_PORTS is true, SV_INTERFACE_PORT or SV_INTERFACE_SIGNAL port properties should not be set.

8.2.4. Instance Properties

Name	Description
HDLINSTANCE_GET_GENERATED_NAME	Platform Designer uses this property to get the auto-generated fixed name when the instance property HDLINSTANCE_USE_GENERATED_NAME is set to true, and only applies to fileSet callbacks.
HDLINSTANCE_USE_GENERATED_NAME	If true, instances added with the `add_hdl_instance` command are instructed to use unique auto-generated fixed names based on the parameterization.
SUPPRESS_ALL_INFO_MESSAGES	If true, allows you to suppress all Info messages that originate from a child instance.
SUPPRESS_ALL_WARNINGS	If true, allows you to suppress all warnings that originate from a child instance

8.2.5. Parameter Properties

Type	Name	Description
Boolean	AFFECTS_ELABORATION	Set AFFECTS_ELABORATION to false for parameters that do not affect the external interface of the module. An example of a parameter that does not affect the external interface is isNonVolatileStorage. An example of a parameter that does affect the external interface is width. When the value of a parameter changes, if that parameter has set AFFECTS_ELABORATION=false, the elaboration phase (calling the callback or hardware analysis) is not repeated, improving performance. Because the default value of AFFECTS_ELABORATION is true, the provided HDL file is normally re-analyzed to determine the new port widths and configuration every time a parameter changes.
Boolean	AFFECTS_GENERATION	The default value of AFFECTS_GENERATION is false if you provide a top-level HDL module; it is true if you provide a fileset callback. Set AFFECTS_GENERATION to false if the value of a parameter does not change the results of fileset generation.
Boolean	AFFECTS_VALIDATION	The AFFECTS_VALIDATION property marks whether a parameter's value is used to set derived parameters, and whether the value affects validation messages. When set to false, this may improve response time in the parameter editor UI when the value is changed.
String[]	ALLOWED_RANGES	Indicates the range or ranges that the parameter value can have. For integers, The ALLOWED_RANGES property is a list of ranges that the parameter can take on, where each range is a single value, or a range of values defined by a start and end value separated by a colon, such as 11:15. This property can also specify legal values and display strings for integers, such as {0:None 1:Monophonic 2:Stereo 4:Quadrophonic} meaning 0, 1, 2, and 4 are the legal values. You can also assign display strings to be displayed in the parameter editor for string variables. For example, ALLOWED_RANGES {"dev1:Cyclone IV GX""dev2:Stratix V GT"}.
String	DEFAULT_VALUE	The default value.
Boolean	DERIVED	When true, indicates that the parameter value can only be set by the IP component, and cannot be set by the user. Derived parameters are not saved as part of an instance's parameter values. The default value is false.
String	DESCRIPTION	A short user-visible description of the parameter, suitable for a tooltip description in the parameter editor.
String[]	DISPLAY_HINT	Provides a hint about how to display a property. The following values are possible: • boolean--for integer parameters whose value can be 0 or 1. The parameter displays as an option that you can turn on or off. • radio--displays a parameter with a list of values as radio buttons instead of a drop-down list. • hexadecimal--for integer parameters, display and interpret the value as a hexadecimal number, for example: 0x00000010 instead of 16. • fixed_size--for string_list and integer_list parameters, the fixed_size DISPLAY_HINT eliminates the **add** and **remove** buttons from tables.
String	DISPLAY_NAME	This is the GUI label that appears to the left of this parameter.
String	DISPLAY_UNITS	This is the GUI label that appears to the right of the parameter.

Type	Name	Description
Boolean	ENABLED	When `false`, the parameter is disabled, meaning that it is displayed, but greyed out, indicating that it is not editable on the parameter editor.
String	GROUP	Controls the layout of parameters in GUI
Boolean	HDL_PARAMETER	When true, the parameter must be passed to the HDL IP component description. The default value is `false`.
String	LONG_DESCRIPTION	A user-visible description of the parameter. Similar to DESCRIPTION, but allows for a more detailed explanation.
String	NEW_INSTANCE_VALUE	This property allows you to change the default value of a parameter without affecting older IP components that have did not explicitly set a parameter value, and use the DEFAULT_VALUE property. The practical result is that older instances continue to use DEFAULT_VALUE for the parameter and new instances use the value that NEW_INSTANCE_VALUE assigns.
String	SV_INTERFACE_PARAMETER	This parameter is used in the SystemVerilog interface instantiation. Example: `set_parameter_property my_parameter SV_INTERFACE_PARAMETER my_sv_interface`
String[]	SYSTEM_INFO	Allows you to assign information about the instantiating system to a parameter that you define. SYSTEM_INFO requires an argument specifying the type of information requested, *<info-type>*.
String	SYSTEM_INFO_ARG	Defines an argument to be passed to a particular SYSTEM_INFO function, such as the name of a reset interface.
(various)	SYSTEM_INFO_TYPE	Specifies one of the types of system information that can be queried. Refer to *System Info Type Properties*.
(various)	TYPE	Specifies the type of the parameter. Refer to *Parameter Type Properties*.
(various)	UNITS	Sets the units of the parameter. Refer to *Units Properties*.
Boolean	VISIBLE	Indicates whether or not to display the parameter in the parameterization GUI.
String	WIDTH	For a STD_LOGIC_VECTOR parameter, this indicates the width of the logic vector.

Related Information

- System Info Type Properties on page 788
- Parameter Type Properties on page 770
- Units Properties on page 791

8.2.6. Parameter Type Properties

Name	Description
BOOLEAN	A boolean parameter whose value is `true` or `false`.
FLOAT	A signed 32-bit floating point parameter. Not supported for HDL parameters.
INTEGER	A signed 32-bit integer parameter.
INTEGER_LIST	A parameter that contains a list of 32-bit integers. Not supported for HDL parameters.
LONG	A signed 64-bit integer parameter. Not supported for HDL parameters.
NATURAL	A 32-bit number that contain values `0` to `2147483647` (`0x7fffffff`).
POSITIVE	A 32-bit number that contains values `1` to `2147483647` (`0x7fffffff`).
STD_LOGIC	A single bit parameter whose value can be `1` or `0`;
STD_LOGIC_VECTOR	An arbitrary-width number. The parameter property `WIDTH` determines the size of the logic vector.
STRING	A string parameter.
STRING_LIST	A parameter that contains a list of strings. Not supported for HDL parameters.

8.2.7. Parameter Status Properties

Type	Name	Description
Boolean	ACTIVE	Indicates the parameter is a regular parameter.
Boolean	DEPRECATED	Indicates the parameter exists only for backwards compatibility, and may not have any effect.
Boolean	EXPERIMENTAL	Indicates the parameter is experimental, and not exposed in the design flow.

8.2.8. Port Properties

Type	Name	Description
(various)	DIRECTION	The direction of the port from the IP component's perspective. Refer to *Direction Properties*.
String	DRIVEN_BY	Indicates that this output port is always driven to a constant value or by an input port. If all outputs on an IP component specify a driven_by property, the HDL for the IP component is generated automatically.
String[]	FRAGMENT_LIST	This property can be used in 2 ways: First you can take a single RTL signal and split it into multiple Platform Designer signals add_interface_port <interface> alpha <role> <direction> <width> add_interface_port <interface> bar <role> <direction> <width> set_port_property alpha fragment_list "my_rtl_signal(3:0)" set_port_property bar fragment_list "my_rtl_signal(6:4)" Second you can take multiple RTL signals and combine them into a single Platform Designer signal add_interface_port <interface> baz <role> <direction> <width> set_port_property baz fragment_list "rtl_signal_1(3:0) rtl_signal_2(3:0)" Note: The listed bits in a port fragment must match the declared width of the Platform Designer signal.
String	ROLE	Specifies an Avalon signal type such as waitrequest, readdata, or read. For a complete list of signal types, refer to the *Avalon Interface Specifications*.
String	SV_INTERFACE_PORT	This port from the module is used as I/O in the SystemVerilog interface instantiation. The top-level wrapper of the module which contains this port is from the SystemVerilog interface. Example: `set_port_property port_x SV_INTERFACE_PORT my_sv_interface`
String	SV_INTERFACE_PORT_NAME	This property is used only when the Platform Designer port name defined for the module is different from the port name in the SystemVerilog interface. Example: `set_port_property port_x SV_INTERFACE_PORT_NAME port_a` When writing the RTL, the Platform Designer port name port_x is mapped to RTL name port_a in the SystemVerilog interface
String	SV_INTERFACE_SIGNAL	This port from the module is assumed to be inside the SystemVerilog interface or the modport used by the module. The top-level wrapper of the module containing this port is unwrapped from SystemVerilog interface. Example: `set_port_property port_y SV_INTERFACE_SIGNAL my_sv_interface`
String	SV_INTERFACE_SIGNAL_NAME	This property is only used when the Platform Designer port name defined for the module is different from the port name in the SystemVerilog interface. Example: `set_port_property port_y SV_INTERFACE_SIGNAL_NAME port_b`
Boolean	TERMINATION	When true, instead of connecting the port to the Platform Designer system, it is left unconnected for output and bidir or set to a fixed value for input.

Type	Name	Description
BigInteger	TERMINATION_VALUE	The constant value to drive an input port.
(various)	VHDL_TYPE	Indicates the type of a VHDL port. The default value, auto, selects std_logic if the width is fixed at 1, and std_logic_vector otherwise. Refer to *Port VHDL Type Properties*.
String	WIDTH	The width of the port in bits. Cannot be set directly. Any changes must be set through the WIDTH_EXPR property.
String	WIDTH_EXPR	The width expression of a port. The width_value_expr property can be set directly to a numeric value if desired. When get_port_property is used width always returns the current integer width of the port while width_expr always returns the unevaluated width expression.
Integer	WIDTH_VALUE	The width of the port in bits. Cannot be set directly. Any changes must be set through the WIDTH_EXPR property.

Related Information

- Direction Properties on page 774
- Port VHDL Type Properties on page 787
- Avalon Interface Specifications

8.2.9. Direction Properties

Name	Description
`Bidir`	Direction for a bidirectional signal.
`Input`	Direction for an input signal.
`Output`	Direction for an output signal.

8.2.10. Display Item Properties

Type	Name	Description
String	DESCRIPTION	A description of the display item, which you can use as a tooltip.
String[]	DISPLAY_HINT	A hint that affects how the display item displays in the parameter editor.
String	DISPLAY_NAME	The label for the display item in a the parameter editor.
Boolean	ENABLED	Indicates whether the display item is enabled or disabled.
String	PATH	The path to a file. Only applies to display items of type ICON.
String	TEXT	Text associated with a display item. Only applies to display items of type TEXT.
Boolean	VISIBLE	Indicates whether this display item is visible or not.

8.2.11. Display Item Kind Properties

Name	Description
ACTION	An action displays as a button in the GUI. When the button is clicked, it calls the callback procedure. The button label is the display item `id`.
GROUP	A group that is a child of the `parent_group` group. If the `parent_group` is an empty string, this is a top-level group.
ICON	A `.gif`, `.jpg`, or `.png` file.
PARAMETER	A parameter in the instance.
TEXT	A block of text.

8.2.12. Display Hint Properties

Name	Description
BIT_WIDTH	Bit width of a number
BOOLEAN	Integer value either 0 or 1.
COLLAPSED	Indicates whether a group is collapsed when initially displayed.
COLUMNS	Number of columns in text field, for example, "columns:N".
EDITABLE	Indicates whether a list of strings allows free-form text entry (editable combo box).
FILE	Indicates that the string is an optional file path, for example, "file:jpg,png,gif".
FIXED_SIZE	Indicates a fixed size for a table or list.
GROW	if set, the widget can grow when the IP component is resized.
HEXADECIMAL	Indicates that the long integer is hexadecimal.
RADIO	Indicates that the range displays as radio buttons.
ROWS	Number of rows in text field, or visible rows in a table, for example, "rows:N".
SLIDER	Range displays as slider.
TAB	if present for a group, the group displays in a tab
TABLE	if present for a group, the group must contain all list-type parameters, which display collectively in a single table.
TEXT	String is a text field with a limited character set, for example, "text:A-Za-z0-9_".
WIDTH	width of a table column

8.2.13. Module Properties

Name	Description
ANALYZE_HDL	When set to false, prevents a call to the Quartus Prime mapper to verify port widths and directions, speeding up generation time at the expense of fewer validation checks. If this property is set to false, invalid port widths and directions are discovered during the Quartus Prime software compilation. This does not affect IP components using filesets to manage synthesis files.
AUTHOR	The IP component author.
COMPOSITION_CALLBACK	The name of the composition callback. If you define a composition callback, you cannot not define the generation or elaboration callbacks.
DATASHEET_URL	Deprecated. Use `add_documentation_link` to provide documentation links.
DESCRIPTION	The description of the IP component, such as "This IP component implements a half-rate bridge."
DISPLAY_NAME	The name to display when referencing the IP component, such as "My Platform Designer IP Component".
EDITABLE	Indicates whether you can edit the IP component in the Component Editor.
ELABORATION_CALLBACK	The name of the elaboration callback. When set, the IP component's elaboration callback is called to validate and elaborate interfaces for instances of the IP component.
GROUP	The group in the IP Catalog that includes this IP component.
ICON_PATH	A path to an icon to display in the IP component's parameter editor.
INSTANTIATE_IN_SYSTEM_MODULE	If true, this IP component is implemented by HDL provided by the IP component. If false, the IP component creates exported interfaces allowing the implementation to be connected in the parent.
INTERNAL	An IP component which is marked as internal does not appear in the IP Catalog. This feature allows you to hide the sub-IP-components of a larger composed IP component.
MODULE_DIRECTORY	The directory in which the hw.tcl file exists.
MODULE_TCL_FILE	The path to the hw.tcl file.
NAME	The name of the IP component, such as `my_qsys_component`.
OPAQUE_ADDRESS_MAP	For composed IP components created using a _hw.tcl file that include children that are memory-mapped agents, specifies whether the children's addresses are visible to downstream software tools. When `true`, the children's address are not visible. When `false`, the children's addresses are visible.
PREFERRED_SIMULATION_LANGUAGE	The preferred language to use for selecting the fileset for simulation model generation.
REPORT_HIERARCHY	null
STATIC_TOP_LEVEL_MODULE_NAME	Deprecated.
STRUCTURAL_COMPOSITION_CALLBACK	The name of the structural composition callback. This callback is used to represent the structural hierarchical model of the IP component and the RTL can be generated either with module property COMPOSITION_CALLBACK or by ADD_FILESET with target QUARTUS_SYNTH

Name	Description
SUPPORTED_DEVICE_FAMILIES	A list of device family supported by this IP component.
TOP_LEVEL_HDL_FILE	Deprecated.
TOP_LEVEL_HDL_MODULE	Deprecated.
UPGRADEABLE_FROM	null
VALIDATION_CALLBACK	The name of the validation callback procedure.
VERSION	The IP component's version, such as 10.0.

8.2.14. Fileset Properties

Name	Description
ENABLE_FILE_OVERWRITE_MODE	null
ENABLE_RELATIVE_INCLUDE_PATHS	If true, HDL files can include other files using relative paths in the fileset.
TOP_LEVEL	The name of the top-level HDL module that the fileset generates. If set, the HDL top level must match the TOP_LEVEL name, and the HDL must not be parameterized. Platform Designer runs the generate callback one time, regardless of the number of instances in the system.

8.2.15. Fileset Kind Properties

Name	Description
EXAMPLE_DESIGN	Contains example design files.
QUARTUS_SYNTH	Contains files that Platform Designer uses for the Quartus Prime software synthesis.
SIM_VERILOG	Contains files that Platform Designer uses for Verilog HDL simulation.
SIM_VHDL	Contains files that Platform Designer uses for VHDL simulation.
SYSTEMVERILOG_INTERFACE	This file is treated as SystemVerilog interface file by the Platform Designer. Example: `add_fileset_file mem_ifc.sv SYTEM_VERILOG PATH “.ifc/mem_ifc.sv” SYSTEMVERILOG_INTERFACE`

8.2.16. Callback Properties

Description

This list describes each type of callback. Each command may only be available in some callback contexts.

Name	Description
ACTION	Called when an ACTION display item's action is performed.
COMPOSITION	Called during instance elaboration when the IP component contains a subsystem.
EDITOR	Called when the IP component is controlling the parameterization editor.
ELABORATION	Called to elaborate interfaces and signals after a parameter change. In API 9.1 and later, validation is called before elaboration. In API 9.0 and earlier, elaboration is called before validation.
GENERATE_VERILOG_SIMULATION	Called when the IP component uses a custom generator to generates the Verilog simulation model for an instance.
GENERATE_VHDL_SIMULATION	Called when the IP component uses a custom generator to generates the VHDL simulation model for an instance.
GENERATION	Called when the IP component uses a custom generator to generates the synthesis HDL for an instance.
PARAMETER_UPGRADE	Called when attempting to instantiate an IP component with a newer version than the saved version. This allows the IP component to upgrade parameters between released versions of the component.
STRUCTURAL_COMPOSITION	Called during instance elaboration when an IP component is represented by a structural hierarchical model which may be different from the generated RTL.
VALIDATION	Called to validate parameter ranges and report problems with the parameter values. In API 9.1 and later, validation is called before elaboration. In API 9.0 and earlier, elaboration is called before validation.

8.2.17. File Attribute Properties

Name	Description
ALDEC_SPECIFIC	Applies to Aldec simulation tools and for simulation filesets only.
CADENCE_SPECIFIC	Applies to Cadence simulation tools and for simulation filesets only.
COMMON_SYSTEMVERILOG_PACKAGE	The name of the common SystemVerilog package. Applies to simulation filesets only.
MENTOR_SPECIFIC	Applies to Mentor simulation tools and for simulation filesets only.
SYNOPSYS_SPECIFIC	Applies to Synopsys simulation tools and for simulation filesets only.
TOP_LEVEL_FILE	Contains the top-level module for the fileset and applies to synthesis filesets only.

8.2.18. File Kind Properties

Name	Description
DAT	DAT Data
FLI_LIBRARY	FLI Library
HEX	HEX Data
MIF	MIF Data
OTHER	Other
PLI_LIBRARY	PLI Library
SDC	Timing Constraints
SYSTEM_VERILOG	SystemVerilog HDL
SYSTEM_VERILOG_ENCRYPT	Encrypted SystemVerilog HDL
SYSTEM_VERILOG_INCLUDE	SystemVerilog Include
VERILOG	Verilog HDL
VERILOG_ENCRYPT	Encrypted Verilog HDL
VERILOG_INCLUDE	Verilog Include
VHDL	VHDL
VHDL_ENCRYPT	Encrypted VHDL
VPI_LIBRARY	VPI Library

8.2.19. File Source Properties

Name	Description
PATH	Specifies the original source file and copies to `output_file`.
TEXT	Specifies an arbitrary text string for the contents of `output_file`.

8.2.20. Simulator Properties

Name	Description
ENV_ALDEC_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running riviera-pro
ENV_CADENCE_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running ncsim
ENV_MENTOR_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running modelsim
ENV_SYNOPSYS_LD_LIBRARY_PATH	LD_LIBRARY_PATH when running vcs
OPT_ALDEC_PLI	-pli option for riviera-pro
OPT_CADENCE_64BIT	-64bit option for ncsim
OPT_CADENCE_PLI	-loadpli1 option for ncsim
OPT_CADENCE_SVLIB	-sv_lib option for ncsim
OPT_CADENCE_SVROOT	-sv_root option for ncsim
OPT_MENTOR_64	-64 option for modelsim
OPT_MENTOR_CPPPATH	-cpppath option for modelsim
OPT_MENTOR_LDFLAGS	-ldflags option for modelsim
OPT_MENTOR_PLI	-pli option for modelsim
OPT_SYNOPSYS_ACC	+acc option for vcs
OPT_SYNOPSYS_CPP	-cpp option for vcs
OPT_SYNOPSYS_FULL64	-full64 option for vcs
OPT_SYNOPSYS_LDFLAGS	-LDFLAGS option for vcs
OPT_SYNOPSYS_LLIB	-l option for vcs
OPT_SYNOPSYS_VPI	+vpi option for vcs

8.2.21. Port VHDL Type Properties

Name	Description
AUTO	The VHDL type of this signal is automatically determined. Single-bit signals are STD_LOGIC; signals wider than one bit are STD_LOGIC_VECTOR.
STD_LOGIC	Indicates that the signal is not rendered in VHDL as a STD_LOGIC signal.
STD_LOGIC_VECTOR	Indicates that the signal is rendered in VHDL as a STD_LOGIC_VECTOR signal.

8.2.22. System Info Type Properties

Type	Name	Description
String	ADDRESS_MAP	An XML-formatted string describing the address map for the interface specified in the system info argument.
Integer	ADDRESS_WIDTH	The number of address bits required to address all memory-mapped agents connected to the specified memory-mapped host in this instance, using byte addresses.
String	AVALON_SPEC	The version of the interconnect. SOPC Builder interconnect uses Avalon Specification 1.0. Platform Designer interconnect uses Avalon Specification 2.0.
Integer	CLOCK_DOMAIN	An integer that represents the clock domain for the interface specified in the system info argument. If this instance has interfaces on multiple clock domains, this can be used to determine which interfaces are on each clock domain. The absolute value of the integer is arbitrary.
Long, Integer	CLOCK_RATE	The rate of the clock connected to the clock input specified in the system info argument. If 0, the clock rate is currently unknown.
String	CLOCK_RESET_INFO	The name of this instance's primary clock or reset sink interface. This is used to determine the reset sink to use for global reset when using SOPC interconnect.
String	CUSTOM_INSTRUCTION_SLAVES	Provides custom instruction agent information, including the name, base address, address span, and clock cycle type.
(various)	DESIGN_ENVIRONMENT	A string that identifies the current design environment. Refer to *Design Environment Type Properties*.
String	DEVICE	The device part number of the currently selected device.
String	DEVICE_FAMILY	The family name of the currently selected device.
String	DEVICE_FEATURES	A list of key/value pairs delineated by spaces indicating whether a particular device feature is available in the currently selected device family. The format of the list is suitable for passing to the Tcl `array` set command. The keys are device features; the values are 1 if the feature is present, and 0 if the feature is absent.
String	DEVICE_SPEEDGRADE	The speed grade of the currently selected device.
Integer	GENERATION_ID	A integer that stores a hash of the generation time to be used as a unique ID for a generation run.
BigInteger, Long	INTERRUPTS_USED	A mask indicating which bits of an interrupt receiver are connected to interrupt senders. The interrupt receiver is specified in the system info argument.
Integer	MAX_SLAVE_DATA_WIDTH	The data width of the widest agent connected to the specified memory-mapped host.
String, Boolean, Integer	QUARTUS_INI	The value of the quartus.ini setting specified in the system info argument.
Integer	RESET_DOMAIN	An integer that represents the reset domain for the interface specified in the system info argument. If this instance has interfaces on multiple reset domains, this can be used to determine which interfaces are on each reset domain. The absolute value of the integer is arbitrary.

Type	Name	Description
String	TRISTATECONDUIT_INFO	An XML description of the Avalon Tri-state Conduit hosts connected to an Avalon Tri-state Conduit agent. The agent is specified as the system info argument. The value contains information about the agent, the connected host instance and interface names, and signal names, directions and widths.
String	TRISTATECONDUIT_MASTERS	The names of the instance's interfaces that are tri-state conduit agents.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

Related Information

Design Environment Type Properties on page 790

8.2.23. Design Environment Type Properties

Description

A design environment is used by IP to tell what sort of interfaces are most appropriate to connect in the parent system.

Name	Description
NATIVE	Design environment prefers native IP interfaces.
QSYS	Design environment prefers standard Platform Designer interfaces.

8.2.24. Units Properties

Name	Description
`Address`	A memory-mapped address.
`Bits`	Memory size, in bits.
`BitsPerSecond`	Rate, in bits per second.
`Bytes`	Memory size, in bytes.
`Cycles`	A latency or count, in clock cycles.
`GigabitsPerSecond`	Rate, in gigabits per second.
`Gigabytes`	Memory size, in gigabytes.
`Gigahertz`	Frequency, in GHz.
`Hertz`	Frequency, in Hz.
`KilobitsPerSecond`	Rate, in kilobits per second.
`Kilobytes`	Memory size, in kilobytes.
`Kilohertz`	Frequency, in kHz.
`MegabitsPerSecond`	Rate, in megabits per second.
`Megabytes`	Memory size, in megabytes.
`Megahertz`	Frequency, in MHz.
`Microseconds`	Time, in micros.
`Milliseconds`	Time, in ms.
`Nanoseconds`	Time, in ns.
`None`	Unspecified units.
`Percent`	A percentage.
`Picoseconds`	Time, in ps.
`Seconds`	Time, in s.

8.2.25. Operating System Properties

Name	Description
ALL	All operating systems
LINUX32	Linux 32-bit
LINUX64	Linux 64-bit
WINDOWS32	Windows 32-bit
WINDOWS64	Windows 64-bit

8.2.26. Quartus.ini Type Properties

Name	Description
ENABLED	Returns `1` if the setting is turned on, otherwise returns `0`.
STRING	Returns the string value of the `.ini` setting.

8.3. Component Interface Tcl Reference Revision History

The table below indicates edits made to the *Component Interface Tcl Reference* content since its creation:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout. • Added missing file option to *Parameter Properties* topic.
2023.04.03	23.1	• The product family name is updated to "Intel Agilex 7" to reflect the different family members.
2022.09.26	22.3	• Updated *Port Properties* and *Module Properties* topics to remove instances of obsolete `GENERATION_CALLBACK` port and module properties.
2022.06.20	22.2	• Revised `add_instance` topic to indicate that the `version` argument is required.
2021.03.29	21.1	• Converted to "host" and "agent" inclusive terminology for Avalon memory mapped interface descriptions and related GUI elements throughout.
2019.04.01	19.1.0	• Described new domain and post adaptation assignments in "Interconnect Parameters" topic.
2018.09.24	18.1.0	• Added new _hw.tcl interface properties that allow importing and exporting register maps in IP-XACT format.
2018.05.07	18.0.0	• Added wire-level expression commands to support wire-level interfaces. • Updated `send_message` level availability for `INFO` messages. • Updated `set_port_property` availability.
2017.11.06	17.1.0	• Changed instances of *Qsys Pro* to *Platform Designer* • Added statement clarifying use of brackets. • Added properties and interface commands to support SystemVerilog.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Implemented *Qsys* rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Edit to `add_fileset_file` command.
December 2014	14.1.0	• set_interface_upgrade_map • Moved **Port Roles (Interface Signal Types)** section to *Qsys Interconnect*.
November 2013	13.1.0	• add_hdl_instance
May 2013	13.0.0	• Consolidated content from other Qsys chapters. • Added AMBA APB support.
November 2012	12.1.0	• Added the **demo_axi_memory** example with screen shots and example **_hw.tcl** code.
June 2012	12.0.0	• Added AXI 3 support. • Added: `set_display_item_property`, `set_parameter_property`,`LONG_DESCRIPTION`, and static filesets.

continued...

Document Version	Quartus Prime Version	Changes
November 2011	11.1.0	• Template update. • Added: `set_qip_strings`, `get_qip_strings`, `get_device_family_displayname`, `check_device_family_equivalence`.
May 2011	11.0.0	• Revised section describing HDL and composed component implementations. • Separated reset and clock interfaces in example. • Added: `TRISTATECONDUIT_INFO`, `GENERATION_ID`, `UNIQUE_ID` `SYSTEM_INFO`. • Added: `WIDTH` and `SYSTEM_INFO_ARG` parameter properties. • Removed the `doc_type` argument from the `add_documentation_link` command. • Removed: `get_instance_parameter_properties` • Added: `add_fileset`, `add_fileset_file`, `create_temp_file`. • Updated Tcl examples to show separate clock and reset interfaces.
December 2010	10.1.0	• Initial release.

9. Quartus Prime Pro Edition User Guide: Platform Designer Document Archives

For the latest and previous versions of this user guide, refer to Quartus Prime Pro Edition User Guide: Platform Designer. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Intel® Quartus® Prime Pro Edition User Guide

Design Recommendations

Updated for Intel® Quartus® Prime Design Suite: **23.1**

This document is part of a collection - Intel® Quartus® Prime Pro Edition User Guides - Combined PDF link

Online Version
Send Feedback

683082
UG-20131
2023.08.03

Contents

1. Recommended HDL Coding Styles

This chapter provides Hardware Description Language (HDL) coding style recommendations to ensure optimal synthesis results when targeting Intel FPGA devices.

HDL coding styles have a significant effect on the quality of results for programmable logic designs. Synthesis tools optimize HDL code for both logic utilization and performance; however, synthesis tools cannot interpret the intent of your design. Therefore, the most effective optimizations require conformance to recommended coding styles.

Note: For style recommendations, options, or HDL attributes specific to your synthesis tool, refer to the synthesis tool vendor's documentation.

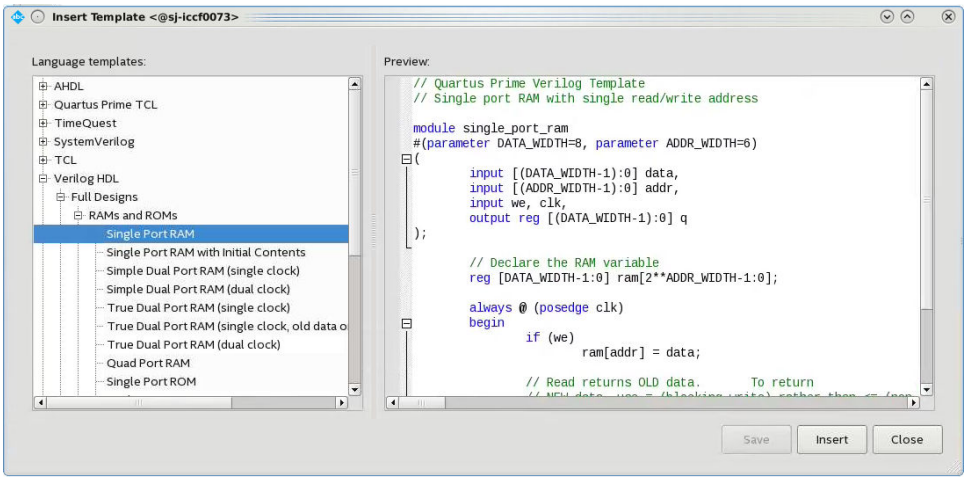
1.1. Using Provided HDL Templates

The Intel® Quartus® Prime software provides templates for Verilog HDL, SystemVerilog, and VHDL templates to start your HDL designs. Many of the HDL examples in this document correspond with the **Full Designs** examples in the **Intel Quartus Prime Templates**. You can insert HDL code into your own design using the templates or examples.

1.1.1. Inserting HDL Code from a Provided Template

1. Click **File ➤ New**.
2. In the **New** dialog box, select the HDL language for the design files: **SystemVerilog HDL File**, **VHDL File**, or **Verilog HDL File**; and click **OK**. A text editor tab with a blank file opens.
3. Right-click the blank file and click **Insert Template**.
4. In the **Insert Template** dialog box, expand the section corresponding to the appropriate HDL, then expand the **Full Designs** section.
5. Select a template. The template now appears in the **Preview** pane.
6. To paste the HDL design into the blank Verilog or VHDL file you created, click **Insert**.
7. Click **Close** to close the **Insert Template** dialog box.

Figure 1. Inserting a RAM Template



Note: Use the Intel Quartus Prime Text Editor to modify the HDL design or save the template as an HDL file to edit in your preferred text editor.

1.2. Instantiating IP Cores in HDL

Intel provides parameterizable IP cores that are optimized for Intel FPGA device architectures. Using IP cores instead of coding your own logic saves valuable design time.

Additionally, the Intel-provided IP cores offer more efficient logic synthesis and device implementation. Scale the IP core's size and specify various options by setting parameters. To instantiate the IP core directly in your HDL file code, invoke the IP core name and define its parameters as you would do for any other module, component, or sub design. Alternatively, you can use the IP Catalog (**Tools ➤ IP Catalog**) and parameter editor GUI to simplify customization of your IP core variation. You can infer or instantiate IP cores that optimize device architecture features, for example:

- Transceivers
- LVDS drivers
- Memory and DSP blocks
- Phase-locked loops (PLLs)
- Double-data rate input/output (DDIO) circuitry

For some types of logic functions, such as memories and DSP functions, you can infer device-specific dedicated architecture blocks instead of instantiating an IP core. Intel Quartus Prime synthesis recognizes certain HDL code structures and automatically infers the appropriate IP core or map directly to device atoms.

1.3. Inferring Multipliers and DSP Functions

The following sections describe how to infer multiplier and DSP functions from generic HDL code, and, if applicable, how to target the dedicated DSP block architecture in Intel FPGA devices.

Related Information

Intel FPGA Digital Signal Processing

1.3.1. Inferring Multipliers

To infer multiplier functions, synthesis tools detect multiplier logic and implement this in Intel FPGA IP cores, or map the logic directly to device atoms.

For devices with DSP blocks, Intel Quartus Prime synthesis can implement the function in a DSP block instead of logic, depending on device utilization. The Intel Quartus Prime fitter can also place input and output registers in DSP blocks (that is, perform register packing) to improve performance and area utilization.

The following Verilog HDL and VHDL code examples show that synthesis tools can infer signed and unsigned multipliers as IP cores or DSP block atoms. Each example fits into one DSP block element. In addition, when register packing occurs, no extra logic cells for registers are required.

Example 1. Verilog HDL Unsigned Multiplier

```
module unsigned_mult (out, a, b);
    output [15:0] out;
    input [7:0] a;
    input [7:0] b;
    assign out = a * b;
endmodule
```

Note: The `signed` declaration in Verilog HDL is a feature of the Verilog 2001 Standard.

Example 2. Verilog HDL Signed Multiplier with Input and Output Registers (Pipelining = 2)

```
module signed_mult (out, clk, a, b);
   output [15:0] out;
   input clk;
   input signed [7:0] a;
   input signed [7:0] b;

   reg signed [7:0] a_reg;
   reg signed [7:0] b_reg;
   reg signed [15:0] out;
   wire signed [15:0] mult_out;

   assign mult_out = a_reg * b_reg;

   always @ (posedge clk)
   begin
      a_reg <= a;
      b_reg <= b;
      out <= mult_out;
   end
endmodule
```

Example 3. VHDL Unsigned Multiplier with Input and Output Registers (Pipelining = 2)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY unsigned_mult IS
   PORT (
      a: IN UNSIGNED (7 DOWNTO 0);
```

```
      b: IN UNSIGNED (7 DOWNTO 0);
      clk: IN STD_LOGIC;
      aclr: IN STD_LOGIC;
      result: OUT UNSIGNED (15 DOWNTO 0)
   );
END unsigned_mult;

ARCHITECTURE rtl OF unsigned_mult IS
   SIGNAL a_reg, b_reg: UNSIGNED (7 DOWNTO 0);
BEGIN
   PROCESS (clk, aclr)
   BEGIN
      IF (aclr ='1') THEN
         a_reg <= (OTHERS => '0');
         b_reg <= (OTHERS => '0');
         result <= (OTHERS => '0');
      ELSIF (rising_edge(clk)) THEN
         a_reg <= a;
         b_reg <= b;
         result <= a_reg * b_reg;
      END IF;
   END PROCESS;
END rtl;
```

Example 4. VHDL Signed Multiplier

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY signed_mult IS
   PORT (
      a: IN SIGNED (7 DOWNTO 0);
      b: IN SIGNED (7 DOWNTO 0);
      result: OUT SIGNED (15 DOWNTO 0)
   );
END signed_mult;

ARCHITECTURE rtl OF signed_mult IS
BEGIN
   result <= a * b;
END rtl;
```

1.3.2. Inferring Multiply-Accumulator and Multiply-Adder Functions

Synthesis tools detect multiply-accumulator or multiply-adder functions, and either implement them as Intel FPGA IP cores or map them directly to device atoms. During placement and routing, the Intel Quartus Prime software places multiply-accumulator and multiply-adder functions in DSP blocks.

Note: Synthesis tools infer multiply-accumulator and multiply-adder functions only if the Intel device family has dedicated DSP blocks that support these functions.

A simple multiply-accumulator consists of a multiplier feeding an addition operator. The addition operator feeds a set of registers that then feeds the second input to the addition operator. A simple multiply-adder consists of two to four multipliers feeding one or two levels of addition, subtraction, or addition/subtraction operators. Addition is always the second-level operator, if it is used. In addition to the multiply-accumulator and multiply-adder, the Intel Quartus Prime Fitter also places input and output registers into the DSP blocks to pack registers and improve performance and area utilization.

Some device families offer additional advanced multiply-adder and accumulator functions, such as complex multiplication, input shift register, or larger multiplications.

The Verilog HDL and VHDL code samples infer multiply-accumulator and multiply-adder functions with input, output, and pipeline registers, as well as an optional asynchronous `clear` signal. Using the three sets of registers provides the best performance through the function, with a latency of three. To reduce latency, remove the registers in your design.

Note: To obtain high performance in DSP designs, use register pipelining and avoid unregistered DSP functions.

Example 5. Verilog HDL Multiply-Accumulator

```
module sum_of_four_multiply_accumulate
   #(parameter INPUT_WIDTH=18, parameter OUTPUT_WIDTH=44)
   (
      input clk, ena,
      input [INPUT_WIDTH-1:0] dataa, datab, datac, datad,
      input [INPUT_WIDTH-1:0] datae, dataf, datag, datah,
      output reg [OUTPUT_WIDTH-1:0] dataout
   );
   // Each product can be up to 2*INPUT_WIDTH bits wide.
   // The sum of four of these products can be up to 2 bits wider.
   wire [2*INPUT_WIDTH+1:0] mult_sum;

   // Store the results of the operations on the current inputs
   assign mult_sum = (dataa * datab + datac * datad) +
                     (datae * dataf + datag * datah);

   // Store the value of the accumulation
   always @ (posedge clk)
   begin
      if (ena == 1)
         begin
            dataout <= dataout + mult_sum;
         end
   end
endmodule
```

1.4. Inferring Memory Functions from HDL Code

The following coding recommendations provide portable examples of generic HDL code targeting dedicated Intel FPGA memory IP cores. However, if you want to use some of the advanced memory features in Intel FPGA devices, consider using the IP core directly so that you can customize the ports and parameters easily.

You can also use the Intel Quartus Prime templates provided in the Intel Quartus Prime software as a starting point.

Table 1. Intel Memory HDL Language Templates

Language	Full Design Name
VHDL	Single-Port RAM Single-Port RAM with Initial Contents Simple Dual-Port RAM (single clock) Simple Dual-Port RAM (dual clock) True Dual-Port RAM (single clock) True Dual-Port RAM (dual clock)

continued...

Language	Full Design Name
	Mixed-Width RAM Mixed-Width True Dual-Port RAM Byte-Enabled Simple Dual-Port RAM Byte-Enabled True Dual-Port RAM Single-Port ROM Dual-Port ROM
Verilog HDL	Single-Port RAM Single-Port RAM with Initial Contents Simple Dual-Port RAM (single clock) Simple Dual-Port RAM (dual clock) True Dual-Port RAM (single clock) True Dual-Port RAM (dual clock) Single-Port ROM Dual-Port ROM
SystemVerilog	Mixed-Width Port RAM Mixed-Width True Dual-Port RAM Mixed-Width True Dual-Port RAM (new data on same port read during write) Byte-Enabled Simple Dual Port RAM Byte-Enabled True Dual-Port RAM

Related Information

- Introduction to Intel® FPGA IP Cores
- Intel® Hyperflex™ Architecture High-Performance Design Handbook
- Intel® Arria® 10 Core Fabric and General Purpose I/Os Handbook

1.4.1. Inferring RAM functions from HDL Code

To infer RAM functions, synthesis tools recognize certain types of HDL code and map the detected code to technology-specific implementations. For device families that have dedicated RAM blocks, the Intel Quartus Prime software uses an Intel FPGA IP core to target the device memory architecture.

Synthesis tools typically consider all signals and variables that have a multi-dimensional array type and then create a RAM block, if applicable. This is based on the way the signals or variables are assigned or referenced in the HDL source description.

Standard synthesis tools recognize single-port and simple dual-port (one read port and one write port) RAM blocks. Some synthesis tools (such as the Intel Quartus Prime software) also recognize true dual-port (two read ports and two write ports) RAM blocks that map to the memory blocks in certain Intel FPGA devices.

Some tools (such as the Intel Quartus Prime software) also infer memory blocks for array variables and signals that are referenced (read/written) by two indexes, to recognize mixed-width and byte-enabled RAMs for certain coding styles.

Note: If your design contains a RAM block that your synthesis tool does not recognize and infer, the design might require a large amount of system memory that can potentially cause compilation problems.

1.4.1.1. Use Synchronous Memory Blocks

Memory blocks in Intel FPGA are synchronous. Therefore, RAM designs must be synchronous to map directly into dedicated memory blocks. For these devices, Intel Quartus Prime synthesis implements asynchronous memory logic in regular logic cells.

Synchronous memory offers several advantages over asynchronous memory, including higher frequencies and thus higher memory bandwidth, increased reliability, and less standby power. To convert asynchronous memory, move registers from the datapath into the memory block.

A memory block is synchronous if it has one of the following read behaviors:

- Memory read occurs in a Verilog HDL `always` block with a `clock` signal or a VHDL clocked process. The recommended coding style for synchronous memories is to create your design with a registered read output.
- Memory read occurs outside a clocked block, but there is a synchronous read address (that is, the address used in the read statement is registered). Synthesis does not always infer this logic as a memory block, or may require external bypass logic, depending on the target device architecture. Avoid this coding style for synchronous memories.

Note: The synchronous memory structures in Intel FPGA devices can differ from the structures in other vendors' devices. For best results, match your design to the target device architecture.

This chapter provides coding recommendations for various memory types. All the examples in this document are synchronous to ensure that they can be directly mapped into the dedicated memory architecture available in Intel FPGAs.

1.4.1.2. Avoid Unsupported Reset and Control Conditions

To ensure correct implementation of HDL code in the target device architecture, avoid unsupported reset conditions or other control logic that does not exist in the device architecture.

The RAM contents of Intel FPGA memory blocks cannot be cleared with a `reset` signal during device operation. If your HDL code describes a RAM with a `reset` signal for the RAM contents, the logic is implemented in regular logic cells instead of a memory block. Do not place RAM read or write operations in an `always` block or `process` block with a `reset` signal. To specify memory contents, initialize the memory or write the data to the RAM during device operation.

In addition to reset signals, other control logic can prevent synthesis from inferring memory logic as a memory block. For example, if you use a clock enable on the read address registers, you can alter the output latch of the RAM, resulting in the synthesized RAM result not matching the HDL description. Use the address stall feature as a read address clock enable to avoid this limitation. Check the documentation for your FPGA device to ensure that your code matches the hardware available in the device.

Example 6. Verilog RAM with Reset Signal that Clears RAM Contents: Not Supported in Device Architecture

```
module clear_ram
(
    input clock, reset, we,
    input [7:0] data_in,
    input [4:0] address,
    output reg [7:0] data_out
);

    reg [7:0] mem [0:31];
    integer i;

    always @ (posedge clock or posedge reset)
    begin
        if (reset == 1'b1)
            mem[address] <= 0;
        else if (we == 1'b1)
            mem[address] <= data_in;

        data_out <= mem[address];
    end
endmodule
```

Related Information

Specifying Initial Memory Contents at Power-Up on page 24

1.4.1.3. Check Read-During-Write Behavior

Ensure the read-during-write behavior of the memory block described in your HDL design is consistent with your target device architecture.

Your HDL source code specifies the memory behavior when you read and write from the same memory address in the same clock cycle. The read returns either the old data at the address, or the new data written to the address. This is referred to as the read-during-write behavior of the memory block. Intel FPGA memory blocks have different read-during-write behavior depending on the target device family, memory mode, and block type.

Synthesis tools preserve the functionality described in your source code. Therefore, if your source code specifies unsupported read-during-write behavior for the RAM blocks, the Intel Quartus Prime software implements the logic in regular logic cells as opposed to the dedicated RAM hardware.

Example 7. Continuous read in HDL code

One common problem occurs when there is a continuous read in the HDL code, as in the following examples. Avoid using these coding styles:

```
//Verilog HDL concurrent signal assignment
assign q = ram[raddr_reg];
```

```
-- VHDL concurrent signal assignment
q <= ram(raddr_reg);
```

This type of HDL implies that when a write operation takes place, the read immediately reflects the new data at the address independent of the read clock, which is the behavior of asynchronous memory blocks. Synthesis cannot directly map this behavior to a synchronous memory block. If the write clock and read clock are the

same, synthesis can infer memory blocks and add extra bypass logic so that the device behavior matches the HDL behavior. If the write and read clocks are different, synthesis cannot reliably add bypass logic, so it implements the logic in regular logic cells instead of dedicated RAM blocks. The examples in the following sections discuss some of these differences for read-during-write conditions.

In addition, the MLAB memories in certain device logic array blocks (LABs) does not easily support old data or new data behavior for a read-during-write in the dedicated device architecture. Implementing the extra logic to support this behavior significantly reduces timing performance through the memory.

Note: For best performance in MLAB memories, ensure that your design does not depend on the read data during a write operation.

In many synthesis tools, you can declare that the read-during-write behavior is not important to your design (for example, if you never read from the same address to which you write in the same clock cycle). In Intel Quartus Prime Pro Edition synthesis, set the synthesis attribute `ramstyle` to `no_rw_check` to allow Intel Quartus Prime software to define the read-during-write behavior of a RAM, rather than use the behavior specified by your HDL code. This attribute can prevent the synthesis tool from using extra logic to implement the memory block, or can allow memory inference when it would otherwise be impossible.

1.4.1.4. Controlling RAM Inference and Implementation

Intel Quartus Prime synthesis provides options to control RAM inference and implementation for Intel FPGA devices with synchronous memory blocks. Synthesis tools usually do not infer small RAM blocks because implementing small RAM blocks is more efficient if using the registers in regular logic.

To direct the Intel Quartus Prime software to infer RAM blocks globally for all sizes, enable the **Allow Any RAM Size for Recognition** option in the **Advanced Analysis & Synthesis Settings** dialog box (**Assignments ➤ Settings ➤ Compiler Settings ➤ Synthesis Settings (Advanced)**).

Alternatively, use the `ramstyle` RTL attribute to specify how an inferred RAM is implemented, including the type of memory block or the use of regular logic instead of a dedicated memory block. Intel Quartus Prime synthesis does not map inferred memory into MLABs unless the HDL code specifies the appropriate `ramstyle` attribute, although the Fitter may map some memories to MLABs.

Set the `ramstyle` attribute in the RTL or in the `.qsf` file.

```
(* ramstyle = "mlab" *) my_shift_reg
```

```
set_instance_assignment -name RAMSTYLE_ATTRIBUTE LOGIC -to ram
```

. This attribute controls the implementation of an inferred memory. Apply the attribute to a variable declaration that infers a RAM, ROM, or shift-register. Legal values are: `"M9K"`, `"M10K"`, `"M20K"`, `"M144K"`, `"MLAB`, `"no_rw_check"`, `"logic"`

You can also specify the maximum depth of memory blocks for RAM or ROM inference in RTL. Specify the `max_depth` synthesis attribute to the declaration of a variable that represents a RAM or ROM in your design file. For example:

```
// Limit the depth of the memory blocks implement "ram" to 512
// This forces the Intel Quartus Prime software to use two M512 blocks instead
of one M4K block to implement this RAM
(* max_depth = 512 *) reg [7:0] ram[0:1023];
```

In addition, you can specify the `no_ram` synthesis attribute to prevent RAM inference on a specific array. For example:

```
 (* no_ram *) logic [11:0] my_array [0:12];
```

1.4.1.5. Single-Clock Synchronous RAM with Old Data Read-During-Write Behavior

The code examples in this section show Verilog HDL and VHDL code that infers simple dual-port, single-clock synchronous RAM. Single-port RAM blocks use a similar coding style.

The read-during-write behavior in these examples is to read the old data at the memory address. For best performance in MLAB memories, use the appropriate attribute so that your design does not depend on the read data during a write operation. The simple dual-port RAM code samples map directly into Intel synchronous memory.

Single-port versions of memory blocks (that is, using the same read address and write address signals) allow better RAM utilization than dual-port memory blocks, depending on the device family. Refer to the appropriate device handbook for recommendations on your target device.

Example 8. Verilog HDL Single-Clock, Simple Dual-Port Synchronous RAM with Old Data Read-During-Write Behavior

```
module single_clk_ram(
    output reg [7:0] q,
    input [7:0] d,
    input [4:0] write_address, read_address,
    input we, clk
);
    reg [7:0] mem [31:0];

    always @ (posedge clk) begin
        if (we)
            mem[write_address] <= d;
        q <= mem[read_address]; // q doesn't get d in this clock cycle
    end
endmodule
```

Example 9. VHDL Single-Clock, Simple Dual-Port Synchronous RAM with Old Data Read-During-Write Behavior

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY single_clock_ram IS
    PORT (
        clock: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 to 31;
        read_address: IN INTEGER RANGE 0 to 31;
```

```
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END single_clock_ram;

ARCHITECTURE rtl OF single_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL ram_block: MEM;
BEGIN
    PROCESS (clock)
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
            q <= ram_block(read_address);
            -- VHDL semantics imply that q doesn't get data
            -- in this clock cycle
        END IF;
    END PROCESS;
END rtl;
```

Note: The small size of this `single_clock_ram` causes the Compiler to infer the memory as MLAB memory blocks, rather than M20K memory blocks. If `single_clock_ram` specifies a larger width, the Compiler infers the memory as M20K memory blocks.

1.4.1.6. Single-Clock Synchronous RAM with New Data Read-During-Write Behavior

The examples in this section describe RAM blocks in which the read-during-write behavior returns the new value being written at the memory address.

To implement this behavior in the target device, synthesis tools add bypass logic around the RAM block. This bypass logic increases the area utilization of the design, and decreases the performance if the RAM block is part of the design's critical path. If the device memory supports new data read-during-write behavior when in single-port mode (same clock, same read address, and same write address), the Verilog memory block doesn't require any bypass logic. Refer to the appropriate device handbook for specifications on your target device.

The following examples use a blocking assignment for the write so that the data is assigned intermediately.

Example 10. Verilog HDL Single-Clock, Simple Dual-Port Synchronous RAM with New Data Read-During-Write Behavior

```
module single_clock_wr_ram(
    output reg [7:0] q,
    input [7:0] d,
    input [6:0] write_address, read_address,
    input we, clk
);
    reg [7:0] mem [127:0];

    always @ (posedge clk) begin
        if (we)
            mem[write_address] = d;
        q = mem[read_address]; // q does get d in this clock
                               // cycle if we is high
    end
endmodule
```

Example 11. VHDL Single-Clock, Simple Dual-Port Synchronous RAM with New Data Read-During-Write Behavior:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY single_clock_ram IS
    PORT (
        clock: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 to 31;
        read_address: IN INTEGER RANGE 0 to 31;
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
    );
END single_clock_ram;

ARCHITECTURE rtl OF single_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN
    PROCESS (clock)
    VARIABLE ram_block: MEM;
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) := data;
            END IF;
            q <= ram_block(read_address);
            -- VHDL semantics imply that q doesn't get data
            -- in this clock cycle
        END IF;
    END PROCESS;
END rtl;
```

It is possible to create a single-clock RAM by using an `assign` statement to read the address of `mem` and create the output `q`. By itself, the RTL describes new data read-during-write behavior. However, if the RAM output feeds a register in another hierarchy, a read-during-write results in the old data. Synthesis tools may not infer a RAM block if the tool cannot determine which behavior is described, such as when the memory feeds a hard hierarchical partition boundary. Avoid this type of RTL.

Example 12. Avoid Verilog Coding Style with Vague read-during-write Behavior

```
reg [7:0] mem [127:0];
reg [6:0] read_address_reg;

always @ (posedge clk) begin
    if (we)
        mem[write_address] <= d;
    read_address_reg <= read_address;
end
assign q = mem[read_address_reg];
```

Example 13. Avoid VHDL Coding Style with Vague read-during-write Behavior

The following example uses a concurrent signal assignment to read from the RAM, and presents a similar behavior.

```
ARCHITECTURE rtl OF single_clock_rw_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL ram_block: MEM;
    SIGNAL read_address_reg: INTEGER RANGE 0 to 31;
BEGIN
    PROCESS (clock)
    BEGIN
```

```
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
            read_address_reg <= read_address;
        END IF;
    END PROCESS;
    q <= ram_block(read_address_reg);
END rtl;
```

1.4.1.7. Simple Dual-Port, Dual-Clock Synchronous RAM

With dual-clock designs, synthesis tools cannot accurately infer the read-during-write behavior because it depends on the timing of the two clocks within the target device. Therefore, the read-during-write behavior of the synthesized design is undefined and may differ from your original HDL code.

Example 14. Verilog HDL Simple Dual-Port, Dual-Clock Synchronous RAM

```
module simple_dual_port_ram_dual_clock
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=6)
(
    input [(DATA_WIDTH-1):0] data,
    input [(ADDR_WIDTH-1):0] read_addr, write_addr,
    input we, read_clock, write_clock,
    output reg [(DATA_WIDTH-1):0] q
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    always @ (posedge write_clock)
    begin
        // Write
        if (we)
            ram[write_addr] <= data;
    end

    always @ (posedge read_clock)
    begin
        // Read
        q <= ram[read_addr];
    end

endmodule
```

Example 15. VHDL Simple Dual-Port, Dual-Clock Synchronous RAM

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dual_clock_ram IS
    PORT (
        clock1, clock2: IN STD_LOGIC;
        data: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
        write_address: IN INTEGER RANGE 0 to 31;
        read_address: IN INTEGER RANGE 0 to 31;
        we: IN STD_LOGIC;
        q: OUT STD_LOGIC_VECTOR (3 DOWNTO 0)
    );
END dual_clock_ram;
ARCHITECTURE rtl OF dual_clock_ram IS
    TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL ram_block: MEM;
    SIGNAL read_address_reg : INTEGER RANGE 0 to 31;
BEGIN
```

```
    PROCESS (clock1)
    BEGIN
        IF (rising_edge(clock1)) THEN
            IF (we = '1') THEN
                ram_block(write_address) <= data;
            END IF;
        END IF;
    END PROCESS;
    PROCESS (clock2)
    BEGIN
        IF (rising_edge(clock2)) THEN
            q <= ram_block(read_address_reg);
            read_address_reg <= read_address;
        END IF;
    END PROCESS;
END rtl;
```

Related Information

Check Read-During-Write Behavior on page 11

1.4.1.8. True Dual-Port Synchronous RAM

The code examples in this section show Verilog HDL and VHDL code that infers true dual-port synchronous RAM. Different synthesis tools may differ in their support for these types of memories.

Intel FPGA synchronous memory blocks have two independent address ports, allowing for operations on two unique addresses simultaneously. A read operation and a write operation can share the same port if they share the same address.

The Intel Quartus Prime software infers true dual-port RAMs in Verilog HDL and VHDL, with the following characteristics:

- Any combination of independent read or write operations in the same clock cycle.
- At most two unique port addresses.
- In one clock cycle, with one or two unique addresses, they can perform:
  - Two reads and one write
  - Two writes and one read
  - Two writes and two reads

In the synchronous RAM block architecture, there is no priority between the two ports. Therefore, if you write to the same location on both ports at the same time, the result is indeterminate in the device architecture. You must ensure your HDL code does not imply priority for writes to the memory block, if you want the design to be implemented in a dedicated hardware memory block. For example, if both ports are defined in the same process block, the code is synthesized and simulated sequentially so that there is a priority between the two ports. If your code does imply a priority, the logic cannot be implemented in the device RAM blocks and is implemented in regular logic cells. You must also consider the read-during-write behavior of the RAM block to ensure that it can be mapped directly to the device RAM architecture.

When a read and write operation occurs on the same port for the same address, the read operation may behave as follows:

- **Read new data**—Intel Arria® 10 and Intel Stratix® 10 devices support this behavior.
- **Read old data**—Not supported.

When a read and write operation occurs on different ports for the same address (also known as mixed port), the read operation may behave as follows:

- **Read new data**—Intel Quartus Prime Pro Edition synthesis supports this mode by creating bypass logic around the synchronous memory block.
- **Read old data**—Intel Arria 10 and Intel Cyclone® 10 devices support this behavior.
- **Read don't care**—Synchronous memory blocks support this behavior in simple dual-port mode.

The Verilog HDL single-clock code sample maps directly into synchronous Intel Arria 10 memory blocks. When a read and write operation occurs on the same port for the same address, the new data being written to the memory is read. When a read and write operation occurs on different ports for the same address, the old data in the memory is read. Simultaneous writes to the same location on both ports results in indeterminate behavior.

If you generate a dual-clock version of this design describing the same behavior, the inferred memory in the target device presents undefined mixed port read-during-write behavior, because it depends on the relationship between the clocks.

Example 16. Verilog HDL True Dual-Port RAM with Single Clock

```
/ Quartus Prime Verilog Template
// True Dual Port RAM with single clock
//
// Read-during-write behavior is undefined for mixed ports
// and "new data" on the same port

module true_dual_port_ram_single_clock
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=6)
(
    input [(DATA_WIDTH-1):0] data_a, data_b,
    input [(ADDR_WIDTH-1):0] addr_a, addr_b,
    input we_a, we_b, clk,
    output reg [(DATA_WIDTH-1):0] q_a, q_b
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    // Port A
    always @ (posedge clk)
    begin
        if (we_a)
        begin
            ram[addr_a] = data_a;
        end
        q_a <= ram[addr_a];
    end

    // Port B
    always @ (posedge clk)
    begin
        if (we_b)
        begin
            ram[addr_b] = data_b;
        end
        q_b <= ram[addr_b];
    end

endmodule
```

Example 17. VHDL Read Statement Example

```
-- Port A
process(clk)
    begin
    if(rising_edge(clk)) then
        if(we_a = '1') then
            ram(addr_a) := data_a;
        end if;
        q_a <= ram(addr_a);
    end if;
end process;

-- Port B
process(clk)
    begin
    if(rising_edge(clk)) then
        if(we_b = '1') then
            ram(addr_b) := data_b;
        end if;
        q_b <= ram(addr_b);
    end if;
end process;
```

The VHDL single-clock code sample maps directly into Intel FPGA synchronous memory. When a read and write operation occurs on the same port for the same address, the new data writing to the memory is read. When a read and write operation occurs on different ports for the same address, the behavior results in old data for Intel Arria 10 and Intel Cyclone 10 devices, and is undefined for Intel Stratix 10 devices. Simultaneous write operations to the same location on both ports results in indeterminate behavior.

If you generate a dual-clock version of this design describing the same behavior, the memory in the target device presents undefined mixed port read-during-write behavior because it depends on the relationship between the clocks.

Example 18. VHDL True Dual-Port RAM with Single Clock

```
-- Quartus Prime VHDL Template
-- True Dual-Port RAM with single clock
--
-- Read-during-write behavior is undefined for mixed ports
-- and "new data" on the same port

library ieee;
use ieee.std_logic_1164.all;

entity true_dual_port_ram_single_clock is

    generic
    (
        DATA_WIDTH : natural := 8;
        ADDR_WIDTH : natural := 6
    );

    port
    (
        clk        : in std_logic;
        addr_a    : in natural range 0 to 2**ADDR_WIDTH - 1;
        addr_b    : in natural range 0 to 2**ADDR_WIDTH - 1;
        data_a    : in std_logic_vector((DATA_WIDTH-1) downto 0);
        data_b    : in std_logic_vector((DATA_WIDTH-1) downto 0);
        we_a    : in std_logic := '1';
        we_b    : in std_logic := '1';
        q_a        : out std_logic_vector((DATA_WIDTH -1) downto 0);
        q_b        : out std_logic_vector((DATA_WIDTH -1) downto 0)
    );
```

```
end true_dual_port_ram_single_clock;

architecture rtl of true_dual_port_ram_single_clock is

    -- Build a 2-D array type for the RAM
    subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
    type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

    -- Declare the RAM
    shared variable ram : memory_t;

begin

    -- Port A
    process(clk)
    begin
    if(rising_edge(clk)) then
        if(we_a = '1') then
            ram(addr_a) := data_a;
        end if;
        q_a <= ram(addr_a);
    end if;
    end process;

    -- Port B
    process(clk)
    begin
    if(rising_edge(clk)) then
        if(we_b = '1') then
            ram(addr_b) := data_b;
        end if;
          q_b <= ram(addr_b);
    end if;
    end process;

end rtl;
```

Related Information

Intel® Arria® 10 Core Fabric and General Purpose I/Os Handbook

1.4.1.9. Mixed-Width Dual-Port RAM

The RAM code examples in this section show SystemVerilog and VHDL code that infers RAM with data ports with different widths.

Verilog-1995 doesn't support mixed-width RAMs because the standard lacks a multi-dimensional array to model the different read width, write width, or both. Verilog-2001 doesn't support mixed-width RAMs because this type of logic requires multiple packed dimensions. Different synthesis tools may differ in their support for these memories. This section describes the inference rules for Intel Quartus Prime Pro Edition synthesis.

The first dimension of the multi-dimensional packed array represents the ratio of the wider port to the narrower port. The second dimension represents the narrower port width. The read and write port widths must specify a read or write ratio supported by the memory blocks in the target device. Otherwise, the synthesis tool does not infer a RAM.

Refer to the Intel Quartus Prime HDL templates for parameterized examples with supported combinations of read and write widths. You can also find examples of true dual port RAMs with two mixed-width read ports and two mixed-width write ports.

Example 19. SystemVerilog Mixed-Width RAM with Read Width Smaller than Write Width

```
module mixed_width_ram    // 256x32 write and 1024x8 read
(
        input [7:0] waddr,
        input [31:0] wdata,
        input we, clk,
        input [9:0] raddr,
        output logic [7:0] q
);
    logic [3:0][7:0] ram[0:255];
    always_ff@(posedge clk)
        begin
            if(we) ram[waddr] <= wdata;
            q <= ram[raddr / 4][raddr % 4];
        end
endmodule : mixed_width_ram
```

Example 20. SystemVerilog Mixed-Width RAM with Read Width Larger than Write Width

```
module mixed_width_ram     // 1024x8 write and 256x32 read
(
        input [9:0] waddr,
        input [31:0] wdata,
        input we, clk,
        input [7:0] raddr,
        output logic [9:0] q
);
    logic [3:0][7:0] ram[0:255];
    always_ff@(posedge clk)
         begin
            if(we) ram[waddr / 4][waddr % 4] <= wdata;
            q <= ram[raddr];
         end
endmodule : mixed_width_ram
```

Example 21. VHDL Mixed-Width RAM with Read Width Smaller than Write Width

```
library ieee;
use ieee.std_logic_1164.all;

package ram_types is
    type word_t is array (0 to 3) of std_logic_vector(7 downto 0);
    type ram_t is array (0 to 255) of word_t;
end ram_types;

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.ram_types.all;

entity mixed_width_ram is
    port (
        we, clk : in  std_logic;
        waddr   : in  integer range 0 to 255;
        wdata   : in  word_t;
        raddr   : in  integer range 0 to 1023;
        q       : out std_logic_vector(7 downto 0));
end mixed_width_ram;

architecture rtl of mixed_width_ram is
    signal ram : ram_t;
begin  -- rtl
    process(clk, we)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
```

```
                ram(waddr) <= wdata;
            end if;
            q <= ram(raddr / 4 )(raddr mod 4);
        end if;
    end process;
end rtl;
```

Example 22. VHDL Mixed-Width RAM with Read Width Larger than Write Width

```
library ieee;
use ieee.std_logic_1164.all;

package ram_types is
    type word_t is array (0 to 3) of std_logic_vector(7 downto 0);
    type ram_t is array (0 to 255) of word_t;
end ram_types;

library ieee;
use ieee.std_logic_1164.all;
library work;
use work.ram_types.all;

entity mixed_width_ram is
    port (
        we, clk : in  std_logic;
        waddr   : in  integer range 0 to 1023;
        wdata   : in  std_logic_vector(7 downto 0);
        raddr   : in  integer range 0 to 255;
        q       : out word_t);
end mixed_width_ram;

architecture rtl of mixed_width_ram is
    signal ram : ram_t;
begin  -- rtl
    process(clk, we)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                ram(waddr / 4)(waddr mod 4) <= wdata;
            end if;
            q <= ram(raddr);
        end if;
    end process;
end rtl;
```

1.4.1.10. RAM with Byte-Enable Signals

The RAM code examples in this section show SystemVerilog and VHDL code that infers RAM with controls for writing single bytes into the memory word, or byte-enable signals.

Synthesis models byte-enable signals by creating write expressions with two indexes, and writing part of a RAM "word." With these implementations, you can also write more than one byte at once by enabling the appropriate byte enables.

Verilog-1995 doesn't support mixed-width RAMs because the standard lacks a multi-dimensional array to model the different read width, write width, or both. Verilog-2001 doesn't support mixed-width RAMs because this type of logic requires multiple packed dimensions. Different synthesis tools may differ in their support for these memories. This section describes the inference rules for Intel Quartus Prime Pro Edition synthesis.

Refer to the Intel Quartus Prime HDL templates for parameterized examples that you can use for different address widths, and true dual port RAM examples with two read ports and two write ports.

Example 23. SystemVerilog Simple Dual-Port Synchronous RAM with Byte Enable

```
module byte_enabled_simple_dual_port_ram
(
    input we, clk,
    input [ADDRESS_WIDTH-1:0] waddr, raddr,// address width = 6
    input [NUM_BYTES-1:0] be, // 4 bytes per word
    input [(BYTE_WIDTH * NUM_BYTES -1):0] wdata, // byte width = 8, 4 bytes per
word
    output reg [(BYTE_WIDTH * NUM_BYTES -1):0] q // byte width = 8, 4 bytes per
word
);

   parameter ADDRESS_WIDTH = 6;
   parameter DEPTH = 2**ADDRESS_WIDTH;
   parameter BYTE_WIDTH = 8;
   parameter NUM_BYTES = 4;

   // use a multi-dimensional packed array
   //to model individual bytes within the word
   logic [NUM_BYTES-1:0][BYTE_WIDTH-1:0] ram[0:DEPTH-1];
     // # words = 1 << address width

    // port A
   always@(posedge clk)
   begin
       if(we) begin
          for (int i = 0; i < NUM_BYTES; i = i + 1) begin
            if(be[i]) ram[waddr][i] <= wdata[i*BYTE_WIDTH +: BYTE_WIDTH];
          end
      end
      q <= ram[raddr];
   end
endmodule
```

Example 24. VHDL Simple Dual-Port Synchronous RAM with Byte Enable

```
library ieee;
use ieee.std_logic_1164.all;
library work;

entity byte_enabled_simple_dual_port_ram is
generic (DEPTH      : integer := 64;
         NUM_BYTES  : integer :=  4;
         BYTE_WIDTH : integer :=  8
);
port (
    we, clk : in  std_logic;
    waddr, raddr : in  integer range 0 to DEPTH -1 ;     -- address width = 6
    be   : in  std_logic_vector (NUM_BYTES-1 downto 0);   -- 4 bytes per word
    wdata: in  std_logic_vector((NUM_BYTES * BYTE_WIDTH -1) downto 0);   --
width = 32
    q    : out std_logic_vector((NUM_BYTES * BYTE_WIDTH -1) downto 0) ); --
width = 32
end byte_enabled_simple_dual_port_ram;

architecture rtl of byte_enabled_simple_dual_port_ram is

    --  build up 2D array to hold the memory
    type word_t is array (0 to NUM_BYTES-1) of std_logic_vector(BYTE_WIDTH-1
downto 0);
    type ram_t is array (0 to DEPTH-1) of word_t;

    signal ram : ram_t;
    signal q_local : word_t;
```

```
    begin  -- Re-organize the read data from the RAM to match the output
        unpack: for i in 0 to NUM_BYTES-1 generate
            q(BYTE_WIDTH*(i+1) - 1 downto BYTE_WIDTH*i) <= q_local(i);
    end generate unpack;

    -- port A
    process(clk)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                for I in (NUM_BYTES-1) downto 0 loop
                    if(be(I) = '1') then
                        ram(waddr)(I) <= wdata(((I+1)*BYTE_WIDTH-1) downto
I*BYTE_WIDTH);
                    end if;
                 end loop;
            end if;
            q_local <= ram(raddr);
        end if;
    end process;
end rtl;
```

1.4.1.11. Specifying Initial Memory Contents at Power-Up

Your synthesis tool may offer various ways to specify the initial contents of an inferred memory. There are slight power-up and initialization differences between dedicated RAM blocks and the MLAB memory, due to the continuous read of the MLAB.

Intel FPGA dedicated RAM block outputs always power-up to zero, and are set to the initial value on the first read. For example, if address `0` is pre-initialized to FF, the RAM block powers up with the output at `0`. A subsequent read after power-up from address `0` outputs the pre-initialized value of FF. Therefore, if a RAM powers up and an enable (read enable or clock enable) is held low, the power-up output of `0` maintains until the first valid read cycle. The synthesis tool implements MLAB using registers that power-up to `0`, but initialize to their initial value immediately at power-up or reset. Therefore, the initial value is seen, regardless of the enable status. The Intel Quartus Prime software maps inferred memory to MLABs when the HDL code specifies an appropriate `ramstyle` attribute.

In Verilog HDL, you can use an initial block to initialize the contents of an inferred memory. Intel Quartus Prime Pro Edition synthesis automatically converts the initial block into a Memory Initialization File (`.mif`) for the inferred RAM.

Example 25. Verilog HDL RAM with Initialized Contents

```
module ram_with_init(
   output reg [7:0] q,
   input [7:0] d,
   input [4:0] write_address, read_address,
   input we, clk
);
   reg [7:0] mem [0:31];
   integer i;

   initial begin
      for (i = 0; i < 32; i = i + 1)
         mem[i] = i[7:0];
   end

   always @ (posedge clk) begin
      if (we)
         mem[write_address] <= d;
```

```
      q <= mem[read_address];
   end
endmodule
```

Intel Quartus Prime Pro Edition synthesis and other synthesis tools also support the `$readmemb` and `$readmemh` attributes. These attributes allow RAM initialization and ROM initialization work identically in synthesis and simulation.

Example 26. Verilog HDL RAM Initialized with the readmemb Command

```
reg [7:0] ram[0:15];
initial
begin
    $readmemb("ram.txt", ram);
end
```

In VHDL, you can initialize the contents of an inferred memory by specifying a default value for the corresponding signal. Intel Quartus Prime Pro Edition synthesis automatically converts the default value into a `.mif` file for the inferred RAM.

Example 27. VHDL RAM with Initialized Contents

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY ram_with_init IS
    PORT(
            clock: IN STD_LOGIC;
            data: IN UNSIGNED (7 DOWNTO 0);
            write_address: IN integer RANGE 0 to 31;
            read_address: IN integer RANGE 0 to 31;
            we: IN std_logic;
            q: OUT UNSIGNED (7 DOWNTO 0));
END;

ARCHITECTURE rtl OF ram_with_init IS

    TYPE MEM IS ARRAY(31 DOWNTO 0) OF unsigned(7 DOWNTO 0);
    FUNCTION initialize_ram
        return MEM is
        variable result : MEM;
    BEGIN
        FOR i IN 31 DOWNTO 0 LOOP
            result(i) := to_unsigned(natural(i), natural'(8));
        END LOOP;
        RETURN result;
    END initialize_ram;

    SIGNAL ram_block : MEM := initialize_ram;
BEGIN
    PROCESS (clock)
    BEGIN
        IF (rising_edge(clock)) THEN
            IF (we = '1') THEN
            ram_block(write_address) <= data;
            END IF;
            q <= ram_block(read_address);
        END IF;
    END PROCESS;
END rtl;
```

1.4.2. Inferring ROM Functions from HDL Code

Synthesis tools infer ROMs when a CASE statement exists in which a value is set to a constant for every choice in the CASE statement.

Because small ROMs typically achieve the best performance when they are implemented using the registers in regular logic, each ROM function must meet a minimum size requirement for inference and placement in memory.

For device architectures with synchronous RAM blocks, to infer a ROM block, synthesis must use registers for either the address or the output. When your design uses output registers, synthesis implements registers from the input registers of the RAM block without affecting the functionality of the ROM. If you register the address, the power-up state of the inferred ROM can be different from the HDL design. In this scenario, Intel Quartus Prime synthesis issues a warning.

The following ROM examples map directly to the Intel FPGA memory architecture.

Example 28. Verilog HDL Synchronous ROM

```
module sync_rom (clock, address, data_out);
    input clock;
    input [7:0] address;
    output reg [5:0] data_out;
    reg [5:0] data_out;

    always @ (posedge clock)
    begin
        case (address)
            8'b00000000: data_out = 6'b101111;
            8'b00000001: data_out = 6'b110110;
            ...
            8'b11111110: data_out = 6'b000001;
            8'b11111111: data_out = 6'b101010;
        endcase
    end
endmodule
```

Example 29. VHDL Synchronous ROM

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY sync_rom IS
    PORT (
        clock: IN STD_LOGIC;
        address: IN STD_LOGIC_VECTOR(7 downto 0);
        data_out: OUT STD_LOGIC_VECTOR(5 downto 0)
    );
END sync_rom;

ARCHITECTURE rtl OF sync_rom IS
BEGIN
PROCESS (clock)
    BEGIN
    IF rising_edge (clock) THEN
        CASE address IS
            WHEN "00000000" => data_out <= "101111";
            WHEN "00000001" => data_out <= "110110";
            ...
            WHEN "11111110" => data_out <= "000001";
            WHEN "11111111" => data_out <= "101010";
            WHEN OTHERS     => data_out <= "101111";
        END CASE;
```

```
    END IF;
    END PROCESS;
END rtl;
```

Example 30. Verilog HDL Dual-Port Synchronous ROM Using readmemb

```
module dual_port_rom
#(parameter data_width=8, parameter addr_width=8)
(
    input [(addr_width-1):0] addr_a, addr_b,
    input clk,
    output reg [(data_width-1):0] q_a, q_b
);
    reg [data_width-1:0] rom[2**addr_width-1:0];

    initial // Read the memory contents in the file
             //dual_port_rom_init.txt.
    begin
        $readmemb("dual_port_rom_init.txt", rom);
    end

    always @ (posedge clk)
    begin
        q_a <= rom[addr_a];
        q_b <= rom[addr_b];
    end
endmodule
```

Example 31. VHDL Dual-Port Synchronous ROM Using Initialization Function

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dual_port_rom is
    generic (
        DATA_WIDTH : natural := 8;
        ADDR_WIDTH : natural := 8
    );
    port (
        clk     : in std_logic;
        addr_a  : in natural range 0 to 2**ADDR_WIDTH - 1;
        addr_b  : in natural range 0 to 2**ADDR_WIDTH - 1;
        q_a     : out std_logic_vector((DATA_WIDTH -1) downto 0);
        q_b     : out std_logic_vector((DATA_WIDTH -1) downto 0)
    );
end entity;

architecture rtl of dual_port_rom is
    -- Build a 2-D array type for the ROM
    subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
    type memory_t is array(2**ADDR_WIDTH - 1 downto 0) of word_t;

    function init_rom
        return memory_t is
        variable tmp : memory_t := (others => (others => '0'));
    begin
        for addr_pos in 0 to 2**ADDR_WIDTH - 1 loop
            -- Initialize each address with the address itself
            tmp(addr_pos) := std_logic_vector(to_unsigned(addr_pos, DATA_WIDTH));
        end loop;
        return tmp;
    end init_rom;

    -- Declare the ROM signal and specify a default initialization value.
    signal rom : memory_t := init_rom;
begin
    process(clk)
```

```
    begin
    if (rising_edge(clk)) then
        q_a <= rom(addr_a);
        q_b <= rom(addr_b);
    end if;
    end process;
end rtl;
```

1.4.3. Inferring Shift Registers in HDL Code

The Intel Quartus Prime software Analysis & Synthesis stage of the Compiler automatically detects and infers shift registers in your HDL code according to the following guidelines:

- Shift Register Inference for Intel Stratix 10 and Intel Agilex™ 7 Devices on page 29
- Shift Register Inference for Intel Arria 10 and Intel Cyclone 10 GX Devices on page 30

Shift Register Inference for Intel Stratix 10 and Intel Agilex™ 7 Devices

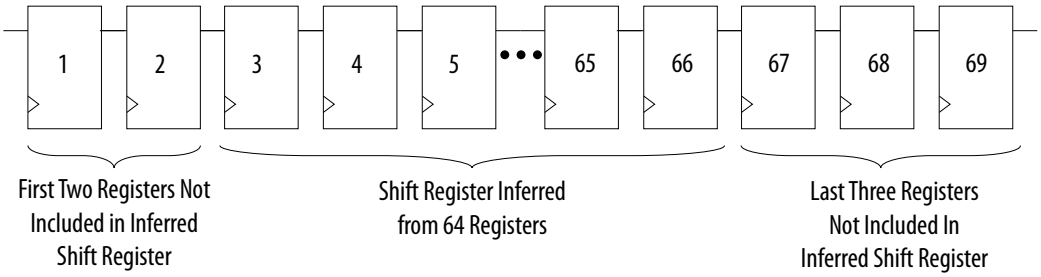
Because of the high prevalence of registers in routing segments of the Intel Hyperflex™ architecture, the Compiler's threshold for shift register inference is increased for Intel Stratix 10 and Intel Agilex™ 7 devices. This increase in the threshold means that some logic that the Compiler infers as a shift register in a previous generation FPGA, may not be inferred as a shift register when targeting Intel Stratix 10 or Intel Agilex 7 devices. This threshold increase allows more register retiming, thus improving overall design performance.

The following criteria apply to shift register detection and inference for Intel Stratix 10 and Intel Agilex 7 devices.

Default shift register inference requirements for Intel Stratix 10 and Intel Agilex 7 Devices:

1. The minimum number of registers inferred in the shift register is 64. When the width of a chain of registers is 1, the chain must contain at least 69 registers for synthesis to infer a shift register. From these 69 registers, synthesis does not include the first and second registers in the chain in the inferred shift register. Synthesis places these first and second shift registers in ALMs. Synthesis infers a 64 bit long shift register with the third through sixty sixth registers. Synthesis does not include the last three registers in the chain in the inferred shift register. Rather, synthesis places the last three registers in ALMs.
2. The minimum depth of registers inferred in the shift register is 32. When the width of a chain of registers is two or more, the chain must contain at least 37 register levels for synthesis to infer a shift register. As in the first requirement, synthesis does not include the first and second registers levels in each chain in the inferred shift register, nor are the last three register levels.

Figure 2. Shift Register Inference for Intel Stratix 10 and Intel Agilex 7 Devices



- With the following assignment, the total number of required registers (depth * width) drops to 37:

```
set_global_assignment -name ALLOW_ANY_SHIFT_REGISTER_SIZE_FOR_RECOGNITION ON
```

  Note: An additional stage of inference takes place during the early retiming stage as physical synthesis optimization recovers area for registers that have not been retimed.

- With both of the following assignments, the total number of required registers (`depth * width`) drops to 13:

```
set_global_assignment -name ALLOW_ANY_SHIFT_REGISTER_SIZE_FOR_RECOGNITION ON
set_global_assignment -name PHYSICAL_SHIFT_REGISTER_INFERENCE=OFF
```

Note: Reducing the shift register inference threshold can negatively impact design performance, as the technique reduces the number of registers available for retiming.

Shift Register Inference for Intel Arria 10 and Intel Cyclone 10 GX Devices

For Intel Arria 10 devices, Analysis & Synthesis detects a group of shift registers of the same length, and implements the registers using the Shift Register Intel FPGA IP.

For automatic detection, all of the shift registers must have the following characteristics:

- Use the same clock and clock enable
- Have no other secondary signals
- Have equally spaced taps that are at least three registers apart

Synthesis recognizes shift registers only for device families with dedicated RAM blocks. Intel Quartus Prime Pro Edition synthesis uses the following guidelines:

- The Intel Quartus Prime software determines whether to infer the Shift Register Intel FPGA IP based on the width of the registered bus (W), the length between each tap (L), or the number of taps (N).
- If the **Auto Shift Register Recognition** option is set to **Auto**, Intel Quartus Prime Pro Edition synthesis determines which shift registers are implemented in RAM blocks for logic by using the following methods:
  - The **Optimization Technique** setting
  - Logic and RAM utilization information about the design
  - Timing information from **Timing-Driven Synthesis**
- If the registered bus width is one ($W = 1$), Intel Quartus Prime synthesis infers the shift register IP if the number of taps, times the length between each tap, is greater than or equal to 64 ($N \times L > 64$).
- If the registered bus width is greater than one ($W > 1$), and the registered bus width times the number of taps times the length between each tap is greater than or equal to 32 ($W \times N \times L > 32$), then Intel Quartus Prime synthesis the Shift Register Intel FPGA IP.
- If the length between each tap (L) is not a power of two, Intel Quartus Prime synthesis needs external logic (LEs or ALMs) to decode the read and write counters, because of different sizes of shift registers. This extra decode logic eliminates the performance and utilization advantages of implementing shift registers in memory.

The registers that Intel Quartus Prime synthesis maps to the Shift Register Intel FPGA IP, and places in RAM are not available in a Verilog HDL or VHDL output file for simulation tools, because their node names do not exist after synthesis.

Note: The Compiler cannot implement a shift register that uses a `shift enable` signal into MLAB memory; instead, the Compiler uses dedicated RAM blocks. To control the type of memory structure that implements the shift register, use the `ramstyle` attribute. For example:

```
(* ramstyle = "mlab" *) my_shift_reg
```

1.4.3.1. Simple Shift Register

The examples in this section show a simple, single-bit wide, 69-bit long shift register.

Intel Quartus Prime synthesis implements the register (`W = 1` and `M = 69`) by using the Shift Register Intel FPGA IP, and maps it to RAM in the device, which may be placed in dedicated RAM blocks or MLAB memory. If the length of the register is less than 69 bits, Intel Quartus Prime synthesis implements the shift register in logic.

Example 32. Verilog HDL Single-Bit Wide, 69-Bit Long Shift Register

```
module shift_1x69 (clk, shift, sr_in, sr_out);
    input clk, shift;
    input sr_in;
    output sr_out;

    reg [68:0] sr;

    always @ (posedge clk)
    begin
        if (shift == 1'b1)
        begin
            sr[68:1] <= sr[67:0];
            sr[0] <= sr_in;
        end
    end
    assign sr_out = sr[68];
endmodule
```

Example 33. VHDL Single-Bit Wide, 69-Bit Long Shift Register

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
ENTITY shift_1x69 IS
    PORT (
        clk: IN STD_LOGIC;
        shift: IN STD_LOGIC;
        sr_in: IN STD_LOGIC;
        sr_out: OUT STD_LOGIC
    );
END shift_1x69;

ARCHITECTURE arch OF shift_1x69 IS
    TYPE sr_length IS ARRAY (68 DOWNTO 0) OF STD_LOGIC;
    SIGNAL sr: sr_length;
BEGIN
    PROCESS (clk)
        BEGIN
        IF (rising_edge(clk)) THEN
            IF (shift = '1') THEN
            sr(68 DOWNTO 1) <= sr(67 DOWNTO 0);
            sr(0) <= sr_in;
            END IF;
```

```
        END IF;
    END PROCESS;
    sr_out <= sr(68);
END arch;
```

1.4.3.2. Shift Register with Evenly Spaced Taps

The following examples show a Verilog HDL and VHDL 8-bit wide, 255-bit long shift register (`W > 1` and `M = 255`) with evenly spaced taps at 64, 128, 192, and 254.

The synthesis software implements this function in a single ALTSHIFT_TAPS IP core and maps it to RAM in supported devices, which is allowed placement in dedicated RAM blocks or MLAB memory.

Example 34. Verilog HDL 8-Bit Wide, 255-Bit Long Shift Register with Evenly Spaced Taps

```
module top (clk, shift, sr_in, sr_out, sr_tap_one, sr_tap_two,
                sr_tap_three );
    input clk, shift;
    input [7:0] sr_in;
    output [7:0] sr_tap_one, sr_tap_two, sr_tap_three, sr_out;
    reg [7:0] sr [254:0];
    integer n;
    always @ (posedge clk)
        begin
        if (shift == 1'b1)
            begin
            for (n = 254; n>0; n = n-1)
                begin
                sr[n] <= sr[n-1];
                end
            sr[0] <= sr_in;
        end
    end
    assign sr_tap_one = sr[64];
    assign sr_tap_two = sr[128];
    assign sr_tap_three = sr[192];
    assign sr_out = sr[254];
endmodule
```

1.4.4. Inferring FIFOs in HDL Code

There are various methods of implementing dual clock FIFOs, depending on the features needed in your design. The following dual clock FIFO example shows the basic FIFO functionality, with a design goal of high speed (f_{MAX}) and small area.

The FIFO supports parameterization up to 32 words deep, and targets memory LABs (MLABs) for its memory block. Synthesis infers the MLABs from behavioral RTL in the `generic_mlab_dc` module.

Note: If you don't want to code your own FIFO, you can parameterize the dual clock FIFO IP with the IP parameter editor in the Intel Quartus Prime software. Refer to the *FIFO Intel FPGA IP User Guide*.

Related Information

FIFO Intel FPGA IP User Guide

1.4.4.1. Dual Clock FIFO Example in Verilog HDL

```
// Copyright 2021 Intel Corporation.
//
// This reference design file is subject licensed to you by the terms and
// conditions of the applicable License Terms and Conditions for Hardware
// Reference Designs and/or Design Examples (either as signed by you or
// found at https://www.altera.com/common/legal/leg-license_agreement.html ).
//
// As stated in the license, you agree to only use this reference design
// solely in conjunction with Intel FPGAs or Intel CPLDs.
//
// THE REFERENCE DESIGN IS PROVIDED "AS IS" WITHOUT ANY EXPRESS OR IMPLIED
// WARRANTY OF ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY,
// NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. Intel does not
// warrant or assume responsibility for the accuracy or completeness of any
// information, links or other items within the Reference Design and any
// accompanying materials.
//
// In the event that you do not agree with such terms and conditions, do not
// use the reference design file.
////////////////////////////////////////////////////////////////////////////////

module dcfifo_example
#(
    parameter LOG_DEPTH      = 5,
    parameter WIDTH          = 20,
    parameter ALMOST_FULL_VALUE = 30,
    parameter ALMOST_EMPTY_VALUE = 2,
    parameter NUM_WORDS = 2**LOG_DEPTH - 1,
    parameter OVERFLOW_CHECKING = 0, // Overflow checking circuitry is \
       using extra area. Use only if you need it
    parameter UNDERFLOW_CHECKING = 0 // Underflow checking circuitry is \
       using extra area. Use only if you need it
)
(
    input aclr,

    input wrclk,
    input wrreq,
    input [WIDTH-1:0] data,
    output reg wrempty,
    output reg wrfull,
    output reg wr_almost_empty,
    output reg wr_almost_full,
    output [LOG_DEPTH-1:0] wrusedw,

    input rdclk,
    input rdreq,
    output [WIDTH-1:0] q,
    output reg rdempty,
    output reg rdfull,
    output reg rd_almost_empty,
    output reg rd_almost_full,
    output [LOG_DEPTH-1:0] rdusedw
);

initial begin
    if ((LOG_DEPTH > 5) || (LOG_DEPTH < 3))
        $error("Invalid parameter value: LOG_DEPTH = %0d; valid range is 2 \
        < LOG_DEPTH < 6", LOG_DEPTH);

    if ((ALMOST_FULL_VALUE > 2 ** LOG_DEPTH - 1) || (ALMOST_FULL_VALUE < 1))
        $error("Incorrect parameter value: ALMOST_FULL_VALUE = %0d; valid \
        range is 0 < ALMOST_FULL_VALUE < %0d",
            ALMOST_FULL_VALUE, 2 ** LOG_DEPTH);

    if ((ALMOST_EMPTY_VALUE > 2 ** LOG_DEPTH - 1) || (ALMOST_EMPTY_VALUE < 1))
        $error("Incorrect parameter value: ALMOST_EMPTY_VALUE = %0d; valid \
        range is 0 < ALMOST_EMPTY_VALUE < %0d",
```

```
            ALMOST_EMPTY_VALUE, 2 ** LOG_DEPTH);

    if ((NUM_WORDS > 2 ** LOG_DEPTH - 1) || (NUM_WORDS < 1))
        $error("Incorrect parameter value: NUM_WORDS = %0d; \
        valid range is 0 < NUM_WORDS < %0d",
            NUM_WORDS, 2 ** LOG_DEPTH);
end

(* altera_attribute = "-name AUTO_CLOCK_ENABLE_RECOGNITION OFF" *) reg \
     [LOG_DEPTH-1:0] write_addr = 0;
(* altera_attribute = "-name AUTO_CLOCK_ENABLE_RECOGNITION OFF" *) reg \
     [LOG_DEPTH-1:0] read_addr = 0;
reg [LOG_DEPTH-1:0] wrcapacity = 0;
reg [LOG_DEPTH-1:0] rdcapacity = 0;

wire [LOG_DEPTH-1:0] wrcapacity_w;
wire [LOG_DEPTH-1:0] rdcapacity_w;

wire [LOG_DEPTH-1:0] rd_write_addr;
wire [LOG_DEPTH-1:0] wr_read_addr;

wire wrreq_safe;
wire rdreq_safe;
assign wrreq_safe = OVERFLOW_CHECKING ? wrreq & ~wrfull : wrreq;
assign rdreq_safe = UNDERFLOW_CHECKING ? rdreq & ~rdempty : rdreq;

initial begin
    write_addr = 0;
    read_addr = 0;
    wrempty = 1;
    wrfull = 0;
    rdempty = 1;
    rdfull = 0;
    wrcapacity = 0;
    rdcapacity = 0;
    rd_almost_empty = 1;
    rd_almost_full = 0;
    wr_almost_empty = 1;
    wr_almost_full = 0;
end


// ------------------ Write ------------------------

add_a_b_s0_s1 #(LOG_DEPTH) wr_adder(
    .a(write_addr),
    .b(~wr_read_addr),
    .s0(wrreq_safe),
    .s1(1'b1),
    .out(wrcapacity_w)
);

always @(posedge wrclk or posedge aclr) begin

    if (aclr) begin
        write_addr <= 0;
        wrcapacity <= 0;
        wrempty <= 1;
        wrfull <= 0;
        wr_almost_full <= 0;
        wr_almost_empty <= 1;
    end else begin
        write_addr <= write_addr + wrreq_safe;
        wrcapacity <= wrcapacity_w;
        wrempty <= (wrcapacity == 0) && (wrreq == 0);
        wrfull <= (wrcapacity == NUM_WORDS) || (wrcapacity == NUM_WORDS - 1) \
        && (wrreq == 1);

        wr_almost_empty <=
            (wrcapacity < (ALMOST_EMPTY_VALUE-1)) ||
            (wrcapacity == (ALMOST_EMPTY_VALUE-1)) && (wrreq == 0);
```

```
        wr_almost_full <=
            (wrcapacity >= ALMOST_FULL_VALUE) ||
            (wrcapacity == ALMOST_FULL_VALUE - 1) && (wrreq == 1);
    end
end

assign wrusedw = wrcapacity;

// ------------------ Read ------------------------

add_a_b_s0_s1 #(LOG_DEPTH) rd_adder(
    .a(rd_write_addr),
    .b(~read_addr),
    .s0(1'b0),
    .s1(~rdreq_safe),
    .out(rdcapacity_w)
);

always @(posedge rdclk or posedge aclr) begin
    if (aclr) begin
        read_addr <= 0;
        rdcapacity <= 0;
        rdempty <= 1;
        rdfull <= 0;
        rd_almost_empty <= 1;
        rd_almost_full <= 0;
    end else begin
        read_addr <= read_addr + rdreq_safe;
        rdcapacity <= rdcapacity_w;
        rdempty <= (rdcapacity == 0) || (rdcapacity == 1) && (rdreq == 1);
        rdfull <= (rdcapacity == NUM_WORDS) && (rdreq == 0);
        rd_almost_empty <=
            (rdcapacity < ALMOST_EMPTY_VALUE) ||
            (rdcapacity == ALMOST_EMPTY_VALUE) && (rdreq == 1);

        rd_almost_full <=
            (rdcapacity > ALMOST_FULL_VALUE) ||
            (rdcapacity == ALMOST_FULL_VALUE) && (rdreq == 0);
    end
end

assign rdusedw = rdcapacity;

// ---------------- Synchronizers --------------------

wire [LOG_DEPTH-1:0] gray_read_addr;
wire [LOG_DEPTH-1:0] wr_gray_read_addr;
wire [LOG_DEPTH-1:0] gray_write_addr;
wire [LOG_DEPTH-1:0] rd_gray_write_addr;

binary_to_gray #(.WIDTH(LOG_DEPTH)) rd_b2g (.clock(rdclk), .aclr(aclr), \
     .din(read_addr), .dout(gray_read_addr));
synchronizer_ff_r2 #(.WIDTH(LOG_DEPTH)) rd2wr
(.din_clk(rdclk), .din(gray_read_addr), \
     .dout_clk(wrclk), .dout(wr_gray_read_addr));
gray_to_binary #(.WIDTH(LOG_DEPTH)) rd_g2b (.clock(wrclk), .aclr(aclr), \
     .din(wr_gray_read_addr), .dout(wr_read_addr));


binary_to_gray #(.WIDTH(LOG_DEPTH)) wr_b2g
(.clock(wrclk), .aclr(aclr), .din(write_addr), \
     .dout(gray_write_addr));
synchronizer_ff_r2 #(.WIDTH(LOG_DEPTH)) wr2rd
(.din_clk(wrclk), .din(gray_write_addr), \
     .dout_clk(rdclk), .dout(rd_gray_write_addr));
gray_to_binary #(.WIDTH(LOG_DEPTH)) wr_g2b (.clock(rdclk), .aclr(aclr), \
     .din(rd_gray_write_addr), .dout(rd_write_addr));

// ------------------ MLAB ----------------------------
```

```
generic_mlab_dc #(.WIDTH(WIDTH), .ADDR_WIDTH(LOG_DEPTH)) mlab_inst (
    .rclk(rdclk),
    .wclk(wrclk),
    .din(data),
    .waddr(write_addr),
    .we(1'b1),
    .re(1'b1),
    .raddr(read_addr),
    .dout(q)
);

endmodule

module add_a_b_s0_s1 #(
    parameter SIZE = 5
)(
    input [SIZE-1:0] a,
    input [SIZE-1:0] b,
    input s0,
    input s1,
    output [SIZE-1:0] out
);
    wire [SIZE:0] left;
    wire [SIZE:0] right;
    wire temp;

    assign left = {a ^ b, s0};
    assign right = {a[SIZE-2:0] & b[SIZE-2:0], s1, s0};
    assign {out, temp} = left + right;

endmodule

module binary_to_gray #(
    parameter WIDTH = 5
) (
    input clock,
    input aclr,
    input [WIDTH-1:0] din,
    output reg [WIDTH-1:0] dout
);

    always @(posedge clock or posedge aclr) begin
         if (aclr)
              dout <= 0;
         else
              dout <= din ^ (din >> 1);
    end

endmodule

module gray_to_binary #(
    parameter WIDTH = 5
) (
    input clock,
    input aclr,
    input [WIDTH-1:0] din,
    output reg [WIDTH-1:0] dout
);

    wire [WIDTH-1:0] dout_w;

    genvar i;
    generate
         for (i = 0; i < WIDTH; i=i+1) begin : loop
              assign dout_w[i] = ^(din[WIDTH-1:i]);
         end
    endgenerate

    always @(posedge clock or posedge aclr) begin
         if (aclr)
              dout <= 0;
```

```
          else
               dout <= dout_w;
    end

endmodule

(* altera_attribute = "-name SYNCHRONIZER_IDENTIFICATION OFF" *)
module generic_mlab_dc #(
    parameter WIDTH = 8,
    parameter ADDR_WIDTH = 5
)(
    input rclk,
    input wclk,
    input [WIDTH-1:0] din,
    input [ADDR_WIDTH-1:0] waddr,
    input we,
    input re,
    input [ADDR_WIDTH-1:0] raddr,
    output [WIDTH-1:0] dout
);

    localparam DEPTH = 1 << ADDR_WIDTH;
    (* ramstyle = "mlab" *) reg [WIDTH-1:0] mem[0:DEPTH-1];

    reg [WIDTH-1:0] dout_r;
    always @(posedge wclk) begin
         if (we)
              mem[waddr] <= din;
    end
    always @(posedge rclk) begin
         if (re)
              dout_r <= mem[raddr];
    end
    assign dout = dout_r;

endmodule


module synchronizer_ff_r2 #(
    parameter WIDTH = 8
)(
    input din_clk,
    input [WIDTH-1:0] din,
    input dout_clk,
    output [WIDTH-1:0] dout
);

    reg [WIDTH-1:0] ff_launch = {WIDTH {1'b0}}
    /* synthesis preserve dont_replicate */;
    always @(posedge din_clk) begin
         ff_launch <= din;
    end

    reg [WIDTH-1:0] ff_meta = {WIDTH {1'b0}}
    /* synthesis preserve dont_replicate */;
    always @(posedge dout_clk) begin
         ff_meta <= ff_launch;
    end

    reg [WIDTH-1:0] ff_sync = {WIDTH {1'b0}}
   /* synthesis preserve dont_replicate */;
    always @(posedge dout_clk) begin
         ff_sync <= ff_meta;
    end

    assign dout = ff_sync;
endmodule
```

1.4.4.2. Dual Clock FIFO Timing Constraints

If you choose to code your own dual clock FIFO, you must also create appropriate timing constraints in Synopsis Design Constraints format (`.sdc`).

Typically, you set the read and write clock domains asynchronous to each other by using the `set_clock_groups` SDC command. You typically specify the `set_clock_groups` command in a top-level `.sdc` file.

Constrain the read and write pointer clock domain crossings with skew and net delay constraints.

A skew constraint ensures the gray-coded pointer values transfer correctly between the clock domains. A net delay constraint bounds the wire delay between the two clock domains, to help reduce latency through the FIFO.

In the RTL example above, the pointers cross clock domains at the `ff_launch` to `ff_meta` register path, in two instances of the `synchronizer_ff_r2` entity.

The following example constraints are appropriate for the RTL above. You can customize the `-from` and `-to` names as necessary for your implementation.

```
# Skew from read to write domain
set_max_skew -from rd2wr|ff_launch[*] -to rd2wr|ff_meta[*] \
     -get_skew_value_from_clock_period
src_clock_period -skew_value_multiplier 0.8
# Skew from write to read domain
set_max_skew -from wr2rd|ff_launch[*] -to wr2rd|ff_meta[*] \
     -get_skew_value_from_clock_period
src_clock_period -skew_value_multiplier 0.8
# Net delay from read to write domain
set_net_delay -from rd2wr|ff_launch[*] -to rd2wr|ff_meta[*] \
     -get_value_from_clock_period
dst_clock_period -value_multiplier 0.8 -max
# Net delay from write to read domain
set_net_delay -from wr2rd|ff_launch[*] -to wr2rd|ff_meta[*] \
     -get_value_from_clock_period
dst_clock_period -value_multiplier 0.8 -max
```

After writing the skew and net delay constraints in the `.sdc` file, specify an entity-bound `.sdc` file `.qsf` assignment to apply the constraints to the synchronizer register paths in all instances of your FIFO.

Use the name of the `.sdc` file containing these constraints in the entity-bound `.sdc` file assignment in your `.qsf`. Also provide the name of the FIFO entity to which the constraints apply.

The following `.qsf` assignment example assumes that you save the constraints in `fifo_synchronizer.sdc` in your project directory, and that the constraints therein apply to the `dcfifo_example` entity:

```
set_global_assignment -name SDC_ENTITY_FILE fifo_synchronizer.sdc \
     -entity dcfifo_example
```

1.5. Register and Latch Coding Guidelines

This section provides device-specific coding recommendations for Intel registers and latches. Understanding the architecture of the target Intel device helps ensure that your RTL produces the expected results and achieves the optimal quality of results.

1.5.1. Register Power-Up Values

Registers in the device core power-up to a low (0) logic level on all Intel FPGA devices. However, for designs that specify a power-up level other than 0, synthesis tools can implement logic that directs registers to behave as if they were powering up to a high (1) logic level.

For designs that use `preset` signals, but the target device does not support presets in the register architecture, synthesis may convert the `preset` signal to a `clear` signal, which requires to perform a NOT gate push-back optimization. NOT gate push-back adds an inverter to the input and the output of the register, so that the reset and power-up conditions appear high, and the device operates as expected. In this case, the synthesis tool may issue a message about the power-up condition. The register itself powers up low, but since the register output inverts, the signal that arrives at all destinations is high.

Due to these effects, if you specify a non-zero reset value, the synthesis tool may use the asynchronous clear (`aclr`) signals available on the registers to implement the high bits with NOT gate push-back. In that case, the registers look as though they power-up to the specified reset value.

When an asynchronous load (`aload`) signal is available in the device registers, the synthesis tools can implement a reset of `1` or `0` value by using an asynchronous load of `1` or `0`. When the synthesis tool uses a `load` signal, it is not performing NOT gate push-back, so the registers power-up to a `0` logic level. For additional details, refer to the appropriate device family handbook.

Optionally you can force all registers into their appropriate values after reset through an explicit reset signal. This technique allows to reset the device after power-up to restore the proper state.

Synchronizing the device architecture's external or combinational logic before driving the register's asynchronous control ports allows for more stable designs and avoids potential glitches.

Related Information

Recommended Design Practices on page 69

1.5.1.1. Specifying a Power-Up Value

Options available in synthesis tools allow you to specify power-up conditions for the design. Intel Quartus Prime Pro Edition synthesis provides the **Power-Up Level** logic option.

You can also specify the power-up level with an `altera_attribute` assignment in the source code. This attribute forces synthesis to perform NOT gate push-back, because synthesis tools cannot change the power-up states of core registers.

You can apply the **Power-Up Level** logic option to a specific register, or to a design entity, module, or sub design. When you assign this option, every register in that block receives the value. Registers power up to `0` by default. Therefore, you can use this assignment to force all registers to power-up to `1` using NOT gate push-back.

Setting the **Power-Up Level** to a logic level of **high** for a large design entity could degrade the quality of results due to the number of inverters that requires. In some situations, this design style causes issues due to `enable` signal inference or secondary control logic inference. It may also be more difficult to migrate this type of designs.

Some synthesis tools can also read the default or initial values for registered signals and implement this behavior in the device. For example, Intel Quartus Prime Pro Edition synthesis converts default values for registered signals into **Power-Up Level** settings. When the Intel Quartus Prime software reads the default values, the synthesized behavior matches the power-up state of the HDL code during a functional simulation.

Example 35. Verilog Register with High Power-Up Value

```
reg q = 1'b1; //q has a default value of '1'

always @ (posedge clk)
begin
    q <= d;
end
```

Example 36. VHDL Register with High Power-Up Level

```
SIGNAL q : STD_LOGIC := '1'; -- q has a default value of '1'

PROCESS (clk, reset)
BEGIN
    IF (rising_edge(clk)) THEN
        q <= d;
    END IF;
END PROCESS;
```

Your design may contain undeclared default power-up conditions based on signal type. If you declare a VHDL register signal as an integer, Intel Quartus Prime synthesis uses the left end of the integer range as the power-up value. For the default signed integer type, the default power-up value is the highest magnitude negative integer (`100…001`). For an unsigned integer type, the default power-up value is `0`.

Note: If the target device architecture does not support two asynchronous control signals, such as `aclr` and `aload`, you cannot set a different power-up state and reset state. If the NOT gate push-back algorithm creates logic to set a register to `1`, that register powers-up high. If you set a different power-up condition through a synthesis attribute or initial value, synthesis ignores the power-up level.

1.5.2. Secondary Register Control Signals Such as Clear and Clock Enable

The registers in Intel FPGAs provide a number of secondary control signals. Use these signals to implement control logic for each register without using extra logic cells. Intel FPGA device families vary in their support for secondary signals, so consult the device family data sheet to verify which signals are available in your target device.

To make the most efficient use of the signals in the device, ensure that HDL code matches the device architecture as closely as possible. The control signals have a certain priority due to the nature of the architecture. Your HDL code must follow that priority where possible.

Your synthesis tool can emulate any control signals using regular logic, so achieving functionally correct results is always possible. However, if your design requirements allow flexibility in controlling use and priority of control signals, match your design to the target device architecture to achieve the most efficient results. If the priority of the signals in your design is not the same as that of the target architecture, you may require extra logic to implement the control signals. This extra logic uses additional device resources, and can cause additional delays for the control signals.

In certain cases, using logic other than the dedicated control logic in the device architecture can have a larger impact. For example, the `clock enable` signal has priority over the synchronous `reset` or `clear` signal in the device architecture. The `clock enable` turns off the clock line in the LAB, and the `clear` signal is synchronous. Therefore, in the device architecture, the synchronous clear takes effect only when a clock edge occurs.

If you define a register with a synchronous `clear` signal that has priority over the `clock enable` signal, Intel Quartus Prime synthesis emulates the clock enable functionality using data inputs to the registers. You cannot apply a Clock Enable Multicycle constraint, because the emulated functionality does not use the `clock enable` port of the register. In this case, using a different priority causes unexpected results with an assignment to the `clock enable` signal.

The signal order is the same for all Intel FPGA device families. However, not all device families provide every signal. The priority order is:

1. Asynchronous Clear (`clrn`)—highest priority
2. Enable (`ena`)
3. Synchronous Clear (`sclr`)
4. Synchronous Load (`sload`)
5. Data In (`data`)—lowest priority

The priority order for secondary control signals in Intel FPGA devices differs from the order for other vendors' FPGA devices. If your design requirements are flexible regarding priority, verify that the secondary control signals meet design performance requirements when migrating designs between FPGA vendors. To achieve the best results. try to match your target device architecture.

Example 37. Verilog D-type Flipflop bus with Secondary Signals

This module uses all Intel Arria 10 DFF secondary signals: `clrn`, `ena`, `sclr`, and `sload`. Note that it instantiates 8-bit bus of DFFs rather than a single DFF, because synthesis infers some secondary signals only if there are multiple DFFs with the same secondary signal.

```
module top(clk, clrn, sclr, sload, ena, data, sdata, q);
    input clk, clrn, sclr, sload, ena;
    input [7:0] data, sdata;
    output [7:0] q;
    reg [7:0] q;
    always @ (posedge clk or posedge clrn)
        begin
        if (clrn)
            q <= 8'b0;
        else if (ena)
            begin
            if (sclr)
                q <= 8'b0;
```

```
            else if (!sload)
                q <= data;
            else
                q <= sdata;
            end
        end
endmodule
```

Related Information

Intel® Quartus® Prime Timing Analyzer Cookbook

1.5.3. Latches

A latch is a small combinational loop that holds the value of a signal until a new value is assigned. Synthesis tools can infer latches from HDL code when you did not intend to use a latch. If you do intend to infer a latch, it is important to infer it correctly to guarantee correct device operation.

Note: Design without the use of latches whenever possible.

Related Information

Avoid Unintended Latch Inference on page 72

1.5.3.1. Avoid Unintentional Latch Generation

When you design combinational logic, certain coding styles can create an unintentional latch. For example, when CASE or IF statements do not cover all possible input conditions, synthesis tools can infer latches to hold the output if a new output value is not assigned. Check your synthesis tool messages for references to inferred latches.

If your code unintentionally creates a latch, modify your RTL to remove the latch:

- Synthesis infers a latch when HDL code assigns a value to a signal outside of a clock edge (for example, with an asynchronous `reset`), but the code does not assign a value in an edge-triggered design block.
- Unintentional latches also occur when HDL code assigns a value to a signal in an edge-triggered design block, but synthesis optimizations remove that logic. For example, when a CASE or IF statement tests a condition that only evaluates to FALSE, synthesis removes any logic or signal assignment in that statement during optimization. This optimization may result in the inference of a latch for the signal.
- Omitting the final ELSE or WHEN OTHERS clause in an IF or CASE statement can also generate a latch. Don't care (X) assignments on the default conditions are useful in preventing latch generation. For the best logic optimization, assign the default CASE or final ELSE value to don't care (X) instead of a logic value.

In Verilog HDL designs, use the `full_case` attribute to treat unspecified cases as don't care values (X). However, since the `full_case` attribute is synthesis-only, it can cause simulation mismatches, because simulation tools still treat the unspecified cases as latches.

Example 38. VHDL Code Preventing Unintentional Latch Creation

Without the final ELSE clause, the following code creates unintentional latches to cover the remaining combinations of the SEL inputs. When you are targeting a Stratix series device with this code, omitting the final ELSE condition can cause synthesis

tools to use up to six LEs, instead of the three it uses with the ELSE statement. Additionally, assigning the final ELSE clause to 1 instead of X can result in slightly more LEs, because synthesis tools cannot perform as much optimization when you specify a constant value as opposed to a don't care value.

```
LIBRARY ieee;
USE IEEE.std_logic_1164.all;

ENTITY nolatch IS
    PORT (a,b,c: IN STD_LOGIC;
        sel: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
        oput: OUT STD_LOGIC);
END nolatch;

ARCHITECTURE rtl OF nolatch IS
BEGIN
    PROCESS (a,b,c,sel) BEGIN
        IF sel = "00000" THEN
            oput <= a;
        ELSIF sel = "00001" THEN
            oput <= b;
        ELSIF sel = "00010" THEN
            oput <= c;
        ELSE                    --- Prevents latch inference
            oput <= 'X'; --/
        END IF;
    END PROCESS;
END rtl;
```

1.5.3.2. Inferring Latches Correctly

Synthesis tools can infer a latch that does not exhibit the glitch and timing hazard problems typically associated with combinational loops. Intel Quartus Prime Pro Edition software reports latches that synthesis inferred in the **User-Specified and Inferred Latches** section of the Compilation Report. This report indicates whether the latch presents a timing hazard, and the total number of user-specified and inferred latches.

Note: In some cases, timing analysis does not completely model latch timing. As a best practice, avoid latches unless required by the design and you fully understand the impact.

If latches or combinational loops in the design do not appear in the **User Specified and Inferred Latches** section, then Intel Quartus Prime synthesis did not infer the latch as a safe latch, so the latch is not considered glitch-free.

All combinational loops listed in the **Analysis & Synthesis Logic Cells Representing Combinational Loops** table in the Compilation Report are at risk of timing hazards. These entries indicate possible problems with the design that require further investigation. However, correct designs can include combinational loops. For example, it is possible that the combinational loop cannot be sensitized. This occurs when there is an electrical path in the hardware, but either:

- The designer knows that the circuit never encounters data that causes that path to be activated, or
- The surrounding logic is set up in a mutually exclusive manner that prevents that path from ever being sensitized, independent of the data input.

For 6-input LUT-based devices, Intel Quartus Prime synthesis implements all latch inputs with a single adaptive look-up table (ALUT) in the combinational loop. Therefore, all latches in the **User-Specified and Inferred Latches** table are free of timing hazards when a single input changes.

If Intel Quartus Prime synthesis report lists a latch as a safe latch, other optimizations, such as physical synthesis netlist optimizations in the Fitter, maintain the hazard-free performance. To ensure hazard-free behavior, only one control input can change at a time. Changing two inputs simultaneously, such as deasserting `set` and `reset` at the same time, or changing data and enable at the same time, can produce incorrect behavior in any latch.

Intel Quartus Prime synthesis infers latches from `always` blocks in Verilog HDL and `process` statements in VHDL. However, Intel Quartus Prime synthesis does not infer latches from continuous assignments in Verilog HDL, or concurrent signal assignments in VHDL. These rules are the same as for register inference. The Intel Quartus Prime synthesis infers registers or flipflops only from `always` blocks and `process` statements.

Example 39. Verilog HDL Set-Reset Latch

```
module simple_latch (
   input SetTerm,
   input ResetTerm,
   output reg LatchOut
   );
   always @ (SetTerm or ResetTerm) begin
      if (SetTerm)
         LatchOut = 1'b1;
      else if (ResetTerm)
         LatchOut = 1'b0;
   end
endmodule
```

Example 40. VHDL Data Type Latch

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY simple_latch IS
    PORT (
        enable, data    : IN STD_LOGIC;
        q               : OUT STD_LOGIC
    );
END simple_latch;
ARCHITECTURE rtl OF simple_latch IS
BEGIN
    latch : PROCESS (enable, data)
        BEGIN
        IF (enable = '1') THEN
            q <= data;
        END IF;
    END PROCESS latch;
END rtl;
```

The following example shows a Verilog HDL continuous assignment that does not infer a latch in the Intel Quartus Prime software:

Example 41. Verilog Continuous Assignment Does Not Infer Latch

```
assign latch_out = (~en & latch_out) | (en & data);
```

The behavior of the assignment is similar to a latch, but it may not function correctly as a latch, and its timing is not analyzed as a latch. Intel Quartus Prime Pro Edition synthesis also creates safe latches when possible for instantiations of an Intel FPGA latch IP core. Intel FPGA latch IPs allow you to define a latch with any combination of data, enable, set, and `reset` inputs. The same limitations apply for creating safe latches as for inferring latches from HDL code.

Inferring the Intel FPGA latch IP core in another synthesis tool ensures that Intel Quartus Prime synthesis also recognizes the implementation as a latch. If a third-party synthesis tool implements a latch using the Intel FPGA latch IP core, Intel Quartus Prime Pro Edition synthesis reports the latch in the **User-Specified and Inferred Latches** table, in the same manner as it lists latches you define in HDL source code. The coding style necessary to produce an Intel FPGA latch IP core implementation depends on the synthesis tool. Some third-party synthesis tools list the number of Intel FPGA latch IP cores that are inferred.

The Fitter uses global routing for control signals, including signals that synthesis identifies as latch enables. In some cases, the global insertion delay decreases timing performance. If necessary, you can turn off the **Intel Quartus Prime Global Signal** logic option to manually prevent the use of global signals. The **Global & Other Fast Signals** table in the Compilation Report reports Global latch enables.

1.6. General Coding Guidelines

This section describes how coding styles impact synthesis of HDL code into the target Intel FPGA devices. You can improve your design efficiency and performance by following these recommended coding styles, and designing logic structures to match the appropriate device architecture.

1.6.1. Tri-State Signals

Use tri-state signals only when they are attached to top-level bidirectional or output pins.

Avoid lower-level bidirectional pins. Also avoid using the Z logic value unless it is driving an output or bidirectional pin. Even though some synthesis tools implement designs with internal tri-state signals correctly in Intel FPGA devices using multiplexer logic, do not use this coding style for Intel FPGA designs.

Note: In hierarchical block-based design flows, a hierarchical boundary cannot contain any bidirectional ports, unless the lower-level bidirectional port is connected directly through the hierarchy to a top-level output pin without connecting to any other design logic. If you use boundary tri-states in a lower-level block, synthesis software must push the tri-states through the hierarchy to the top level to make use of the tri-state drivers on output pins of Intel FPGA devices. Because pushing tri-states requires optimizing through hierarchies, lower-level tri-states are restricted with block-based design methodologies.

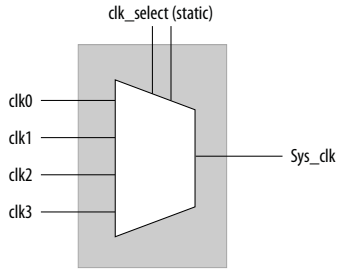
1.6.2. Clock Multiplexing

Clock multiplexing is sometimes used to operate the same logic function with different clock sources. This type of logic can introduce glitches that create functional problems. The delay inherent in the combinational logic can also lead to timing problems. Clock multiplexers trigger warnings from a wide range of design rule check and timing analysis tools.

Use dedicated hardware to perform clock multiplexing when it is available, instead of using multiplexing logic. For example, you can use the Clock Switchover feature or the Clock Control Block available in certain Intel FPGA devices. These dedicated hardware blocks avoid glitches, ensure that you use global low-skew routing lines, and avoid any possible hold time problems on the device due to logic delay on the clock line. Intel FPGA devices also support dynamic PLL reconfiguration, which is the safest and most robust method of changing clock rates during device operation.

If your design has too many clocks to use the clock control block, or if dynamic reconfiguration is too complex for your design, you can implement a clock multiplexer in logic cells. However, if you use this implementation, consider simultaneous toggling inputs and ensure glitch-free transitions.

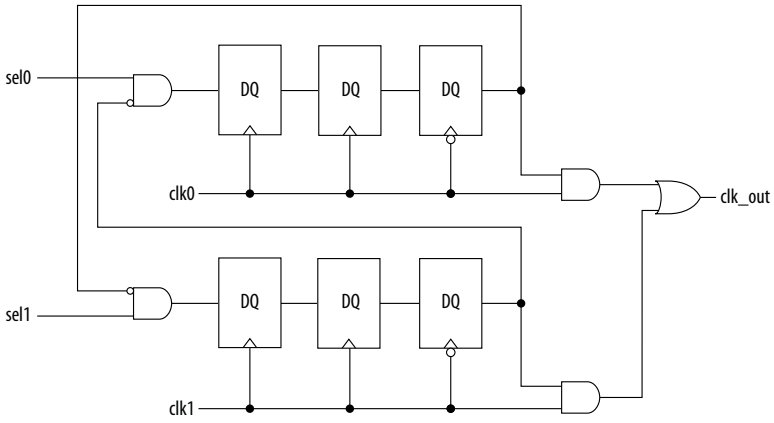
Figure 3. Simple Clock Multiplexer in a 6-Input LUT



Each device datasheet describes how LUT outputs can glitch during a simultaneous toggle of input signals, independent of the LUT function. Even though the 4:1 MUX function does not generate detectable glitches during simultaneous data input toggles, some cell implementations of multiplexing logic exhibit significant glitches, so this clock mux structure is not recommended. An additional problem with this implementation is that the output behaves erratically during a change in the `clk_select` signals. This behavior could create timing violations on all registers fed by the system clock and result in possible metastability.

A more sophisticated clock select structure can eliminate the simultaneous toggle and switching problems.

Figure 4. Glitch-Free Clock Multiplexer Structure



You can generalize this structure for any number of clock channels. The design ensures that no clock activates until all others are inactive for at least a few cycles, and that activation occurs while the clock is low. The design applies a `synthesis_keep` directive to the AND gates on the right side, which ensures there are no simultaneous toggles on the input of the `clk_out` OR gate.

Note: Switching from clock A to clock B requires that clock A continue to operate for at least a few cycles. If clock A stops immediately, the design sticks. The select signals are implemented as a "one-hot" control in this example, but you can use other encoding if you prefer. The input side logic is asynchronous and is not critical. This design can tolerate extreme glitching during the switch process.

Example 42. Verilog HDL Clock Multiplexing Design to Avoid Glitches

This example works with Verilog-2001.

```
module clock_mux (clk,clk_select,clk_out);

    parameter num_clocks = 4;

    input [num_clocks-1:0] clk;
    input [num_clocks-1:0] clk_select; // one hot
    output clk_out;

    genvar i;

    reg [num_clocks-1:0] ena_r0;
    reg [num_clocks-1:0] ena_r1;
    reg [num_clocks-1:0] ena_r2;
    wire [num_clocks-1:0] qualified_sel;

    // A look-up-table (LUT) can glitch when multiple inputs
    // change simultaneously. Use the keep attribute to
    // insert a hard logic cell buffer and prevent
    // the unrelated clocks from appearing on the same LUT.

    wire [num_clocks-1:0] gated_clks /* synthesis keep */;

    initial begin
        ena_r0 = 0;
        ena_r1 = 0;
        ena_r2 = 0;
    end

    generate
        for (i=0; i<num_clocks; i=i+1)
        begin : lp0
            wire [num_clocks-1:0] tmp_mask;
            assign tmp_mask = {num_clocks{1'b1}} ^ (1 << i);

            assign qualified_sel[i] = clk_select[i] & (~|(ena_r2 & tmp_mask));

            always @(posedge clk[i]) begin
                ena_r0[i] <= qualified_sel[i];
                ena_r1[i] <= ena_r0[i];
            end

            always @(negedge clk[i]) begin
                ena_r2[i] <= ena_r1[i];
            end

            assign gated_clks[i] = clk[i] & ena_r2[i];
        end
    endgenerate

    // These will not exhibit simultaneous toggle by construction
```

```
    assign clk_out = |gated_clks;

endmodule
```

1.6.3. Adder Trees

Structuring adder trees appropriately to match your targeted Intel FPGA device architecture can provide significant improvements in your design's efficiency and performance.

A good example of an application using a large adder tree is a finite impulse response (FIR) correlator. Using a pipelined binary or ternary adder tree appropriately can greatly improve the quality of your results.

1.6.3.1. Architectures with 6-Input LUTs in Adaptive Logic Modules

In Intel FPGA device families with 6-input LUT in their basic logic structure, ALMs can simultaneously add three bits. Take advantage of this feature by restructuring your code for better performance.

Although code targeting 4-input LUT architectures compiles successfully for 6-input LUT devices, the implementation can be inefficient. For example, to take advantage of the 6-input adaptive ALUT, you must rewrite large pipelined binary adder trees designed for 4-input LUT architectures. By restructuring the tree as a ternary tree, the design becomes much more efficient, significantly improving density utilization.

Example 43. Verilog HDL Pipelined Ternary Tree

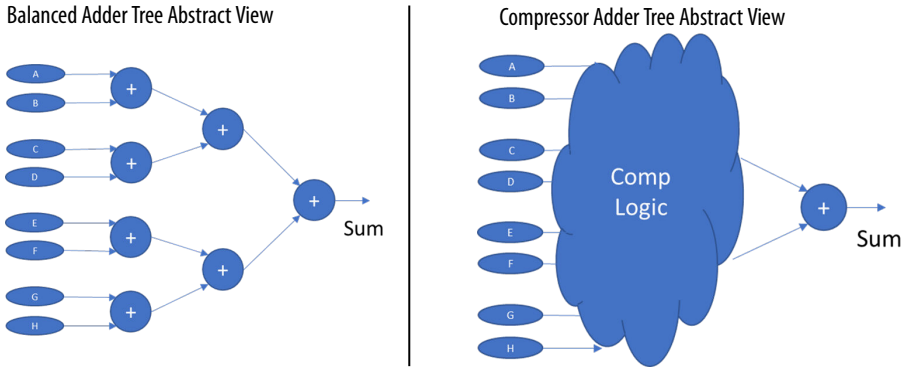
The example shows a pipelined adder, but partitioning your addition operations can help you achieve better results in non-pipelined adders as well. If your design is not pipelined, a ternary tree provides much better performance than a binary tree. For example, depending on your synthesis tool, the HDL code `sum = (A + B + C) + (D + E)` is more likely to create the optimal implementation of a 3-input adder for `A + B + C` followed by a 3-input adder for `sum1 + D + E` than the code without the parentheses. If you do not add the parentheses, the synthesis tool may partition the addition in a way that is not optimal for the architecture.

```
module ternary_adder_tree (a, b, c, d, e, clk, out);
    parameter width = 16;
    input [width-1:0] a, b, c, d, e;
    input    clk;
    output [width-1:0] out;

    wire [width-1:0] sum1, sum2;
    reg [width-1:0] sumreg1, sumreg2;
    // registers

    always @ (posedge clk)
        begin
            sumreg1 <= sum1;
            sumreg2 <= sum2;
        end

    // 3-bit additions
    assign sum1 = a + b + c;
    assign sum2 = sumreg1 + d + e;
    assign out = sumreg2;
endmodule
```

1.6.3.2. Change Adder Tree Styles

Because ALMs can implement functions of up to six inputs, you can improve the performance of certain designs by using a compressor implementation for adder trees, rather than the default balanced binary tree implementation. The expected downside tradeoff of the compressor implementation is the use of more ALM logic resources. However the overall logic depth is lower, and the final timing characteristics improve.

Figure 5. Balanced Binary Versus Compressor Style Adder Trees



For designs that may benefit, you can apply the **Use Compressor Implementation** (`USE_COMPRESSOR_IMPLEMENTATION`) global, entity, or instance assignment to specify whether the Compiler synthesizes adder trees as balanced binary trees, or as compressor style trees.

You can specify this assignment in the Assignment Editor, or with the following assignment in the `.qsf`.

```
set_instance_assignment -name USE_COMPRESSOR_IMPLEMENTATION ALWAYS -to <foo>
```

The following options are available for this assignment:

Table 2. Use Compressor Implementation Assignment Options

Option	Description
Always	The Compiler always synthesizes all adder trees with this assignment as compressor style trees. There is a limit of at least 2 non-constant operands before this triggers (otherwise synthesis implements a binary add or a pure-LUT implementation depending on size).
Never	The Compiler never synthesizes the assigned adder tree as a compressor. The Compiler synthesizes the adder as either a balanced binary tree, or if sufficiently small, in pure LUTs.
Auto	This setting currently behaves the same as the **Never** setting. The Compiler synthesizes the adder as either a balanced binary tree, or if sufficiently small, in pure LUTs. This setting never uses compressor style adder trees.

1.6.4. State Machine HDL Guidelines

Synthesis tools can recognize and encode Verilog HDL and VHDL state machines during synthesis. This section presents guidelines to secure the best results when you use state machines.

Synthesis tools that can recognize a piece of code as a state machine can perform optimizations that improve the design area and performance. For example, the tool can recode the state variables to improve the quality of results, or optimize other parts of the design through known properties of state machines.

To achieve the best results, synthesis tools often use one-hot encoding for FPGA devices and minimal-bit encoding for CPLD devices, although the choice of implementation can vary for different state machines and different devices. Refer to the synthesis tool documentation for techniques to control the encoding of state machines.

To ensure proper recognition and inference of state machines and to improve the quality of results, observe the following guidelines for both Verilog HDL and VHDL:

- Assign default values to outputs derived from the state machine so that synthesis does not generate unwanted latches.
- Separate state machine logic from all arithmetic functions and datapaths, including assigning output values.
- For designs in which more than one state perform the same operation, define the operation outside the state machine, and direct the output logic of the state machine to use this value.
- Ensure a defined power-up state with a simple asynchronous or synchronous `reset`. In designs where the state machine contains more elaborate `reset` logic, such as both an asynchronous `reset` and an asynchronous load, the Intel Quartus Prime software infers regular logic rather than a state machine.

If a state machine enters an illegal state due to a problem with the device, the design likely ceases to function correctly until the next `reset` of the state machine. Synthesis tools do not provide for this situation by default. The same issue applies to any other registers if there is some fault in the system. A `default` or `when others` clause does not affect this operation, assuming that the design never deliberately enters this state. Synthesis tools remove any logic generated by a default state if it is not reachable by normal state machine operation.

Many synthesis tools (including Intel Quartus Prime synthesis) have an option to implement a safe state machine. The Intel Quartus Prime software inserts extra logic to detect illegal states and force the state machine's transition to the `reset` state. Safe state machines are useful when the state machine can enter an illegal state, for example, when a state machine has control inputs that originate in another clock domain, such as the control logic for a dual-clock FIFO.

This option protects state machines by forcing them into the `reset` state. All other registers in the design are not protected this way. As a best practice for designs with asynchronous inputs, use a synchronization register chain instead of relying on the safe state machine option.

1.6.4.1. State Machine Power-Up

In Intel Stratix 10 devices, registers do not necessarily power-up in the same clock cycle if they are not in the same sector. This fact can cause issues with state machines if the state machine enters an undefined state.

One-hot encoded state machines are especially susceptible to this issue, as the number of undefined states is large compared to the number of legal states. Retiming also increases the risk of this issue because when state registers retime across logic or routing, it becomes more likely that the different state registers of one state machine are in different sectors.

To mitigate this risk, the Compiler automatically uses **Safe State Machine** for any state machine of 6 or less states for Intel Stratix 10 designs. This **Safe State Machine** setting forces the state machines back into the reset state if they enter an undefined state. The Compiler does not automatically use **Safe State Machine** for state machines of more than 6 states, or for Intel Arria 10 or Intel Cyclone 10 GX devices, because the effect on the quality of results can be significant.

1.6.4.2. Verilog HDL State Machines

To ensure proper recognition and inference of Verilog HDL state machines, observe the following additional Verilog HDL guidelines.

Refer to your synthesis tool documentation for specific coding recommendations. If the synthesis tool doesn't recognize and infer the state machine, the tool implements the state machine as regular logic gates and registers, and the state machine doesn't appear as a state machine in the **Analysis & Synthesis** section of the Intel Quartus Prime Compilation Report. In this case, Intel Quartus Prime synthesis does not perform any optimizations specific to state machines.

- If you are using the SystemVerilog standard, use enumerated types to describe state machines.
- Represent the states in a state machine with the parameter data types in Verilog-1995 and Verilog-2001, and use the parameters to make state assignments. This parameter implementation makes the state machine easier to read and reduces the risk of errors during coding.
- Do not directly use integer values for state variables, such as `next_state <= 0`. However, using an integer does not prevent inference in the Intel Quartus Prime software.
- Intel Quartus Prime software doesn't infer a state machine if the state transition logic uses arithmetic similar to the following example:

```
case (state)
    0: begin
        if (ena) next_state <= state + 2;
        else next_state <= state + 1;
        end
    1: begin
    ...
endcase
```

- Intel Quartus Prime software doesn't infer a state machine if the state variable is an output.
- Intel Quartus Prime software doesn't infer a state machine for signed variables.

1.6.4.2.1. Verilog-2001 State Machine Coding Example

The following module `verilog_fsm` is an example of a typical Verilog HDL state machine implementation. This state machine has five states.

The asynchronous reset sets the variable `state` to `state_0`. The sum of `in_1` and `in_2` is an output of the state machine in `state_1` and `state_2`. The difference (`in_1` – `in_2`) is also used in `state_1` and `state_2`. The temporary variables `tmp_out_0` and `tmp_out_1` store the sum and the difference of `in_1` and `in_2`. Using these temporary variables in the various states of the state machine ensures proper resource sharing between the mutually exclusive states.

Example 44. Verilog-2001 State Machine

```
module verilog_fsm (clk, reset, in_1, in_2, out);
    input clk, reset;
    input [3:0] in_1, in_2;
    output [4:0] out;
    parameter state_0 = 3'b000;
    parameter state_1 = 3'b001;
    parameter state_2 = 3'b010;
    parameter state_3 = 3'b011;
    parameter state_4 = 3'b100;

    reg [4:0] tmp_out_0, tmp_out_1, tmp_out_2;
    reg [2:0] state, next_state;

    always @ (posedge clk or posedge reset)
    begin
        if (reset)
            state <= state_0;
        else
            state <= next_state;
    end
    always @ (*)
    begin
        tmp_out_0 = in_1 + in_2;
        tmp_out_1 = in_1 - in_2;
        case (state)
            state_0: begin
               tmp_out_2 = in_1 + 5'b00001;
               next_state = state_1;
            end
            state_1: begin
                if (in_1 < in_2) begin
                    next_state = state_2;
                    tmp_out_2 = tmp_out_0;
                end
                else begin
                    next_state = state_3;
                    tmp_out_2 = tmp_out_1;
                end
            end
            state_2: begin
                tmp_out_2 = tmp_out_0 - 5'b00001;
                next_state = state_3;
            end
            state_3: begin
                tmp_out_2 = tmp_out_1 + 5'b00001;
                next_state = state_0;
            end
            state_4:begin
                tmp_out_2 = in_2 + 5'b00001;
                next_state = state_0;
            end
            default:begin
                tmp_out_2 = 5'b00000;
                next_state = state_0;
            end
        endcase
```

```
    end
    assign out = tmp_out_2;
endmodule
```

You can achieve an equivalent implementation of this state machine by using `‘define` instead of the `parameter` data type, as follows:

```
‘define state_0 3'b000
‘define state_1 3'b001
‘define state_2 3'b010
‘define state_3 3'b011
‘define state_4 3'b100
```

In this case, you assign `` `state_x `` instead of `state_x` to `state` and `next_state`, for example:

```
next_state <= ‘state_3;
```

Note: Although Intel supports the `‘define` construct, use the `parameter` data type, because it preserves the state names throughout synthesis.

1.6.4.2.2. SystemVerilog State Machine Coding Example

Use the following coding style to describe state machines in SystemVerilog.

Example 45. SystemVerilog State Machine Using Enumerated Types

The module `enum_fsm` is an example of a SystemVerilog state machine implementation that uses enumerated types.

In Intel Quartus Prime Pro Edition synthesis, the enumerated type that defines the states for the state machine must be of an unsigned integer type. If you do not specify the enumerated type as `int unsigned`, synthesis uses a signed `int` type by default. In this case, the Intel Quartus Prime software synthesizes the design, but does not infer or optimize the logic as a state machine.

```
module enum_fsm (input clk, reset, input int data[3:0], output int o);
enum int unsigned { S0 = 0, S1 = 2, S2 = 4, S3 = 8 } state, next_state;
always_comb begin : next_state_logic
      next_state = S0;
      case(state)
        S0: next_state = S1;
        S1: next_state = S2;
        S2: next_state = S3;
        S3: next_state = S3;
      endcase
end
always_comb begin
      case(state)
         S0: o = data[3];
         S1: o = data[2];
         S2: o = data[1];
         S3: o = data[0];
      endcase
end
always_ff@(posedge clk or negedge reset) begin
      if(~reset)
         state <= S0;
      else
         state <= next_state;
end
endmodule
```

1.6.4.3. VHDL State Machines

To ensure proper recognition and inference of VHDL state machines, represent the different states with enumerated types, and use the corresponding types to make state assignments.

This implementation makes the state machine easier to read, and reduces the risk of errors during coding. If your RTL does not represent states with an enumerated type, Intel Quartus Prime synthesis (and other synthesis tools) do not recognize the state machine. Instead, synthesis implements the state machine as regular logic gates and registers. Consequently, and the state machine does not appear in the state machine list of the Intel Quartus Prime Compilation Report, **Analysis & Synthesis** section. Moreover, Intel Quartus Prime synthesis does not perform any of the optimizations that are specific to state machines.

1.6.4.3.1. VHDL State Machine Coding Example

The following state machine has five states. The asynchronous reset sets the variable `state` to `state_0`.

The sum of `in1` and `in2` is an output of the state machine in `state_1` and `state_2`. The difference (`in1 - in2`) is also used in `state_1` and `state_2`. The temporary variables `tmp_out_0` and `tmp_out_1` store the sum and the difference of `in1` and `in2`. Using these temporary variables in the various states of the state machine ensures proper resource sharing between the mutually exclusive states.

Example 46. VHDL State Machine

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY vhdl_fsm IS
   PORT(
      clk: IN STD_LOGIC;
      reset: IN STD_LOGIC;
      in1: IN UNSIGNED(4 downto 0);
      in2: IN UNSIGNED(4 downto 0);
      out_1: OUT UNSIGNED(4 downto 0)
      );
END vhdl_fsm;
ARCHITECTURE rtl OF vhdl_fsm IS
   TYPE Tstate IS (state_0, state_1, state_2, state_3, state_4);
   SIGNAL state: Tstate;
   SIGNAL next_state: Tstate;
BEGIN
   PROCESS(clk, reset)
   BEGIN
      IF reset = '1' THEN
            state <=state_0;
      ELSIF rising_edge(clk) THEN
            state <= next_state;
      END IF;
   END PROCESS;
PROCESS (state, in1, in2)
      VARIABLE tmp_out_0: UNSIGNED (4 downto 0);
      VARIABLE tmp_out_1: UNSIGNED (4 downto 0);
   BEGIN
      tmp_out_0 := in1 + in2;
      tmp_out_1 := in1 - in2;
      CASE state IS
         WHEN state_0 =>
            out_1 <= in1;
```

```
            next_state <= state_1;
         WHEN state_1 =>
            IF (in1 < in2) then
               next_state <= state_2;
               out_1 <= tmp_out_0;
            ELSE
               next_state <= state_3;
               out_1 <= tmp_out_1;
            END IF;
         WHEN state_2 =>
            IF (in1 < "0100") then
               out_1 <= tmp_out_0;
            ELSE
               out_1 <= tmp_out_1;
            END IF;
               next_state <= state_3;
         WHEN state_3 =>
               out_1 <= "11111";
               next_state <= state_4;
         WHEN state_4 =>
               out_1 <= in2;
               next_state <= state_0;
         WHEN OTHERS =>
               out_1 <= "00000";
               next_state <= state_0;
      END CASE;
   END PROCESS;
END rtl;
```

1.6.5. Multiplexer HDL Guidelines

Multiplexers form a large portion of the logic utilization in many FPGA designs. By optimizing your multiplexer logic, you ensure the most efficient implementation.

This section addresses common problems and provides design guidelines to achieve optimal resource utilization for multiplexer designs. The section also describes various types of multiplexers, and how they are implemented.

For more information, refer to the *Advanced Synthesis Cookbook*.

1.6.5.1. Intel Quartus Prime Software Option for Multiplexer Restructuring

Intel Quartus Prime Pro Edition synthesis provides the **Restructure Multiplexers** logic option that extracts and optimizes buses of multiplexers during synthesis. The default **Auto** for this option setting uses the optimization whenever beneficial for your design. You can turn the option on or off specifically to have more control over use.

Even with this Intel Quartus Prime-specific option turned on, it is beneficial to understand how your coding style can be interpreted by your synthesis tool, and avoid the situations that can cause problems in your design.

1.6.5.2. Multiplexer Types

This section addresses how Intel Quartus Prime synthesis creates multiplexers from various types of HDL code.

State machines, CASE statements, and IF statements are all common sources of multiplexer logic in designs. These HDL structures create different types of multiplexers, including binary multiplexers, selector multiplexers, and priority multiplexers.

The first step toward optimizing multiplexer structures for best results is to understand how Intel Quartus Prime infers and implements multiplexers from HDL code.

1.6.5.2.1. Binary Multiplexers

Binary multiplexers select inputs based on binary-encoded selection bits.

Device families featuring 6-input look up tables (LUTs) are perfectly suited for 4:1 multiplexer building blocks (4 data and 2 select inputs). The extended input mode facilitates implementing 8:1 blocks, and the fractured mode handles residual 2:1 multiplexer pairs.

Example 47. Verilog HDL Binary-Encoded Multiplexers

```
case (sel)
   2'b00: z = a;
   2'b01: z = b;
   2'b10: z = c;
   2'b11: z = d;
endcase
```

1.6.5.2.2. Selector Multiplexers

Selector multiplexers have a separate select line for each data input. The select lines for the multiplexer are one-hot encoded. Intel Quartus Prime commonly builds selector multiplexers as a tree of AND and OR gates.

Even though the implementation of a tree-shaped, N-input selector multiplexer is slightly less efficient than a binary multiplexer, in many cases the select signal is the output of a decoder. Intel Quartus Prime synthesis combines the selector and decoder into a binary multiplexer.

Example 48. Verilog HDL One-Hot-Encoded CASE Statement

```
case (sel)
   4'b0001: z = a;
   4'b0010: z = b;
   4'b0100: z = c;
   4'b1000: z = d;
   default: z = 1'bx;
endcase
```

1.6.5.2.3. Priority Multiplexers

In priority multiplexers, the select logic implies a priority. The options to select the correct item must be checked in a specific order based on signal priority.

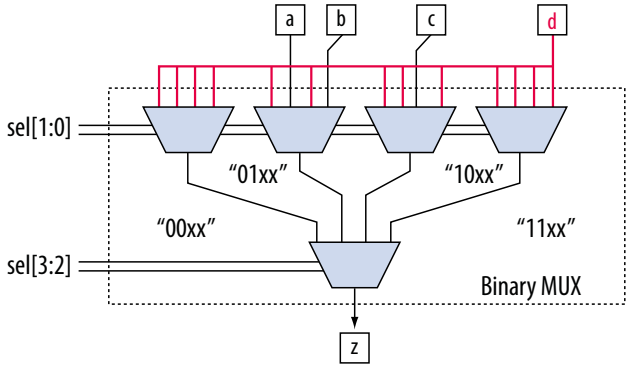
Synthesis tools commonly infer these structures from `IF`, `ELSE`, `WHEN`, `SELECT`, and `?:` statements in VHDL or Verilog HDL.

Example 49. VHDL IF Statement Implying Priority

The multiplexers form a chain, evaluating each condition or select bit sequentially.

```
IF cond1 THEN z <= a;
ELSIF cond2 THEN z <= b;
ELSIF cond3 THEN z <= c;
ELSE z <= d;
END IF;
```

Figure 6. Priority Multiplexer Implementation of an IF Statement



Depending on the number of multiplexers in the chain, the timing delay through this chain can become large, especially for device families with 4-input LUTs.

To improve the timing delay through the multiplexer, avoid priority multiplexers if priority is not required. If the order of the choices is not important to the design, use a CASE statement to implement a binary or selector multiplexer instead of a priority multiplexer. If delay through the structure is important in a multiplexed design requiring priority, consider recoding the design to reduce the number of logic levels to minimize delay, especially along your critical paths.

1.6.5.3. Implicit Defaults in IF Statements

IF statements in Verilog HDL and VHDL can simplify expressing conditions that do not easily lend themselves to a CASE-type approach. However, IF statements can result in complex multiplexer trees that are not easy for synthesis tools to optimize. In particular, all IF statements have an ELSE condition, even when not specified in the code. These implicit defaults can cause additional complexity in multiplexed designs.

You can simplify multiplexed logic and remove unneeded defaults with multiple methods. The optimal method is recoding the design, so the logic takes the structure of a 4:1 CASE statement. Alternatively, if priority is important, you can restructure the code to reduce default cases and flatten the multiplexer. Examine whether the default "ELSE IF" conditions are don't care cases. You can add a default ELSE statement to make the behavior explicit. Avoid unnecessary default conditions in the multiplexer logic to reduce the complexity and logic utilization that the design implementation requires.

1.6.5.4. default or OTHERS CASE Assignment

To fully specify the cases in a CASE statement, include a default (Verilog HDL) or OTHERS (VHDL) assignment.

This assignment is especially important in one-hot encoding schemes where many combinations of the select lines are unused. Specifying a case for the unused select line combinations gives the synthesis tool information about how to synthesize these cases, and is required by the Verilog HDL and VHDL language specifications.

For some designs you do not need to consider the outcome in the unused cases, because these cases are unreachable. For these types of designs, you can specify any value for the `default` or OTHERS assignment. However, the assignment value you choose can have a large effect on the logic utilization required to implement the design.

To obtain best results, explicitly define invalid CASE selections with a separate `default` or OTHERS statement, instead of combining the invalid cases with one of the defined cases.

If the value in the invalid cases is not important, specify those cases explicitly by assigning the X (don't care) logic value instead of choosing another value. This assignment allows your synthesis tool to perform the best area optimizations.

1.6.6. Cyclic Redundancy Check Functions

CRC computations are used heavily by communications protocols and storage devices to detect any corruption of data. These functions are highly effective; there is a very low probability that corrupted data can pass a 32-bit CRC check

CRC functions typically use wide XOR gates to compare the data. The way synthesis tools flatten and factor these XOR gates to implement the logic in FPGA LUTs can greatly impact the area and performance results for the design. XOR gates have a cancellation property that creates an exceptionally large number of reasonable factoring combinations, so synthesis tools cannot always choose the best result by default.

The 6-input ALUT has a significant advantage over 4-input LUTs for these designs. When properly synthesized, CRC processing designs can run at high speeds in devices with 6-input ALUTs.

The following guidelines help you improve the quality of results for CRC designs in Intel FPGA devices.

1.6.6.1. If Performance is Important, Optimize for Speed

To minimize area and depth of levels of logic, synthesis tools flatten XOR gates.

By default, Intel Quartus Prime Pro Edition synthesis targets area optimization for XOR gates. Therefore, for more focus on depth reduction, set the synthesis optimization technique to speed.

Note: Flattening for depth sometimes causes a significant increase in area.

1.6.6.2. Use Separate CRC Blocks Instead of Cascaded Stages

Some designs optimize CRC to use cascaded stages (for example, four stages of 8 bits). In such designs, Intel Quartus Prime synthesis uses intermediate calculations (such as the calculations after 8, 24, or 32 bits) depending on the data width.

This design is not optimal for FPGA devices. The XOR cancellations that Intel Quartus Prime synthesis performs in CRC designs mean that the function does not require all the intermediate calculations to determine the final result. Therefore, forcing the use of intermediate calculations increases the area required to implement the function, as

well as increasing the logic depth because of the cascading. It is typically better to create full separate CRC blocks for each data width that you require in the design, and then multiplex them together to choose the appropriate mode at a given time

1.6.6.3. Use Separate CRC Blocks Instead of Allowing Blocks to Merge

Synthesis tools often attempt to optimize CRC designs by sharing resources and extracting duplicates in two different CRC blocks because of the factoring options in the XOR logic.

CRC logic allows significant reductions, but this works best when the Compiler optimizes CRC function separately. Check for duplicate extraction behavior if for designs with different CRC functions that are driven by common data signals or that feed the same destination signals.

For designs with poor quality results that have two CRC functions sharing logic you can ensure that the blocks are synthesized independently with one of the following methods:

- Define each CRC block as a separate design partition in a hierarchical compilation design flow.
- Synthesize each CRC block as a separate project in a third-party synthesis tool and then write a separate Verilog Quartus Mapping (`.vqm`) or EDIF netlist file for each.

1.6.6.4. Take Advantage of Latency if Available

If your design can use more than one cycle to implement the CRC functionality, adding registers and retiming the design can help reduce area, improve performance, and reduce power utilization.

If your synthesis tool offers a retiming feature (such as the Intel Quartus Prime software **Perform gate-level register retiming** option), you can insert an extra bank of registers at the input and allow the retiming feature to move the registers for better results. You can also build the CRC unit half as wide and alternate between halves of the data in each clock cycle.

1.6.6.5. Save Power by Disabling CRC Blocks When Not in Use

CRC designs are heavy consumers of dynamic power because the logic toggles whenever there is a change in the design.

To save power, use clock enables to disable the CRC function for every clock cycle that the logic is not required. Some designs don't check the CRC results for a few clock cycles while other logic is performing. It is valuable to disable the CRC function even for this short amount of time.

1.6.6.6. Initialize the Device with the Synchronous Load (sload) Signal

CRC designs often require the data to be initialized to 1's before operation. In devices that support the `sload` signal, you can use this signal to set all registers in the design to 1's before operation.

To enable the `sload` signal, follow the coding guidelines in this chapter. After compilation you can check the register equations in the Chip Planner to ensure that the signal behaves as expected.

If you must force a register implementation using an `sload` signal, refer to *Designing with Low-Level Primitives User Guide* to see how you can use low-level device primitives.

Related Information

Secondary Register Control Signals Such as Clear and Clock Enable on page 40

1.6.7. Comparator HDL Guidelines

This section provides information about the different types of implementations available for comparators (<, >, or ==), and provides suggestions on how you can code the design to encourage a specific implementation. Synthesis tools, including Intel Quartus Prime Pro Edition synthesis, use device and context-specific implementation rules, and select the best one for the design.

Synthesis tools implement the == comparator in general logic cells and the < comparison in either the carry chain or general logic cells. In devices with 6-input ALUTs, the carry chain can compare up to three bits per cell. Carry chain implementation tends to be faster than general logic on standalone benchmark test cases, but can result in lower performance on larger designs due to increased restrictions on the Fitter. The area requirement is similar for most input patterns. The synthesis tools select an appropriate implementation based on the input pattern.

You can guide the Intel Quartus Prime Synthesis engine by choosing specific coding styles. To select a carry chain implementation explicitly, rephrase the comparison in terms of addition.

For example, the following coding style allows the synthesis tool to select the implementation, which is most likely using general logic cells in modern device families:

```
wire [6:0] a,b;
wire alb = a<b;
```

In the following coding style, the synthesis tool uses a carry chain (except for a few cases, such as when the chain is very short, or the signals a and b minimize to the same signal):

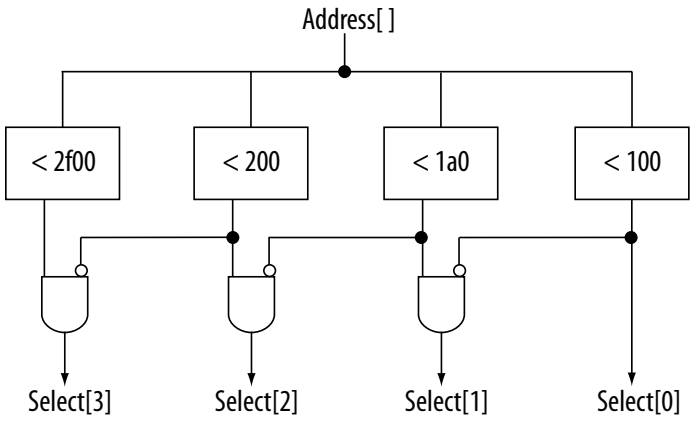
```
wire [6:0] a,b;
wire [7:0] tmp = a - b;
wire alb = tmp[7]
```

This second coding style uses the top bit of the `tmp` signal, which is 1 in two's complement logic if *a* is less than *b*, because the subtraction *a* - *b* results in a negative number.

If you have any information about the range of the input, you can use “don’t care” values to optimize the design. This information is not available to the synthesis tool, so specific hand implementation of the logic can reduce the device area required to implement the comparator.

The following logic structure, which occurs frequently in address decoders, allows you to check whether a bus value is within a constant range with a small amount of logic area:

Figure 7. Example Logic Structure for Using Comparators to Check a Bus Value Range



1.6.8. Counter HDL Guidelines

The Intel Quartus Prime synthesis engine implements counters in HDL code as an adder followed by registers, and makes available register control signals such as enable (`ena`), synchronous clear (`sclr`), and synchronous load (`sload`). For best area utilization, ensure that the up and down control or controls are expressed in terms of one addition operator, instead of two separate addition operators.

If you use the following coding style, your synthesis engine may implement two separate carry chains for addition:

```
out <= count_up ? out + 1 : out - 1;
```

For simple designs, the synthesis engine identifies this coding style and optimizes the logic. However, in complex designs, or designs with preserve pragmas, the Compiler cannot optimize all logic, so more careful coding becomes necessary.

The following coding style requires only one adder along with some other logic:

```
out <= out + (count_up ? 1 : -1);
```

This style makes more efficient use of resources and area, since it uses only one carry chain adder, and the –1 constant logic is implemented in the LUT before the adder.

1.7. Designing with Low-Level Primitives

Low-level HDL design is the practice of using low-level primitives and assignments to dictate a particular hardware implementation for a piece of logic. Low-level primitives are small architectural building blocks that assist you in creating your design.

With the Intel Quartus Prime software, you can use low-level HDL design techniques to force a specific hardware implementation that can help you achieve better resource utilization or faster timing results.

Note: Using low-level primitives is an optional advanced technique to help with specific design challenges. For many designs, synthesizing generic HDL source code and Intel FPGA IP cores give you the best results.

Low-level primitives allow you to use the following types of coding techniques:

- Instantiate the logic cell or LCELL primitive to prevent Intel Quartus Prime Pro Edition synthesis from performing optimizations across a logic cell
- Instantiate registers with specific control signals using DFF primitives
- Specify the creation of LUT functions by identifying the LUT boundaries
- Use I/O buffers to specify I/O standards, current strengths, and other I/O assignments
- Use I/O buffers to specify differential pin names in your HDL code, instead of using the automatically-generated negative pin name for each pair

For details about and examples of using these types of assignments, refer to the *Designing with Low-Level Primitives User Guide*.

1.8. Cross-Module Referencing (XMR) in HDL Code

Cross Module Referencing (XMR), also known as hierarchical reference, is enabled by default. It is a mechanism built into Verilog, SystemVerilog, and VHDL to globally reference nets in any hierarchy across modules, which means you can refer to any net of a particular module in a different module (irrespective of the hierarchy) directly without going through the ports. Hence, XMR can be a downward reference or an upward reference.

Note: XMR also works in mixed language designs.

You can use XMR to read from nets or write to a net in different hierarchies. XMR is helpful in debugging and verification, for example:

- Override or force any signal. For more information, refer to Using force Statements in HDL Code on page 64.
- Write cover points for functional coverage.
- Tap into any signal from anywhere in the entire design.

Consider the following hierarchy within the instance `top` of module `a`:

```
module a
  net x
  instance p of module b
    net x
    instance m of module d
      net x
  instance q of module c
    net x
    instance n of module e
      net x
  instance r of module b
     net x
     instance m of module d
       net x
```

In the above scenario, all modules consist of a net named x. By using full-path-based XMR, you can globally reference each net x anywhere within the hierarchy, as follows:

- top.x
- top.p.x
- top.p.m.x
- top.q.x
- top.q.n.x
- top.r.x
- top.r.m

XMR starts with searching within the current module. Then, it hierarchically searches downwards through child instances. If XMR is unresolved, it searches one step above in the hierarchy (parent module) and hierarchically downwards. It keeps going further in the hierarchy until it gets resolved. To avoid unexpected behavior, Intel recommends always using the hierarchy path where possible.

XMR Use Case Examples

Example 50. XMR of Signals in Higher Modules from Lower Modules

In the following example, the signal d in the sub module is assigned to the value a from the top module:

```
module top (input a, input b, output c, output d);
  sub inst1 (.a(a), .b(b), .c(c), .d(d));
endmodule

module sub (input a, input b, output c, output d);
  assign c = a & b;
  assign d = top.a;
endmodule
```

Example 51. XMR of Signals in Lower Modules from Higher Modules

In the following example, the sub module's value a assigns the output value d, given the complete path in the top module:

```
module top (input a, input b, output c, output d);
  sub inst1 (.a(a), .b(b), .c(c));
  assign d = inst1.a;
endmodule
module sub (input a, input b, output c);
  assign c = a & b;
endmodule
```

Example 52. XMR of Signals in generate Block

Consider the following example with a generate block where you write the value to temp at the top module and read the out2 value from the top module:

```
module top (input [3:0] in1, in2, input clk, output [3:0] out1, out2);
  generate
    begin:blk1 sub inst (in1, clk, out1, temp);
  end:blk1
  endgenerate
```

```
//XMR read
assign out2 = top.blk1.inst.temp;
endmodule
```

Example 53. XMR From Inside an `always` Block

Consider the following example of XMR of signals within the `always_comb` construct, where the output value `d` in the `top` module is assigned from the `sub` module value `a`:

```
module top (input logic a, input logic b, output logic c, output logic d);
  sub inst1 (.a(a), .b(b), .c(c));
  always_comb d = inst1.a;

endmodule

module sub (input logic a, input logic b, output logic c);
  assign c = b & a;
endmodule
```

Limitations of XMR

The following are some limitations of XMR in the Intel Quartus Prime software:

- XMR must be within the same design partition. For example, if there are partitions A and B, then in partition B, there cannot be any XMR to anything in partition A and vice versa.

  Note: The same partition requirement applies for global signals that are specified in a package and can be used in any module.
- Multiple-driven nets are not supported. So, you cannot use XMR to drive a net already driven by another signal.
- You can use XMR only in Verilog, SystemVerilog, and VHDL. It does not work for Text Design File (TDF) or Block Design File (BDF).

  Note: Intel does not recommend using XMR on SystemVerilog interfaces since they are prone to errors.
- You cannot use XMR to refer to signals inside an Intel FPGA IP core or a Signal Tap partition that is automatically created during synthesis.

Related Information

Synplify Pro for Microsemi Edition Language Support Reference Manual

1.9. Using `force` Statements in HDL Code

`force` statement in SystemVerilog is a continuous procedural assignment on a net or a variable. Applying a `force` statement to a net or variable overrides all other drivers to that net or variable. In simulation, you can use a `force` statement in conjunction with a `release` statement. However, Intel Quartus Prime software synthesis supports using only the `force` statement to override the drivers of a net (gate outputs, module outputs, and continuous assignments) and previous assignments made on a particular net or a net bus.

Note: Synthesis supports using a `force` statement only inside an initial block.

Examples of force Statements in Synthesis

The following are some examples of force statements that the Intel Quartus Prime software synthesis supports:

Example 54. Using a force Statement to Set Counter enable to 0

The following is an example of how you can use a force statement to tie the en port of the counter instance u1 to logic 0:

```
module top(clk, rst, enable, dout);
  input clk, rst, enable;
  output [3:0] dout;
  counter u1(.clk(clk), .reset(rst), .en(enable), .q(dout));

  initial  begin
    force u1.en = 1'b0
  end
endmodule
```

You can observe that the force statement overrides the other driver of the en port, which is the enable port of the top module.

Example 55. Using a force Statement to Change Connections

The following example shows how you can use a force statement to change the connections in the design:

```
module top(input [3:0] din, din1, output logic [3:0] dout, dout1, input clk,
rst);
  dff i0(.din(din), .dout(dout), .clk(clk), .rst(rst) );
  dff i1(.din(din1), .dout(dout1), .clk(clk), .rst(rst) );
endmodule

module top_modified(input [3:0] din, din1, output logic [3:0] dout, dout1, input
clk, rst);
  top i_top(.*);
  initial
  begin
    force i_top.i1.din = i_top.din;
  end
endmodule
```

In this example, the design's top module instantiates two instances of the dff module. din and din1 ports of the top module drive the din port of i0 and i1 instances.

Suppose you want to change the connections in the top module without changing the RTL inside the top module. In this situation, you can use a force statement within a wrapper module (top_modified), which becomes the new top module. In the new top module, use the force statement to modify the connections in the top module such that the din port of both i0 and i1 instances is driven by the same din port of the top module. The force statement uses a cross-module reference (XMR) to access signals in a hierarchy below it. For more information about XMR, refer to Cross-Module Referencing (XMR) in HDL Code on page 62.

Note: For this example, instead of creating a wrapper `top_modified` that instantiates the `top` module, you can also create a secondary top-level entity and make the `force` assignment in it, as shown in the following:

```
module secondary_top(input [3:0] din, din1, output logic [3:0] dout, dout1,
input clk, rst);
initial begin
  force top.i1.din = top.din;
end
endmodule
```

1.10. Recommended HDL Coding Styles Revision History

The following revisions history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2023.10.02	23.1	• Made minor updates to code snippets in *Using force Statements in HDL Code* and *Cross-Module Referencing (XMR) in HDL Code*.
2023.04.03	23.1	• Updated product family name to "Intel Agilex 7."
2022.09.26	22.3	• Added *Using force Statements in HDL Code*. • Added *Cross-Module Referencing (XMR) in HDL Code*.
2021.10.04	21.3	• Added new *Inferring FIFOs in HDL Code* topic and linked to *FIFO Intel FPGA IP User Guide*. • Added new *Dual Clock FIFO Example in Verilog HDL* topic. • Added new *Dual Clock FIFO Timing Constraints* topic.
2021.06.21	21.2	• Updated *Inferring Shift Registers in HDL Code* for Intel Stratix 10 and Intel Agilex 7 devices. • Updated wording of *Controlling RAM Inference and Implementation* for clarity.
2019.09.30	19.3	• Updated Simple Dual-Port Synchronous RAM with Byte Enable examples. • Updated True Dual-Port Synchronous RAM examples. • Updated Verilog HDL Single-Bit Wide Shift Register example from 64 to 69 bits. • Updated VHDL Single-Bit Wide Shift Register example from 67 to 69 bits. • Updated Verilog HDL 8-Bit Wide Shift Register with Evenly Spaced Taps from 64 to 254 bits.
2018.09.24	18.1	• Added "State Machine Power-Up" topic. • Updated "Designing with Low-Level Primitives" to remove support for carry and cascade chains using CARRY, CARRY_SUM, and CASCADE primitives. • Renamed topic: "Use the Device Synchronous Load (sload) Signal to Initialize" to "Initialize the Device with the Synchronous Load (sload) Signal"
2017.11.06	17.1	• Described new no_ram synthesis attribute.
2017.05.08	17.0	• Updated example: Verilog HDL Multiply-Accumulator • Updated information about use of safe state machine. • Revised Check Read-During-Write Behavior. • Revised Controlling RAM Inference and Implementation. • Revised Single-Clock Synchronous RAM with Old Data Read-During-Write Behavior. • Revised Single-Clock Synchronous RAM with New Data Read-During-Write Behavior.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Updated and moved template for VHDL Single-Clock Simple Dual Port Synchronous RAM with New Data Read-During-Write Behavior. • Revised Inferring ROM Functions from HDL Code. • Removed example: VHDL 8-Bit Wide, 64-Bit Long Shift Register with Evenly Spaced Taps. • Removed example: Verilog HDL D-Type Flipflop (Register) With ena, aclr, and aload Control Signals • Removed example: VHDL D-Type Flipflop (Register) With ena, aclr, and aload Control Signals • Added example: Verilog D-type Flipflop bus with Secondary Signals • Removed references to 4-input LUT-based devices. • Removed references to Integrated Synthesis. • Created example: Avoid this VHDL Coding Style.
2016.10.31	16.1	• Provided corrected Verilog HDL Pipelined Binary Tree and Ternary Tree examples. • Implemented Intel rebranding.
2016.05.03	16.0	• Added information about use of safe state machine. • Updated example code templates with latest coding styles.
2015.11.02	15.1	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
2015.05.04	15.0	Added information and reference about ramstyle attribute for sift register inference.
2014.12.15	14.1	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
2014.08.18	14.0.a10	• Added recommendation to use register pipelining to obtain high performance in DSP designs.
2014.06.30	14.0	Removed obsolete MegaWizard Plug-In Manager support.
November 2013	13.1	Removed HardCopy device support.
June 2012	12.0	• Revised section on inserting Altera templates. • Code update for Example 11-51. • Minor corrections and updates.
November 2011	11.1	• Updated document template. • Minor updates and corrections.
December 2010	10.1	• Changed to new document template. • Updated Unintentional Latch Generation content. • Code update for Example 11-18.
July 2010	10.0	• Added support for mixed-width RAM • Updated support for no_rw_check for inferring RAM blocks • Added support for byte-enable
November 2009	9.1	• Updated support for Controlling Inference and Implementation in Device RAM Blocks • Updated support for Shift Registers

continued...

Document Version	Intel Quartus Prime Version	Changes
March 2009	9.0	• Corrected and updated several examples • Added support for Arria II GX devices • Other minor changes to chapter
November 2008	8.1	Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0	Updates for the Intel Quartus Prime software version 8.0 release, including: • Added information to "RAM • Functions—Inferring ALTSYNCRAM and ALTDPRAM Megafunctions from HDL Code" on page 6–13 • Added information to "Avoid Unsupported Reset and Control Conditions" on page 6–14 • Added information to "Check Read-During-Write Behavior" on page 6–16 • Added two new examples to "ROM Functions—Inferring ALTSYNCRAM and LPM_ROM Megafunctions from HDL Code" on page 6–28: Example 6–24 and Example 6–25 • Added new section: "Clock Multiplexing" on page 6–46 • Added hyperlinks to references within the chapter • Minor editorial updates

2. Recommended Design Practices

This chapter provides design recommendations for Intel FPGA devices.

Current FPGA applications have reached the complexity and performance requirements of ASICs. In the development of complex system designs, design practices have an enormous impact on the timing performance, logic utilization, and system reliability of a device. Well-coded designs behave in a predictable and reliable manner even when retargeted to different families or speed grades. Good design practices also aid in successful design migration between FPGA and ASIC implementations for prototyping and production.

For optimal performance, reliability, and faster time-to-market when designing with Intel FPGA devices, you should adhere to the following guidelines:

- Understand the impact of synchronous design practices
- Follow recommended design techniques, including hierarchical design partitioning, and timing closure guidelines
- Take advantage of the architectural features in the targeted device

2.1. Following Synchronous FPGA Design Practices

The first step in good design methodology is to understand the implications of your design practices and techniques. This section outlines the benefits of optimal synchronous design practices and the hazards involved in other approaches.

Good synchronous design practices can help you meet your design goals consistently. Problems with other design techniques can include reliance on propagation delays in a device, which can lead to race conditions, incomplete timing analysis, and possible glitches.

In a synchronous design, a clock signal triggers every event. If you ensure that all the timing requirements of the registers are met, a synchronous design behaves in a predictable and reliable manner for all process, voltage, and temperature (PVT) conditions. You can easily migrate synchronous designs to different device families or speed grades.

2.1.1. Implementing Synchronous Designs

In a synchronous design, the clock signal controls the activities of all inputs and outputs.

On every active edge of the clock (usually the rising edge), the data inputs of registers are sampled and transferred to outputs. Following an active clock edge, the outputs of combinational logic feeding the data inputs of registers change values. This change triggers a period of instability due to propagation delays through the logic as the

signals go through several transitions and finally settle to new values. Changes that occur on data inputs of registers do not affect the values of their outputs until after the next active clock edge.

Because the internal circuitry of registers isolates data outputs from inputs, instability in the combinational logic does not affect the operation of the design if you meet the following timing requirements:

- Before an active clock edge, you must ensure that the data input has been stable for at least the setup time of the register.
- After an active clock edge, you must ensure that the data input remains stable for at least the hold time of the register.

  When you specify all your clock frequencies and other timing requirements, the Intel Quartus Prime Timing Analyzer reports actual hardware requirements for the setup times (t_{SU}) and hold times (t_H) for every pin in your design. By meeting these external pin requirements and following synchronous design techniques, you ensure that you satisfy the setup and hold times for all registers in your device.

  Tip: To meet setup and hold time requirements on all input pins, any inputs to combinational logic that feed a register should have a synchronous relationship with the clock of the register. If signals are asynchronous, you can register the signals at the inputs of the device to help prevent a violation of the required setup and hold times.

  When you violate the setup or hold time of a register, you might oscillate the output, or set the output to an intermediate voltage level between the high and low levels called a metastable state. In this unstable state, small perturbations such as noise in power rails can cause the register to assume either the high or low voltage level, resulting in an unpredictable valid state. Various undesirable effects can occur, including increased propagation delays and incorrect output states. In some cases, the output can even oscillate between the two valid states for a relatively long period of time.

2.1.2. Asynchronous Design Hazards

Asynchronous design techniques, such as ripple counters or pulse generators, can work as "short cuts" to save device resources. However, asynchronous techniques have inherent problems. For example, relying on propagation delays can result in incomplete timing constraints and possible glitches and spikes, because propagation delay varies with temperature and voltage fluctuations.

Asynchronous design structures that depend on the relative propagation delays can present race conditions. Race conditions arise when the order of signal changes affect the output of the logic. The same logic design can have varying timing delays with each compilation, depending on placement and routing. The number of possible variations make it impossible to determine the timing delay associated with a particular block of logic. As devices become faster due to process improvements, delays in asynchronous designs may decrease, resulting in designs that do not function as expected. Relying on a particular delay also makes asynchronous designs difficult to migrate to other architectures, devices, or speed grades.

The timing of asynchronous design structures is often difficult or impossible to model with timing assignments and constraints. If you do not have complete or accurate timing constraints, the timing-driven algorithms that synthesis and place-and-route tools use may not be able to perform the best optimizations, and the reported results may be incomplete.

Additionally, asynchronous design structures can generate glitches, which are pulses that are very short compared to clock periods. Combinational logic is the main cause of glitches. When the inputs to the combinational logic change, the outputs exhibit several glitches before settling to their new values. Glitches can propagate through combinational logic, leading to incorrect values on the outputs in asynchronous designs. In synchronous designs, glitches on register's data inputs have no negative consequences, because data processing waits until the next clock edge.

2.2. HDL Design Guidelines

When designing with HDL code, consider how synthesis tools interpret different HDL design techniques and what results to expect.

Design style can affect logic utilization and timing performance, as well as the design's reliability. This section describes basic design techniques that ensure optimal synthesis results for designs that target Intel FPGA devices while avoiding common causes of unreliability and instability. As a best practice, consider potential problems when designing combinational logic, and pay attention to clocking schemes so that the design maintains synchronous functionality and avoids timing issues.

2.2.1. Considerations for the Intel Hyperflex FPGA Architecture

The Intel Hyperflex FPGA architecture and the Hyper-Retimer require a review of the best design practices to achieve the highest clock rates possible.

While most common techniques of high-speed design apply to designing for the Intel Hyperflex architecture, you must use some new approaches to achieve the highest performance. Follow these general RTL design guidelines to enable the Hyper-Retimer to optimize design performance:

- Design in a way that facilitates register retiming by the Hyper-Retimer.
- Use a latency-insensitive design that supports the addition of pipeline stages at clock domain boundaries, top-level I/Os, and at the boundaries of functional blocks.
- Restructure RTL to avoid performance-limiting loops.

For more information about best design practices targeting Intel Stratix 10 devices, refer to the *Intel Stratix 10 High-Performance Design Handbook*.

Related Information

Intel® Hyperflex™ Architecture High-Performance Design Handbook

2.2.2. Optimizing Combinational Logic

Combinational logic structures consist of logic functions that depend only on the current state of the inputs. In Intel FPGAs, these functions are implemented in the look-up tables (LUTs) with either logic elements (LEs) or adaptive logic modules (ALMs).

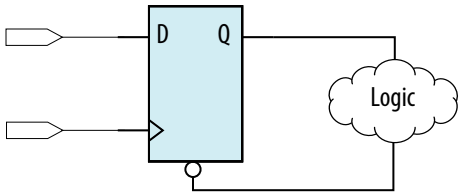
For cases where combinational logic feeds registers, the register control signals can implement part of the logic function to save LUT resources. By following the recommendations in this section, you can improve the reliability of your combinational design.

2.2.2.1. Avoid Combinational Loops

Combinational loops are among the most common causes of instability and unreliability in digital designs. Combinational loops generally violate synchronous design principles by establishing a direct feedback loop that contains no registers.

Avoid combinational loops whenever possible. In a synchronous design, feedback loops should include registers. For example, a combinational loop occurs when the left-hand side of an arithmetic expression also appears on the right-hand side in HDL code. A combinational loop also occurs when you feed back the output of a register to an asynchronous pin of the same register through combinational logic.

Figure 8. Combinational Loop Through Asynchronous Control Pin



Tip: Use recovery and removal analysis to perform timing analysis on asynchronous ports, such as `clear` or `reset` in the Intel Quartus Prime software.

Combinational loops are inherently high-risk design structures for the following reasons:

- Combinational loop behavior generally depends on relative propagation delays through the logic involved in the loop. As discussed, propagation delays can change, which means the behavior of the loop is unpredictable.
- In many design tools, combinational loops can cause endless computation loops . Most tools break open combinational loops to process the design. The various tools used in the design flow may open a given loop differently, and process it in a way inconsistent with the original design intent.

2.2.2.2. Avoid Unintended Latch Inference

Avoid using latches to ensure that you can completely analyze the timing performance and reliability of your design. A latch is a small circuit with combinational feedback that holds a value until a new value is assigned. You can implement latches with the Intel Quartus Prime Text Editor or Block Editor.

A common mistake in HDL code is unintended latch inference; Intel Quartus Prime Synthesis issues a warning message if this occurs. Unlike other technologies, a latch in FPGA architecture is not significantly smaller than a register. However, the architecture is not optimized for latch implementation and latches generally have slower timing performance compared to equivalent registered circuitry.

Latches have a transparent mode in which data flows continuously from input to output. A positive latch is in transparent mode when the enable signal is high (low for a negative latch). In transparent mode, glitches on the input can pass through to the output because of the direct path created. This presents significant complexity for timing analysis. Typical latch schemes use multiple enable phases to prevent long transparent paths from occurring. However, timing analysis cannot identify these safe applications.

The Timing Analyzer analyzes latches as synchronous elements clocked on the falling edge of the positive latch signal by default. It allows you to treat latches as having nontransparent start and end points. Be aware that even an instantaneous transition through transparent mode can lead to glitch propagation. The Timing Analyzer cannot perform cycle-borrowing analysis.

Due to various timing complexities, latches have limited support in formal verification tools. Therefore, you should not rely on formal verification for a design that includes latches.

Related Information

Avoid Unintentional Latch Generation on page 42

2.2.2.3. Avoid Delay Chains in Clock Paths

Delays in PLD designs can change with each placement and routing cycle. Effects such as rise and fall time differences and on-chip variation mean that delay chains, especially those placed on clock paths, can cause significant problems in your design. Avoid using delay chains to prevent these kinds of problems.

You require delay chains when you use two or more consecutive nodes with a single fan-in and a single fan-out to cause delay. Inverters are often chained together to add delay. Delay chains are sometimes used to resolve race conditions created by other asynchronous design practices.

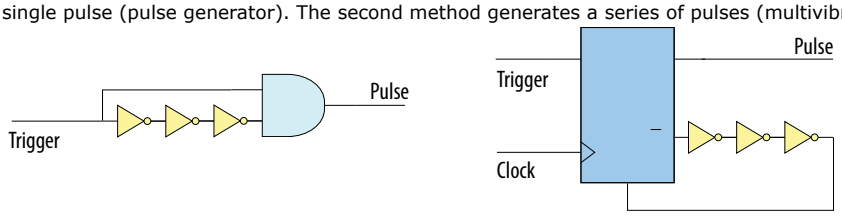
In some ASIC designs, delays are used for buffering signals as they are routed around the device. This functionality is not required in FPGA devices because the routing structure provides buffers throughout the device.

2.2.2.4. Use Synchronous Pulse Generators

Use synchronous techniques to design pulse generators.

Figure 9. Asynchronous Pulse Generators

The figure shows two methods for asynchronous pulse generation. The first method uses a delay chain to generate a single pulse (pulse generator). The second method generates a series of pulses (multivibrators).



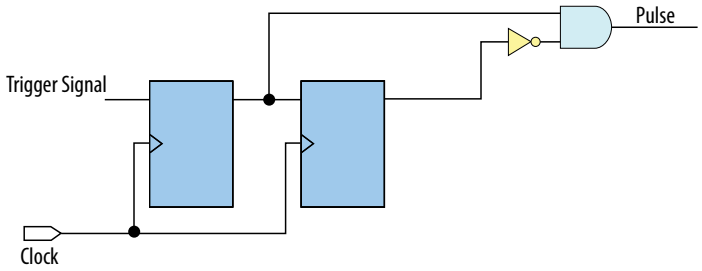
In the first method, a trigger signal feeds both inputs of a 2-input AND gate, and the design adds inverters to one of the inputs to create a delay chain. The width of the pulse depends on the time differences between the path that feeds the gate directly and the path that goes through the delay chain. This is the same mechanism responsible for the generation of glitches in combinational logic following a change of input values. This technique artificially increases the width of the glitch.

In the second method, a register's output drives its asynchronous reset signal through a delay chain. The register resets itself asynchronously after a certain delay. The Compiler can determine the pulse width only after placement and routing, when routing and propagation delays are known. You cannot reliably create a specific pulse

width when creating HDL code, and it cannot be set by EDA tools. The pulse may not be wide enough for the application under all PVT conditions. Also, the pulse width changes if you change to a different device. Additionally, verification is difficult because static timing analysis cannot verify the pulse width.

Multivibrators use a glitch generator to create pulses, together with a combinational loop that turns the circuit into an oscillator. This method creates additional problems because of the number of pulses involved. Additionally, when the structures generate multiple pulses, they also create a new artificial clock in the design that must be analyzed by design tools.

Figure 10. Recommended Synchronous Pulse-Generation Technique



The pulse width is always equal to the clock period. This pulse generator is predictable, can be verified with timing analysis, and is easily moved to other architectures, devices, or speed grades.

2.2.3. Optimizing Clocking Schemes

Like combinational logic, clocking schemes have a large effect on the performance and reliability of a design.

Intel recommends avoiding the use of internally generated clocks (other than PLLs) wherever possible because they can cause functional and timing problems in the design. If not carefully designed, clocks generated with or passing through combinational logic can introduce glitches that create functional problems, and the delay inherent in combinational logic can lead to timing problems. Refer to the sections listed below for information on some common scenarios of using combinational logic in a clock path (for example, clock mux) and design considerations to prevent unexpected failures.

Tip: Specify all clock relationships in the Intel Quartus Prime software to allow for the best timing-driven optimizations during fitting and to allow correct timing analysis. Use clock setting assignments on any derived or internal clocks to specify their relationship to the base clock.

Use global device-wide, low-skew dedicated routing for all internally-generated clocks, instead of routing clocks on regular routing lines.

Avoid data transfers between different clocks wherever possible. If you require a data transfer between different clocks, use FIFO circuitry. You can use the clock uncertainty features in the Intel Quartus Prime software to compensate for the variable delays between clock domains. Consider setting a clock setup uncertainty and clock hold uncertainty value of 10% to 15% of the clock delay.

The following sections provide specific examples and recommendations for avoiding clocking scheme problems:

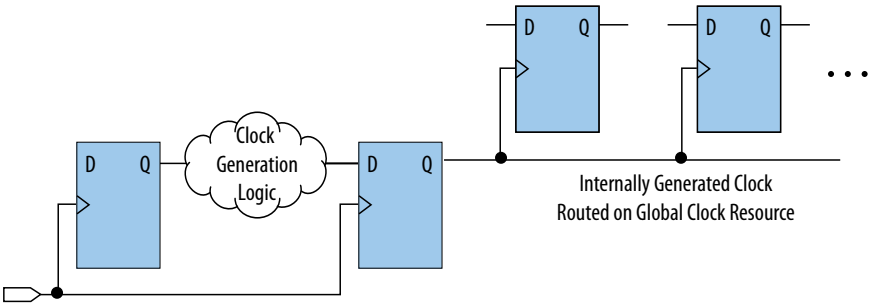
2.2.3.1. Register Combinational Logic Outputs

If you use the output from combinational logic as a clock signal or as an asynchronous reset signal, you can expect to see glitches in your design. In a synchronous design, glitches on data inputs of registers are normal events that have no consequences. However, a glitch or a spike on the clock input (or an asynchronous input) to a register can have significant consequences.

Narrow glitches can violate the register's minimum pulse width requirements. Setup and hold requirements might also be violated if the data input of the register changes when a glitch reaches the clock input. Even if the design does not violate timing requirements, the register output can change value unexpectedly and cause functional hazards elsewhere in the design.

To avoid these problems, you should always register the output of combinational logic before you use it as a clock signal.

Figure 11. Recommended Clock-Generation Technique



Registering the output of combinational logic ensures that glitches generated by the combinational logic are blocked at the data input of the register.

2.2.3.2. Avoid Asynchronous Clock Division

Designs often require clocks that you create by dividing a master clock. Most Intel FPGAs provide dedicated phase-locked loop (PLL) circuitry for clock division. Using dedicated PLL circuitry can help you avoid many of the problems that can be introduced by asynchronous clock division logic.

When you must use logic to divide a master clock, always use synchronous counters or state machines. Additionally, create your design so that registers always directly generate divided clock signals, and route the clock on global clock resources. To avoid glitches, do not decode the outputs of a counter or a state machine to generate clock signals.

2.2.3.3. Avoid Ripple Counters

To simplify verification, avoid ripple counters in your design. In the past, FPGA designers implemented ripple counters to divide clocks by a power of two because the counters are easy to design and may use fewer gates than their synchronous counterparts.

Ripple counters use cascaded registers, in which the output pin of one register feeds the clock pin of the register in the next stage. This cascading can cause problems because the counter creates a ripple clock at each stage. These ripple clocks must be handled properly during timing analysis, which can be difficult and may require you to make complicated timing assignments in your synthesis and placement and routing tools.

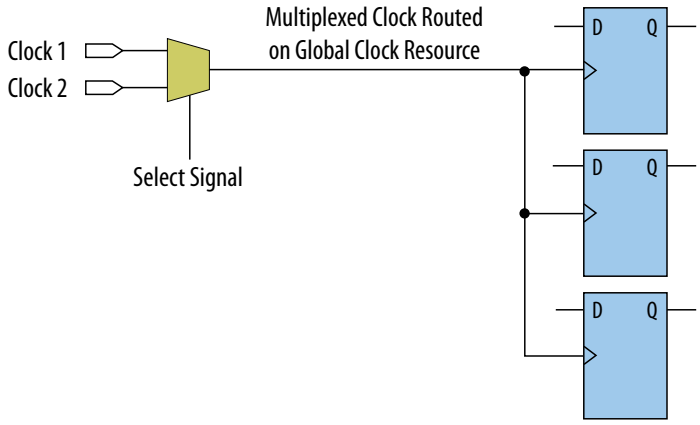
You can often use ripple clock structures to make ripple counters out of the smallest amount of logic possible. However, in all Intel devices supported by the Intel Quartus Prime software, using a ripple clock structure to reduce the amount of logic used for a counter is unnecessary because the device allows you to construct a counter using one logic element per counter bit. You should avoid using ripple counters completely.

2.2.3.4. Use Multiplexed Clocks

Use clock multiplexing to operate the same logic function with different clock sources. In these designs, multiplexing selects a clock source.

For example, telecommunications applications that deal with multiple frequency standards often use multiplexed clocks.

Figure 12. Multiplexing Logic and Clock Sources



Adding multiplexing logic to the clock signal can create the problems addressed in the previous sections, but requirements for multiplexed clocks vary widely, depending on the application. Clock multiplexing is acceptable when the clock signal uses global clock routing resources and if the following criteria are met:

- The clock multiplexing logic does not change after initial configuration
- The design uses multiplexing logic to select a clock for testing purposes
- Registers are always reset when the clock switches
- A temporarily incorrect response following clock switching has no negative consequences

If the design switches clocks in real time with no reset signal, and your design cannot tolerate a temporarily incorrect response, you must use a synchronous design so that there are no timing violations on the registers, no glitches on clock signals, and no race conditions or other logical problems. By default, the Intel Quartus Prime software optimizes and analyzes all possible paths through the multiplexer and between both internal clocks that may come from the multiplexer. This may lead to more restrictive

analysis than required if the multiplexer is always selecting one particular clock. If you do not require the more complete analysis, you can assign the output of the multiplexer as a base clock in the Intel Quartus Prime software, so that all register-to-register paths are analyzed using that clock.

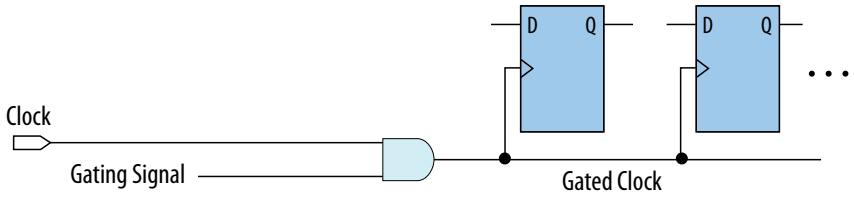
Tip: Use dedicated hardware to perform clock multiplexing when it is available, instead of using multiplexing logic. For example, you can use the clock-switchover feature or clock control block available in certain Intel FPGA devices. These dedicated hardware blocks ensure that you use global low-skew routing lines and avoid any possible hold time problems on the device due to logic delay on the clock line.

Note: For device-specific information about clocking structures, refer to the appropriate device data sheet or handbook.

2.2.3.5. Use Gated Clocks

Gated clocks turn a clock signal on and off using an enable signal that controls gating circuitry. When a clock is turned off, the corresponding clock domain is shut down and becomes functionally inactive.

Figure 13. Gated Clock



You can use gated clocks to reduce power consumption in some device architectures by effectively shutting down portions of a digital circuit when they are not in use. When a clock is gated, both the clock network and the registers driven by it stop toggling, thereby eliminating their contributions to power consumption. However, gated clocks are not part of a synchronous scheme and therefore can significantly increase the effort required for design implementation and verification. Gated clocks contribute to clock skew and make device migration difficult. These clocks are also sensitive to glitches, which can cause design failure.

Use dedicated hardware to perform clock gating rather than an AND or OR gate. For example, you can use the clock control block in newer Intel FPGA devices to shut down an entire clock network. Dedicated hardware blocks ensure that you use global routing with low skew, and avoid any possible hold time problems on the device due to logic delay on the clock line.

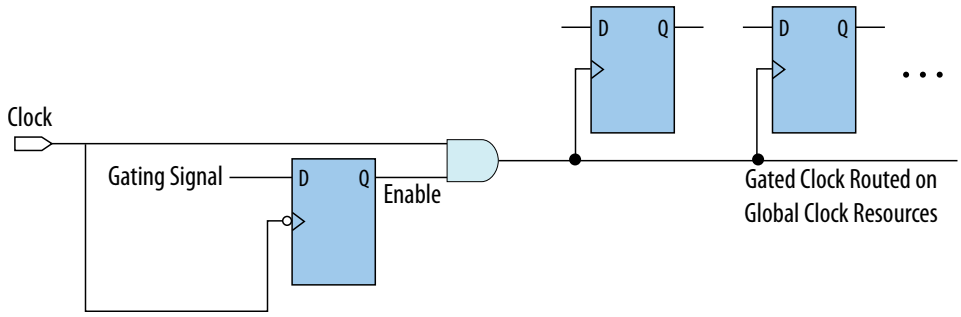
From a functional point of view, you can shut down a clock domain in a purely synchronous manner using a synchronous clock enable signal. However, when using a synchronous clock enable scheme, the clock network continues toggling. This practice does not reduce power consumption as much as gating the clock at the source does. In most cases, use a synchronous scheme.

2.2.3.5.1. Recommended Clock-Gating Methods

Use gated clocks only when your target application requires power reduction and gated clocks provide the required reduction in your device architecture. If you must use clocks gated by logic, follow a robust clock-gating methodology and ensure the gated clock signal uses dedicated global clock routing.

You can gate a clock signal at the source of the clock network, at each register, or somewhere in between. Since the clock network contributes to switching power consumption, gate the clock at the source whenever possible to shut down the entire clock network instead of further along.

Figure 14. Recommended Clock-Gating Technique for Clock Active on Rising Edge



To generate a gated clock with the recommended technique, use a register that triggers on the inactive edge of the clock. With this configuration, only one input of the gate changes at a time, preventing glitches or spikes on the output. If the clock is active on the rising edge, use an AND gate. Conversely, for a clock that is active on the falling edge, use an OR gate to gate the clock and register

Pay attention to the delay through the logic generating the enable signal, because the enable command must be ready in less than one-half the clock cycle. This might cause problems if the logic that generates the enable command is particularly complex, or if the duty cycle of the clock is severely unbalanced. However, careful management of the duty cycle and logic delay may be an acceptable solution when compared with problems created by other methods of gating clocks.

In the Timing Analyzer, ensure to apply a clock setting to the output of the AND gate. Otherwise, the timing analyzer might analyze the circuit using the clock path through the register as the longest clock path and the path that skips the register as the shortest clock path, resulting in artificial clock skew.

In certain cases, converting the gated clocks to clock enable pins may help reduce glitch and clock skew, and eventually produce a more accurate timing analysis. You can set the Intel Quartus Prime software to automatically convert gated clocks to clock enable pins by turning on the **Auto Gated Clock Conversion** option. The conversion applies to two types of gated clocking schemes: single-gated clock and cascaded-gated clock.

Related Information

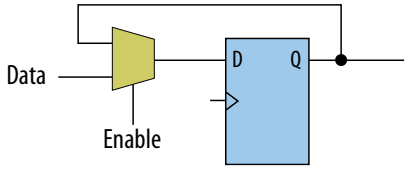
Auto Gated Clock Conversion logic option
In *Intel Quartus Prime Help*

2.2.3.6. Use Synchronous Clock Enables

To turn off a clock domain in a synchronous manner, use a synchronous clock enable signal. FPGAs efficiently support clock enable signals because there is a dedicated clock enable signal available on all device registers.

This scheme does not reduce power consumption as much as gating the clock at the source because the clock network keeps toggling, and performs the same function as a gated clock by disabling a set of registers. Insert a multiplexer in front of the data input of every register to either load new data, or copy the output of the register.

Figure 15. Synchronous Clock Enable



When designing for Intel Stratix 10 devices, consider that high fan-out clock enable signals can limit the performance achievable by the Hyper- Retimer. For specific recommendations, refer to the *Intel Stratix 10 High-Performance Design Handbook*.

Related Information

Intel® Hyperflex™ Architecture High-Performance Design Handbook

2.2.4. Optimizing Physical Implementation and Timing Closure

This section provides design and timing closure techniques for high speed or complex core logic designs with challenging timing requirements. These techniques may also be helpful for low or medium speed designs.

2.2.4.1. Planning Physical Implementation

When planning a design, consider the following elements of physical implementation:

- The number of unique clock domains and their relationships
- The amount of logic in each functional block
- The location and direction of data flow between blocks
- How data routes to the functional blocks between I/O interfaces

Interface-wide control or status signals may have competing or opposing constraints. For example, when a functional block's control or status signals interface with physical channels from both sides of the device. In such cases you must provide enough pipeline register stages to allow these signals to traverse the width of the device. In addition, you can structure the hierarchy of the design into separate logic modules for each side of the device. The side modules can generate and use registered control signals per side. This simplifies floorplanning, particularly in designs with transceivers, by placing per-side logic near the transceivers.

When adding register stages to pipeline control signals, turn off **Auto Shift Register Replacement** in the **Assignment Editor** (**Assignments ➤ Assignment Editor**) for each register as needed. By default, chains of registers can be converted to a RAM-based implementation based on performance and resource estimates. Since pipelining helps meet timing requirements over long distance, this assignment ensures that control signals are not converted.

2.2.4.2. Planning FPGA Resources

Your design requirements impact the use of FPGA resources. Plan functional blocks with appropriate global, regional, and dual-regional network signals in mind.

In general, after allocating the clocks in a design, use global networks for the highest fan-out control signals. When a global network signal distributes a high fan-out control signal, the global signal can drive logic anywhere in the device. Similarly, when using a regional network signal, the driven logic must be in one quadrant of the device, or half the device for a dual-regional network signal. Depending on data flow and physical locations of the data entry and exit between the I/Os and the device, restricting a functional block to a quadrant or half the device may not be practical for performance or resource requirements.

When floorplanning a design, consider the balance of different types of device resources, such as memory, logic, and DSP blocks in the main functional blocks. For example, if a design is memory intensive with a small amount of logic, it may be difficult to develop an effective floorplan. Logic that interfaces with the memory would have to spread across the chip to access the memory. In this case, it is important to use enough register stages in the data and control paths to allow signals to traverse the chip to access the physically disparate resources needed.

2.2.4.3. Optimizing for Timing Closure

To achieve timing closure for your design, you can enable compilation settings in the Intel Quartus Prime software, or you can directly modify your timing constraints.

Compilation Settings for Timing Closure

Note: Changes in project settings can significantly increase compilation time. You can view the performance gain versus runtime cost by reviewing the Fitter messages after design processing.

Table 3. Compilation Settings that Impact Timing Closure

Setting	Location	Effect on Timing Closure
Allow Register Duplication	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**	This technique is most useful where registers have high fan-out, or where the fan-out is in physically distant areas of the device. Review the netlist optimizations report and consider manually duplicating registers automatically added by physical synthesis. You can also locate the original and duplicate registers in the Chip Planner. Compare their locations, and if the fan-out is improved, modify the code and turn off register duplication to save compile time.
Prevent Register Retiming	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**	Useful if some combinatorial paths between registers exceed the timing goal while other paths fall short. If a design is already heavily pipelined, register retiming is less likely to provide significant performance gains, since there should not be significantly unbalanced levels of logic across pipeline stages.

Guidelines for Optimizing Timing Closure using Timing Constraints

Appropriate timing constraints are essential to achieving timing closure. Use the following general guidelines in applying timing constraints:

- Apply multicycle constraints in your design wherever single-cycle timing analysis is not necessary.
- Apply False Path constraints to all asynchronous clock domain crossings or resets in the design. This technique prevents overconstraining and the Fitter focuses only on critical paths to reduce compile time. However, overconstraining timing critical clock domains can sometimes provide better timing results and lower compile times than physical synthesis.
- Overconstrain rather than using physical synthesis when the slack improvement from physical synthesis is near zero. Overconstrain the frequency requirement on timing critical clock domains by using setup uncertainty.
- When evaluating the effect of constraint changes on performance and runtime, compile the design with at least three different seeds to determine the average performance and runtime effects. Different constraint combinations produce various results. Three samples or more establish a performance trend. Modify your constraints based on performance improvement or decline.
- Leave settings at the default value whenever possible. Increasing performance constraints can increase the compile time significantly. While those increases may be necessary to close timing on a design, using the default settings whenever possible minimizes compile time.

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Optimization

2.2.4.4. Optimizing Critical Timing Paths

To close timing in high speed designs, review paths with the largest timing failures. Correcting a single, large timing failure can result in a very significant timing improvement.

Review the register placement and routing paths by clicking **Tools ➤ Chip Planner**. Large timing failures on high fan-out control signals can be caused by any of the following conditions:

- Sub-optimal use of global networks
- Signals that traverse the chip on local routing without pipelining
- Failure to correct high fan-out by register duplication

For high-speed and high-bandwidth designs, optimize speed by reducing bus width and wire usage. To reduce wire usage, move the data as little as possible. For example, if a block of logic functions on a few bits of a word, store inactive bits in a FIFO or memory. Memory is cheaper and denser than registers, and reduces wire usage.

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Optimization

2.2.5. Optimizing Power Consumption

The total FPGA power consumption is comprised of I/O power, core static power, and core dynamic power. Knowledge of the relationship between these components is fundamental in calculating the overall total power consumption.

You can use various optimization techniques and tools to minimize power consumption when applied during FPGA design implementation. The Intel Quartus Prime software offers power-driven compilation features to fully optimize device power consumption. Power-driven compilation focuses on reducing your design's total power consumption using power-driven synthesis and power-driven placement and routing.

Related Information

Intel Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
In *Intel Quartus Prime Pro Edition User Guide: Power Analysis and Optimization*

2.2.6. Managing Design Metastability

In FPGA designs, synchronization of asynchronous signals can cause metastability. You can use the Intel Quartus Prime software to analyze the mean time between failures (MTBF) due to metastability. A high metastability MTBF indicates a more robust design.

Related Information

- Managing Metastability with the Intel Quartus Prime Software on page 121
- Intel Quartus Prime Pro Edition User Guide: Design Optimization

2.3. Use Clock and Register-Control Architectural Features

In addition to following general design guidelines, you must code your design with the device architecture in mind. FPGAs provide device-wide clocks and register control signals that can improve performance.

2.3.1. Use Global Reset Resources

ASIC designs may use local resets to avoid long routing delays. Take advantage of the device-wide asynchronous reset pin available on most FPGAs to eliminate these problems. This reset signal provides low-skew routing across the device.

The following are three types of resets used in synchronous circuits:

- Synchronous Reset
- Asynchronous Reset
- Synchronized Asynchronous Reset—preferred when designing an FPGA circuit

2.3.1.1. Use Synchronous Resets

The synchronous reset ensures that the circuit is fully synchronous. You can easily time the circuit with the Intel Quartus Prime Timing Analyzer.

Because clocks that are synchronous to each other launch and latch the reset signal, the data arrival and data required times are easily determined for proper slack analysis. The synchronous reset is easier to use with cycle-based simulators.

There are two methods by which a reset signal can reach a register; either by being gated in with the data input, or by using an LAB-wide control signal (`synclr`). If you use the first method, you risk adding an additional gate delay to the circuit to accommodate the reset signal, which causes increased data arrival times and negatively impacts setup slack. The second method relies on dedicated routing in the LAB to each register, but this is slower than an asynchronous reset to the same register.

Figure 16. Synchronous Reset

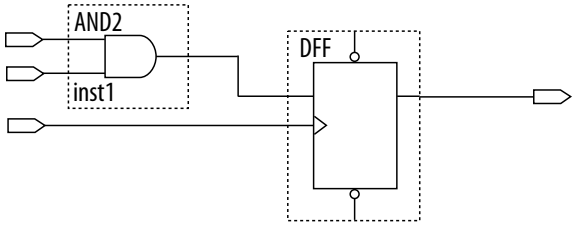
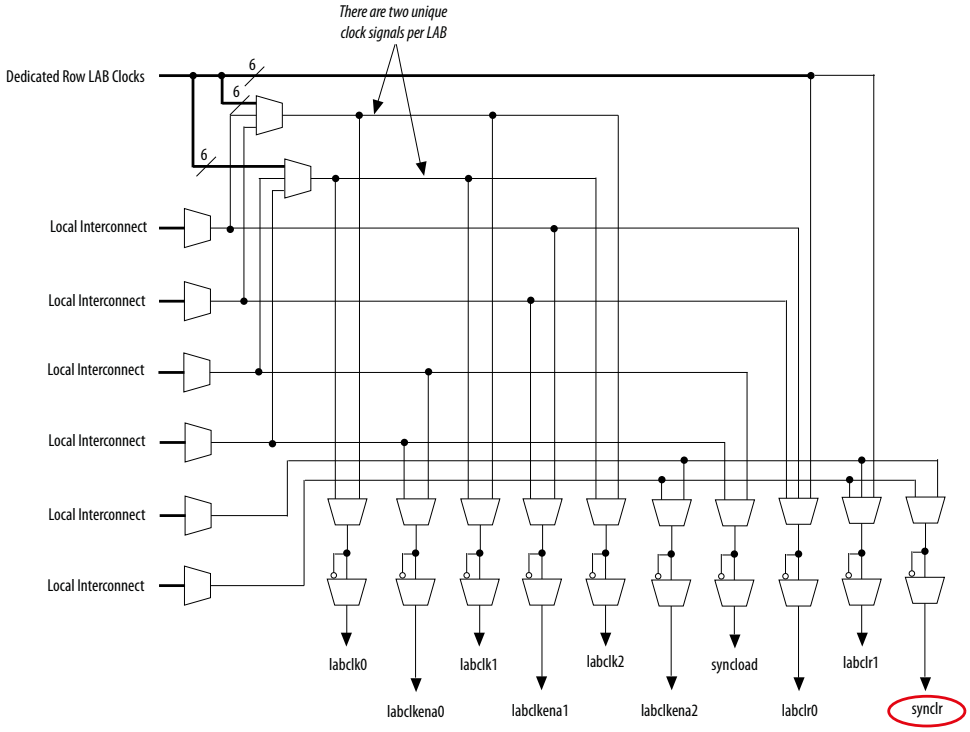
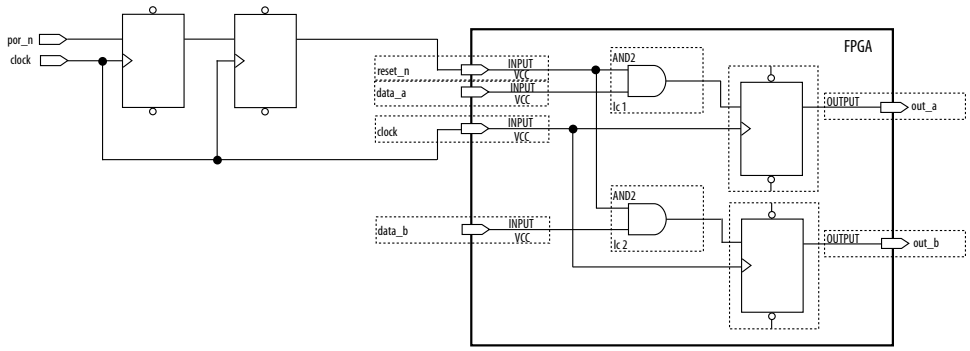


Figure 17. LAB-Wide Control Signals



Consider two types of synchronous resets when you examine the timing analysis of synchronous resets—externally synchronized resets and internally synchronized resets. Externally synchronized resets are synchronized to the clock domain outside the FPGA, and are not very common. A power-on asynchronous reset is dual-rank synchronized externally to the system clock and then brought into the FPGA. Inside the FPGA, gate this reset with the data input to the registers to implement a synchronous reset.

Figure 18. Externally Synchronized Reset



The following example shows the Verilog HDL equivalent of the schematic. When you use synchronous resets, the reset signal is not put in the sensitivity list.

The following example shows the necessary modifications that you should make to the internally synchronized reset.

Example 56. Verilog HDL Code for Externally Synchronized Reset

```
module sync_reset_ext (
         input   clock,
         input   reset_n,
         input   data_a,
         input   data_b,
         output  out_a,
         output  out_b
         );
reg      reg1, reg2
assign   out_a  = reg1;
assign   out_b  = reg2;
always @ (posedge clock)
begin
     if (!reset_n)
     begin
         reg1     <= 1'b0;
         reg2     <= 1'b0;
    end
    else
    begin
         reg1     <= data_a;
         reg2     <= data_b;
    end
end
endmodule     //  sync_reset_ext
```

The following example shows the constraints for the externally synchronous reset. Because the external reset is synchronous, you only need to constrain the `reset_n` signal as a normal input signal with `set_input_delay` constraint for `-max` and `-min`.

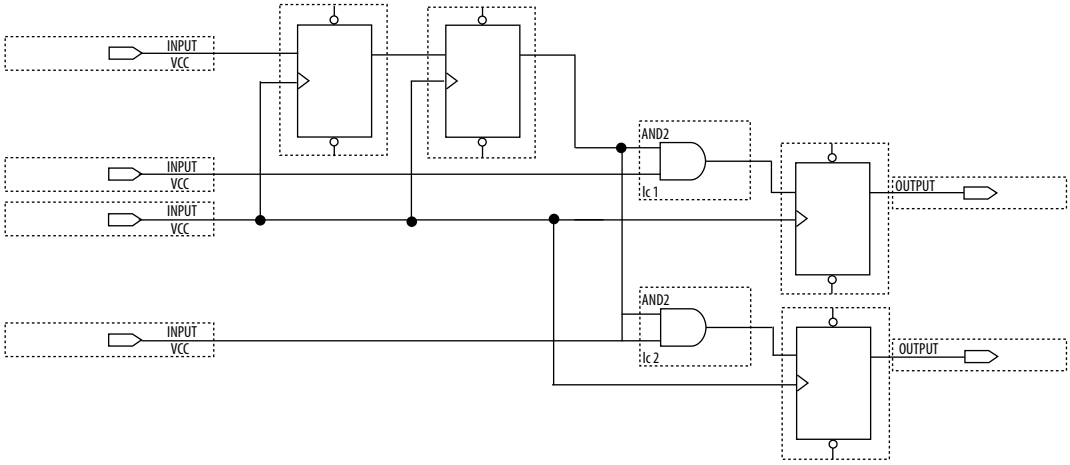
Example 57. SDC Constraints for Externally Synchronized Reset

```
# Input clock - 100 MHz
create_clock [get_ports {clock}] \
        -name {clock} \
        -period 10.0 \
```

```
        -waveform {0.0 5.0}
# Input constraints on low-active reset
# and data
set_input_delay 7.0 \
        -max \
        -clock [get_clocks {clock}] \
        [get_ports {reset_n data_a data_b}]
set_input_delay 1.0 \
        -min \
        -clock [get_clocks {clock}] \
        [get_ports {reset_n data_a data_b}]
```

More often, resets coming into the device are asynchronous, and must be synchronized internally before being sent to the registers.

Figure 19. Internally Synchronized Reset



The following example shows the Verilog HDL equivalent of the schematic. Only the clock edge is in the sensitivity list for a synchronous reset.

Example 58. Verilog HDL Code for Internally Synchronized Reset

```
module sync_reset (
        input clock,
        input reset_n,
        input data_a,
        input data_b,
        output out_a,
        output out_b
        );
reg     reg1, reg2
reg     reg3, reg4

assign     out_a = reg1;
assign     out_b = reg2;
assign     rst_n = reg4;

always @ (posedge clock)
begin
    if (!rst_n)
    begin
        reg1 <= 1'bo;
        reg2 <= 1'b0;
    end
    else
    begin
```

```
        reg1 <= data_a;
        reg2 <= data_b;
    end
end

always @ (posedge clock)
begin
    reg3 <= reset_n;
    reg4 <= reg3;
end
endmodule // sync_reset
```

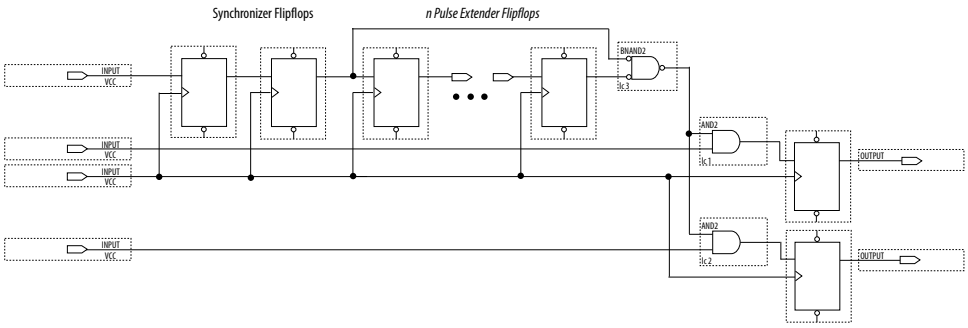
The SDC constraints are similar to the external synchronous reset, except that the input reset cannot be constrained because it is asynchronous. Cut the input path with a `set_false_path` statement to avoid these being considered as unconstrained paths.

Example 59. SDC Constraints for Internally Synchronized Reset

```
# Input clock - 100 MHz
create_clock [get_ports {clock}] \
        -name {clock} \
        -period 10.0 \
        -waveform {0.0 5.0}
# Input constraints on data
set_input_delay 7.0 \
        -max \
        -clock [get_clocks {clock}] \
        [get_ports {data_a data_b}]
set_input_delay 1.0 \
        -min \
        -clock [get_clocks {clock}] \
        [get_ports {data_a data_b}]
# Cut the asynchronous reset input
set_false_path \
        -from [get_ports {reset_n}] \
        -to [all_registers]
```

An issue with synchronous resets is their behavior with respect to short pulses (less than a period) on the asynchronous input to the synchronizer flipflops. This can be a disadvantage because the asynchronous reset requires a pulse width of at least one period wide to guarantee that it is captured by the first flipflop. However, this can also be viewed as an advantage in that this circuit increases noise immunity. Spurious pulses on the asynchronous input have a lower chance of being captured by the first flipflop, so the pulses do not trigger a synchronous reset. In some cases, you might want to increase the noise immunity further and reject any asynchronous input reset that is less than n periods wide to debounce an asynchronous input reset.

Figure 20. Internally Synchronized Reset with Pulse Extender



Junction dots indicate the number of stages. You can have more flipflops to get a wider pulse that spans more clock cycles.

Many designs have more than one clock signal. In these cases, use a separate reset synchronization circuit for each clock domain in the design. When you create synchronizers for PLL output clocks, these clock domains are not reset until you lock the PLL and the PLL output clocks are stable. If you use the reset to the PLL, this reset does not have to be synchronous with the input clock of the PLL. You can use an asynchronous reset for this. Using a reset to the PLL further delays the assertion of a synchronous reset to the PLL output clock domains when using internally synchronized resets.

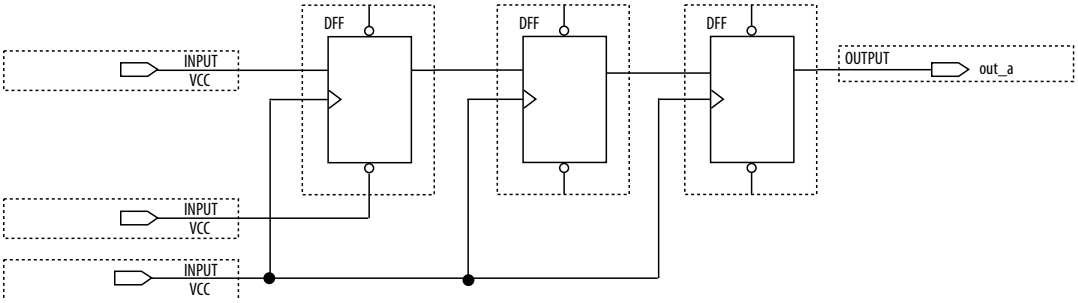
2.3.1.2. Using Asynchronous Resets

Asynchronous resets are the most common form of reset in circuit designs, as well as the easiest to implement. Typically, you can insert the asynchronous reset into the device, turn on the global buffer, and connect to the asynchronous reset pin of every register in the device.

This method is only advantageous under certain circumstances—you do not need to always reset the register. Unlike the synchronous reset, the asynchronous reset is not inserted in the datapath, and does not negatively impact the data arrival times between registers. Reset takes effect immediately, and as soon as the registers receive the reset pulse, the registers are reset. The asynchronous reset is not dependent on the clock.

However, when the reset is deasserted and does not pass the recovery (μt_{SU}) or removal (μt_H) time check (the Timing Analyzer recovery and removal analysis checks both times), the edge is said to have fallen into the metastability zone. Additional time is required to determine the correct state, and the delay can cause the setup time to fail to register downstream, leading to system failure. To avoid this, add a few follower registers after the register with the asynchronous reset and use the output of these registers in the design. Use the follower registers to synchronize the data to the clock to remove the metastability issues. You should place these registers close to each other in the device to keep the routing delays to a minimum, which decreases data arrival times and increases MTBF. Ensure that these follower registers themselves are not reset, but are initialized over a period of several clock cycles by “flushing out” their current or initial state.

Figure 21. Asynchronous Reset with Follower Registers



The following example shows the equivalent Verilog HDL code. The active edge of the reset is now in the sensitivity list for the procedural block, which infers a clock enable on the follower registers with the inverse of the reset signal tied to the clock enable. The follower registers should be in a separate procedural block as shown using non-blocking assignments.

Example 60. Verilog HDL Code of Asynchronous Reset with Follower Registers

```
module async_reset (
        input   clock,
        input   reset_n,
        input   data_a,
        output   out_a,
        );
reg     reg1, reg2, reg3;
assign  out_a  = reg3;
always @ (posedge clock, negedge reset_n)
begin
    if (!reset_n)
        reg1    <= 1'b0;
    else
        reg1    <= data_a;
end
always @ (posedge clock)
begin
    reg2    <= reg1;
    reg3    <= reg2;
end
endmodule  //  async_reset
```

You can easily constrain an asynchronous reset. By definition, asynchronous resets have a non-deterministic relationship to the clock domains of the registers they are resetting. Therefore, static timing analysis of these resets is not possible and you can use the `set_false_path` command to exclude the path from timing analysis. Because the relationship of the reset to the clock at the register is not known, you cannot run recovery and removal analysis in the Timing Analyzer for this path. Attempting to do so even without the false path statement results in no paths reported for recovery and removal.

Example 61. SDC Constraints for Asynchronous Reset

```
# Input clock - 100 MHz
create_clock [get_ports {clock}] \
        -name {clock} \
        -period 10.0 \
        -waveform {0.0 5.0}
```

```
# Input constraints on data
set_input_delay 7.0 \
        -max \
        -clock [get_clocks {clock}]\
        [get_ports {data_a}]
set_input_delay 1.0 \
        -min \
        -clock [get_clocks {clock}] \
        [get_ports {data_a}]
# Cut the asynchronous reset input
set_false_path \
        -from [get_ports {reset_n}] \
        -to [all_registers]
```

The asynchronous reset is susceptible to noise, and a noisy asynchronous reset can cause a spurious reset. You must ensure that the asynchronous reset is debounced and filtered. You can easily enter into a reset asynchronously, but releasing a reset asynchronously can lead to potential problems (also referred to as "reset removal") with metastability, including the hazards of unwanted situations with synchronous circuits involving feedback.

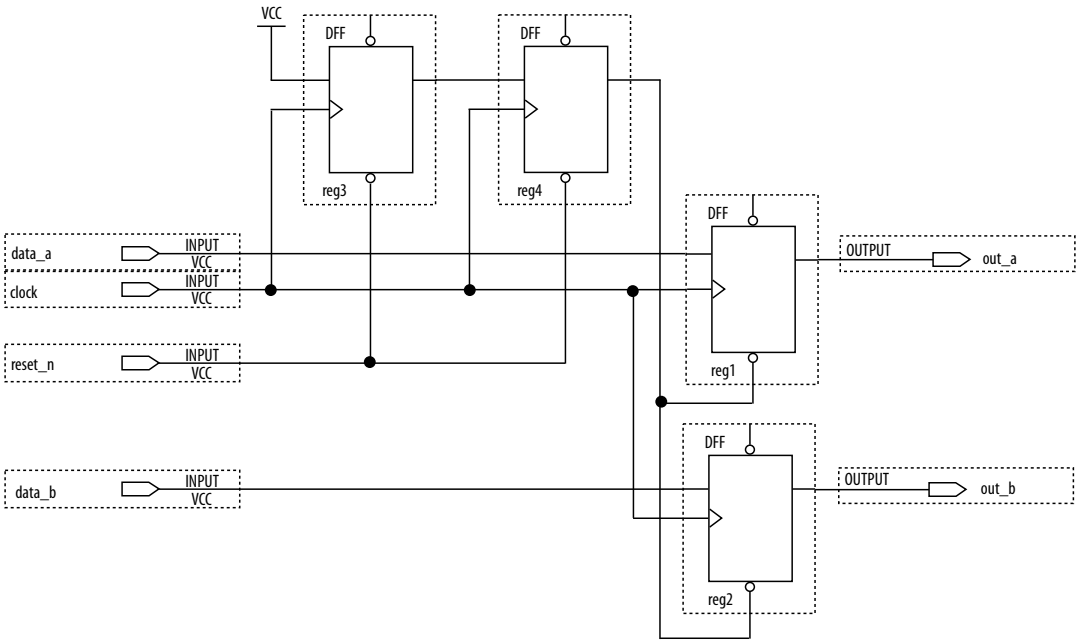
2.3.1.3. Use Synchronized Asynchronous Reset

To avoid potential problems associated with purely synchronous resets and purely asynchronous resets, you can use synchronized asynchronous resets. Synchronized asynchronous resets combine the advantages of synchronous and asynchronous resets.

These resets are asynchronously asserted and synchronously deasserted. This takes effect almost instantaneously, and ensures that no datapath for speed is involved. Also, the circuit is synchronous for timing analysis and is resistant to noise.

The following example shows a method for implementing the synchronized asynchronous reset. You should use synchronizer registers in a similar manner as synchronous resets. However, the asynchronous reset input is gated directly to the CLRN pin of the synchronizer registers and immediately asserts the resulting reset. When the reset is deasserted, logic "1" is clocked through the synchronizers to synchronously deassert the resulting reset.

Figure 22. Schematic of Synchronized Asynchronous Reset



The following example shows the equivalent Verilog HDL code. Use the active edge of the reset in the sensitivity list for the blocks.

Example 62. Verilog HDL Code for Synchronized Asynchronous Reset

```
module sync_async_reset (
        input    clock,
        input    reset_n,
        input    data_a,
        input    data_b,
        output   out_a,
        output   out_b
        );
reg     reg1, reg2;
reg     reg3, reg4;
assign  out_a    = reg1;
assign  out_b    = reg2;
assign  rst_n    = reg4;
always @ (posedge clock, negedge reset_n)
begin
    if (!reset_n)
    begin
       reg3     <= 1'b0;
       reg4     <= 1'b0;
    end
    else
    begin
       reg3     <= 1'b1;
       reg4     <= reg3;
    end
end
always @ (posedge clock, negedge rst_n)
begin
    if (!rst_n)
    begin
       reg1     <= 1'b0;
       reg2     <= 1;b0;
```

```
    end
    else
    begin
       reg1     <= data_a;
       reg2     <= data_b;
    end
end
endmodule  // sync_async_reset
```

To minimize the metastability effect between the two synchronization registers, and to increase the MTBF, the registers should be located as close as possible in the device to minimize routing delay. If possible, locate the registers in the same logic array block (LAB). The input reset signal (`reset_n`) must be excluded with a `set_false_path` command:

```
set_false_path -from [get_ports {reset_n}] -to [all_registers]
```

The `set_false_path` command used with the specified constraint excludes unnecessary input timing reports that would otherwise result from specifying an input delay on the reset pin.

The instantaneous assertion of synchronized asynchronous resets is susceptible to noise and runt pulses. If possible, you should debounce the asynchronous reset and filter the reset before it enters the device. The circuit ensures that the synchronized asynchronous reset is at least one full clock period in length. To extend this time to *n* clock periods, you must increase the number of synchronizer registers to *n* + 1. You must connect the asynchronous input reset (`reset_n`) to the CLRN pin of all the synchronizer registers to maintain the asynchronous assertion of the synchronized asynchronous reset.

2.3.2. Use Global Clock Network Resources

Intel FPGAs provide device-wide global clock routing resources and dedicated inputs. Use the FPGA's low-skew, high fan-out dedicated routing where available.

By assigning a clock input to one of these dedicated clock pins or with an Intel Quartus Prime assignment to assign global routing, you can take advantage of the dedicated routing available for clock signals.

In an ASIC design, you must balance the clock delay distributed across the device. Because Intel FPGAs provide device-wide global clock routing resources and dedicated inputs, there is no need to manually balance delays on the clock network.

Limit the number of clocks in the design to the number of dedicated global clock resources available in the FPGA. Clocks feeding multiple locations that do not use global routing may exhibit clock skew across the device leading to timing problems. In addition, generating internal clocks with combinational logic adds delays on the clock path. Delay on a clock line can result in a clock skew greater than the data path length between two registers. If the clock skew is greater than the data delay, you violate the timing parameters of the register (such as hold time requirements) and the design does not function correctly.

FPGAs offer low-skew global routing resources to distribute high fan-out signals. These resources help with the implementation of large designs with multiple clock domains. Many large FPGA devices provide dedicated global clock networks, regional clock networks, and dedicated fast regional clock networks. These clocks are organized into a hierarchical clock structure that allows multiple clocks in each device region with low

skew and delay. There are typically several dedicated clock pins to drive either global or regional clock networks, and both PLL outputs and internal clocks can drive various clock networks.

Intel Stratix 10 devices have a newer architecture. You can configure Intel Stratix 10 clocking resources to create efficiently balanced clock trees of various sizes, ranging from a single clock sector to the entire device. By default, the Intel Quartus Prime Software automatically determines the size and location of the clock tree. Alternatively, you can directly constrain the clock tree size and location either with a Clock Region assignment or by Logic Lock Regions.

To reduce clock skew in a given clock domain and ensure that hold times are met in that clock domain, assign each clock signal to one of the global high fan-out, low-skew clock networks in the FPGA device. The Intel Quartus Prime software automatically assigns global routing resources for high fan-out control signals, PLL outputs, and signals feeding the global clock pins on the device. To direct the software to assign global routing for a signal, turn on the **Global Signal** option in the Assignment Editor.

Note: Global Signal assignments only controls whether a signal is promoted using the specified dedicated resources or not, but does not control which or how many resources are used.

To take full advantage of the routing resources in a design, make sure that the sources of clock signals (input clock pins or internally-generated clocks) drive only the clock input ports of registers. In older Intel device families, if a clock signal feeds the data ports of a register, the signal may not be able to use dedicated routing, which can lead to decreased performance and clock skew problems. In general, allowing clock signals to drive the data ports of registers is not considered synchronous design and can complicate timing closure.

2.3.3. Use Clock Region Assignments to Optimize Clock Constraints

The Intel Quartus Prime software determines how clock regions are assigned. You can override these assignments with Clock Region assignments to specify that a signal routed with global routing paths must use the specified clock region.

Clock Region assignments allow you to control the placement of the clock region for floorplanning reasons. For example, use a Clock Region assignment to ensure that a certain area of the device has access to a global signal, throughout your design iterations. A Clock Region assignment can also be used in cases of congestion involving global signal resources. By specifying a smaller clock region size, the assignment prevents a signal using spine clock resources in the excluded sectors that may be encountering clock-related congestion.

You can specify Clock Region assignments in the assignment editor.

2.3.3.1. Clock Region Assignments in Intel Stratix 10 Devices

In Intel Stratix 10 devices, clock networks are constructed using programmable clock routing. As with other Intel device families, you can use Clock Region assignments for floorplanning, controlling the size and location of each clock tree.

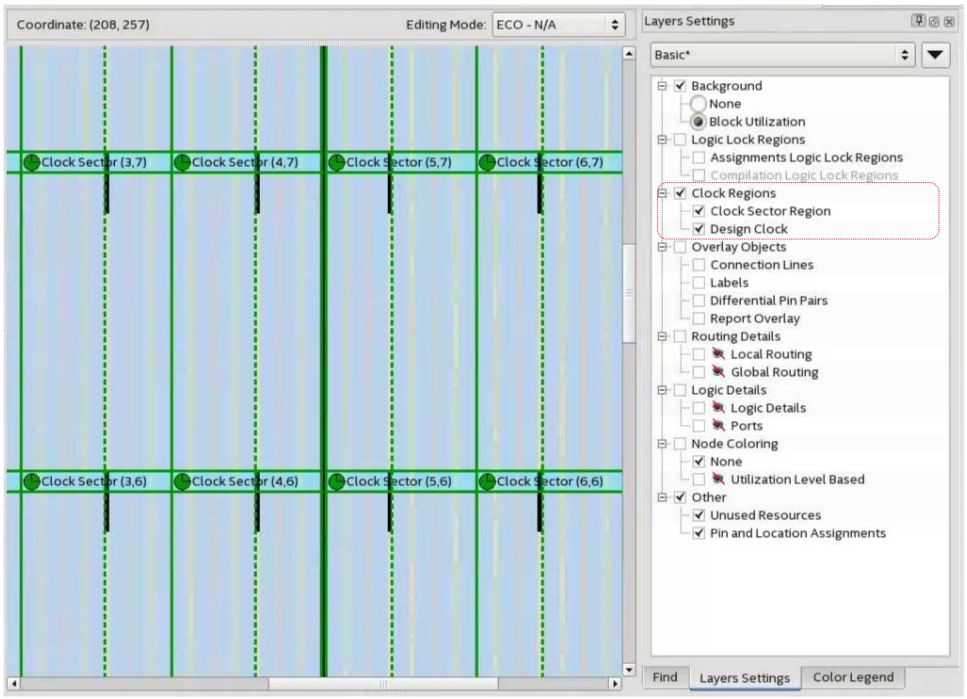
Although the Intel Quartus Prime Pro Edition software generates balanced clock trees, there are sources of timing variation, such as process variation and jitter, which prevents clock trees from being perfectly skew balanced. Longer paths, with higher insertion delay, have more timing variation. However, the Timing Analyzer can account

for and eliminate some sources of variation in timing along common clock paths. In practice, this means that the size of the clock region has a significant impact on the worst-case skew of the clock tree; a larger clock tree experiences higher insertion delay and worst-case clock skew when compared to a smaller clock region. The distance between the clock region and the clock source also increases insertion delay, but the impact of distance on worst-case clock skew is much smaller than the impact of the size of the clock region.

One case to consider is when a design contains high-speed clock domains that are expected to grow during the design process. Specifying a clock region constraint to create a larger clock region than the compiler generates automatically helps ensure that timing closure is robust with higher clock insertion delays and clock skews.

An additional design consideration is the minimum pulse width constraint on clock signals. For a clock signal to propagate correctly on the Intel Stratix 10 clock network, a minimum delay must be met between the rising edge and falling edge of the clock pulse. If the Timing Analyzer cannot guarantee that this constraint is met, the clock signal may not propagate as expected under all operating conditions. This can happen when the delay variation on a clock path becomes too great. This situation does not normally occur, but may arise if clock signals are routed through core logic elements or core routing resources.

In designs that target Intel Stratix 10 devices, clock regions can be constrained to a rectangle whose dimensions are defined by the sector grid, as seen in the Clock Sector Region layer of the Chip Planner.



This assignment specifies the bottom left and top right coordinates of the rectangle in the format "SX# SY# SX# SY#". For example, "SX0 SY0 SX1 SY1" constrains the clock to a 2x2 region, from the bottom left of sector (0,0) to the top right of sector (1,1). For a constraint spanning only one sector, it is sufficient to specify the location of that sector, for example "SX1 SY1". The bounding rectangle can also be specified

by the bottom left and top right corners in chip coordinates, for example, "X37 Y181 X273 Y324". However, such a constraint should be sector aligned (using sector coordinates guarantees this) or the Fitter automatically snaps to the smallest sector aligned rectangle that still encompasses the original assignment. The "SX# SY# SX# SY#"|"X# Y# X# Y#" strings are case-insensitive.

2.3.3.2. Clock Region Assignments in Intel Arria 10 and Older Device Families

In device families with dedicated clock network resources and predefined clock regions, this assignment takes as its value the names of those Global, Regional, Periphery or Spine Clock regions. These region names are visible in Chip Planner by enabling the appropriate Clock Region layer in the **Layers Settings** dialog box. Examples of valid values include `Regional Clock Region 1` or `Periphery Clock Region 1`.

When constraining a global signal to a smaller than normal region, for example, to avoid clock congestion, you may specify a clock region of a different type than the global resources being used. For example, a signal with a Global Signal assignment of `Global Clock`, but a Clock Region assignment of `Regional Clock Region 0`, constrains the clock to use global network routing resources, but only to the region covered by `Regional Clock Region 0`. To provide a finer level of control, you can also list multiple smaller clock regions, separated by commas. For example: `Periphery Clock Region 0, Periphery Clock Region 1` constrains a signal to only the area reachable by those two periphery clock networks.

2.3.4. Avoid Asynchronous Register Control Signals

Avoid using an asynchronous load signal if the design target device architecture does not include registers with dedicated circuitry for asynchronous loads. Also, avoid using both asynchronous clear and preset if the architecture provides only one of these control signals.

Some Intel devices directly support an asynchronous clear function, but not a preset or load function. When the target device does not directly support the signals, the synthesis or placement and routing software must use combinational logic to implement the same functionality. In addition, if you use signals in a priority other than the inherent priority in the device architecture, combinational logic may be required to implement the necessary control signals. Combinational logic is less efficient and can cause glitches and other problems; it is best to avoid these implementations.

2.4. Implementing Embedded RAM

Intel's dedicated memory architecture offers many advanced features that you can enable with Intel-provided IP cores. Use synchronous memory blocks for your design, so that the blocks can be mapped directly into the device dedicated memory blocks.

You can use single-port, dual-port, or three-port RAM with a single- or dual-clocking method. You should not infer the asynchronous memory logic as a memory block or place the asynchronous memory logic in the dedicated memory block, but implement the asynchronous memory logic in regular logic cells.

Intel memory blocks have different read-during-write behaviors, depending on the targeted device family, memory mode, and block type. Read-during-write behavior refers to read and write from the same memory address in the same clock cycle; for example, you read from the same address to which you write in the same clock cycle.

You should check how you specify the memory in your HDL code when you use read-during-write behavior. The HDL code that describes the read returns either the old data stored at the memory location, or the new data being written to the memory location.

In some cases, when the device architecture cannot implement the memory behavior described in your HDL code, the memory block is not mapped to the dedicated RAM blocks, or the memory block is implemented using extra logic in addition to the dedicated RAM block. Implement the read-during-write behavior using single-port RAM in Arria GX devices and the Cyclone and Stratix series of devices to avoid this extra logic implementation.

In many synthesis tools, you can specify that the read-during-write behavior is not important to your design; if, for example, you never read and write from the same address in the same clock cycle.

Related Information

Inferring RAM functions from HDL Code on page 9

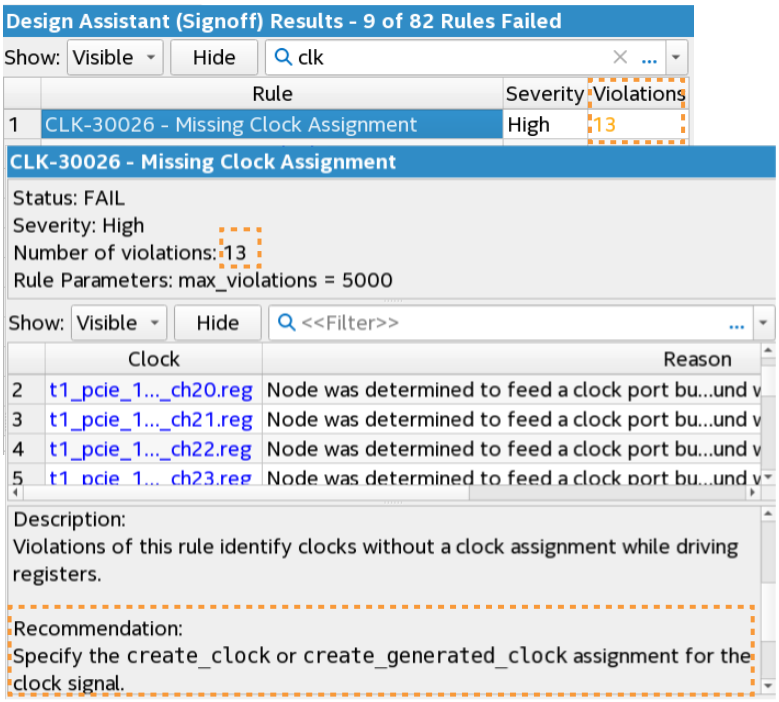
2.5. Design Assistant Design Rule Checking

The Intel Quartus Prime Design Assistant increases productivity by reducing the total number of design iterations for design closure, and by minimizing the time in each iteration with targeted rule checks and guidance at each stage of compilation.

The Design Assistant detects and helps you to resolve design rule violations by providing recommendations for correction and pathways to the violation source. Avoiding design rule violations improves the reliability, timing performance, and logic utilization of your design.

When enabled, Design Assistant automatically reports any violations against a standard set of Intel FPGA-recommended design guidelines [(1)]. You can run Design Assistant automatically during compilation, and report violations detected throughout the compilation process.

Figure 23. Design Assistant Recommends Corrections for Design Rule Violations



Alternatively, you can run Design Assistant in analysis mode, which allows you to launch Design Assistant checks from other Intel Quartus Prime tools, such as Chip Planner. For some rules, Design Assistant supports cross-probing to the Timing Analyzer and Intel Quartus Prime design visualization tools for root cause analysis and correction.

You can specify which rules Design Assistant checks, thus eliminating the rule checks that are unimportant for your design.

(1) A set of default rules ensures design health without significant runtime increase.

Related Information

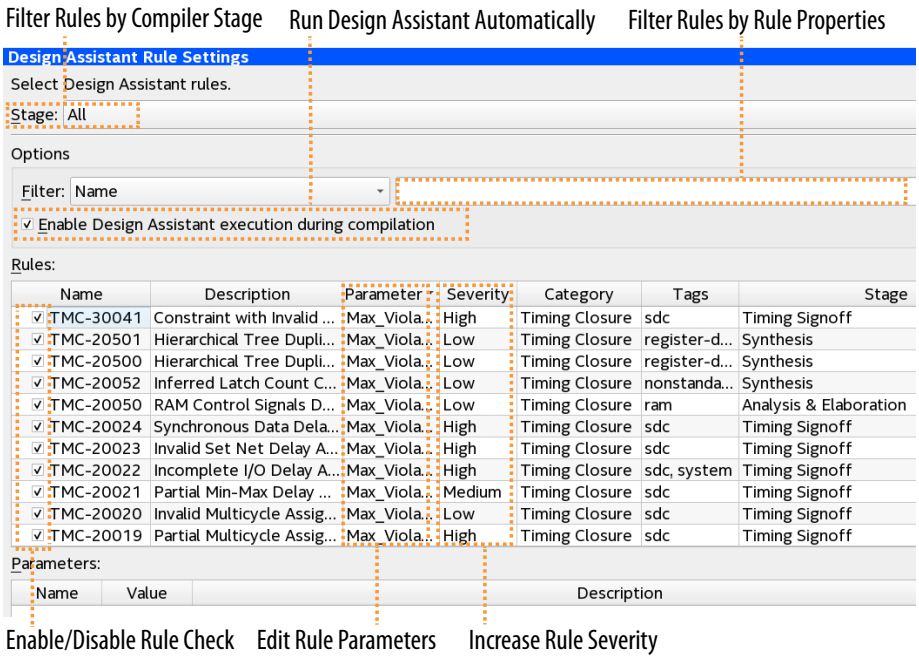
AN 919: Improving Quality of Results with Design Assistant

2.5.1. Setting Up Design Assistant

Customize the Design Assistant for individual design characteristics and reporting requirements. For example, you can disable rules for specific stages of compilation, change the threshold for violation reporting, and other options. Follow these steps to specify initial options for running Design Assistant:

1. Open an Intel Quartus Prime project.
2. Click **Assignments ➤ Settings ➤ Design Assistant Rule Settings**.

Figure 24. Design Assistant Rule Settings



3. Use the default settings or specify any of the following options:

Table 4. Design Assistant Rule Settings

Option	Description
Stage filter	Filters the **Rules** list by **All**, **Analysis & Elaboration**, **Synthesis**, **Plan**, **Place**, **Finalize** or **Timing Signoff** Compiler stages.
Text **Filter**	Filters the **Rules** list by matching text and the **Name**, **Description**, **Parameter**, **Severity**, **Category**, or **Tags** of the rule.
Enable Design Assistant execution during compilation	Runs Design Assistant automatically during compilation. Alternatively, enable this setting with `FLOW_ENABLE_DESIGN_ASSISTANT` in the `.qsf`. The settings in this dialog have no impact when this setting is disabled.
Rules	Lists all available Design Assistant rules and properties. Enable or disable analysis for the rule by enabling or disabling the rule **checkbox**.

continued...

Option	Description
Name Column	Specifies the alphanumeric rule ID. Rules that apply to more than one Compiler stage have sub-rules for each stage.
Description Column	Summary rule description.
Parameter Column	Lists rule parameters or "Multiple Values" for rules that support multiple Compiler stages. Select any rule to edit parameter values. Specify parameters on a per-stage basis by specifying parameters for the stage subrule.
Severity Column	Specifies **Low**, **Medium**, **High**, **Critical**, or **Fatal** as the rule **Severity** for reporting. You can increase the **Severity** level of rules.
Category Column	Specifies the rule class, such as **Timing Closure**, **Reset**, and others.
Tags	Specifies one or more additional facet of the rule for search and filtering purposes. For example, **global-signal** tag for design rule checks related to global signals. *Design Assistant Tags* defines the meaning of each tag.
Stage Column	Specifies the Compiler stages to which the rule applies. Rules for **Analysis & Elaboration**, **Synthesis**, **Plan**, **Place**, **Finalize**, and **Timing Signoff** stages are available. Enable or disable the rule on a per-stage basis by enabling or disabling the **checkbox** option for the stage subrule.
Parameters for rule Column	Allows you to specify parameters for rules that support parameters. Specify parameters on a per-stage basis by specifying parameters for the stage subrule.

Related Information

- Managing Design Assistant Rules on page 107
- Design Assistant Tags on page 115

2.5.1.1. Design Assistant Rule Severity Levels

Design Assistant designates each rule violation with a severity level. You can increase the severity level for any rule to match your particular design requirements.

Table 5. Design Assistant Rule Severity Levels

Severity	Description	Severity Level Color
Fatal	Failure condition that stops the Compiler flow after violation.	Red
Critical	A critical issue that requires correction for sign-off.	Red
High	Potentially causes functional failure. May indicate missing or incorrect design data.	Yellow
Medium	Potentially impacts quality of results for f_{MAX} or resource utilization.	Brown
Low	Optional suggestions that can reflect FPGA design best practices and may have small impact on device performance and utilization in typical designs.	Blue

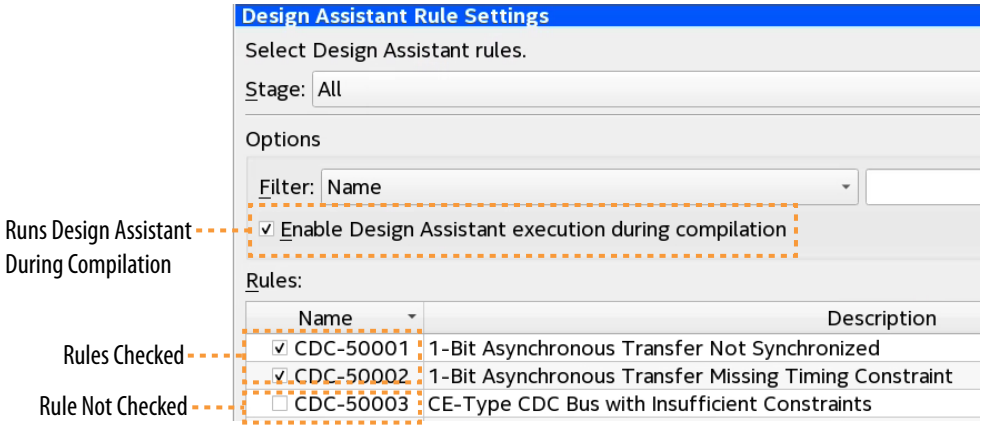
2.5.2. Running Design Assistant During Compilation

When enabled, Design Assistant runs automatically during compilation and reports design rule violations in the Compilation Report.

When you enable or specify parameters for a rule check in compilation mode, those specifications apply by default to running Design Assistant in compilation mode. If you change the rule settings for analysis mode, those settings are independent from the rule settings in compilation mode.

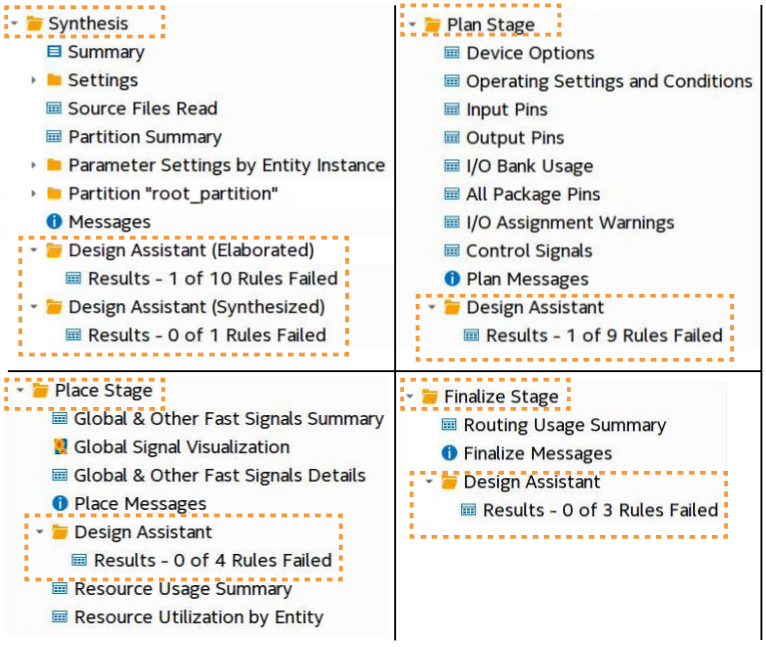
1. To run Design Assistant checking during compilation flows, ensure that **Enable Design Assistant execution during compilation** is on.
2. To enable or disable specific design rule checks, turn on or off the checkbox for that rule in the **Name** column. If the rule is unchecked, Design Assistant does not report violations for the rule.
3. In the **Parameters** field, consider changing default values for rules you enable.

Figure 25. Design Assistant Rule Settings



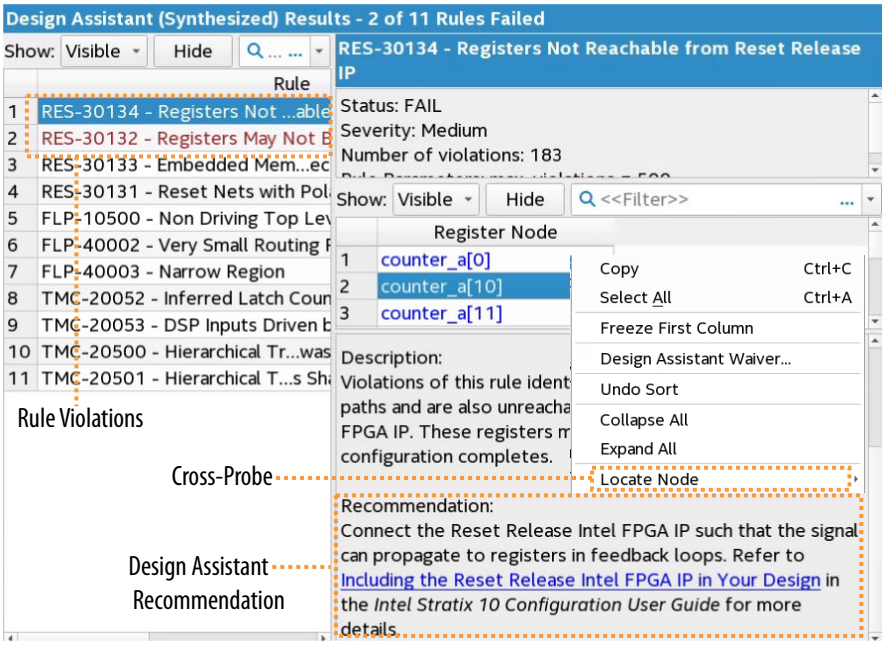
4. To run Design Assistant during compilation, run one or more stages of the Compiler from the Processing menu or Compilation Dashboard.

Figure 26. Example Design Assistant Results in Compilation Reports



5. To view the results for each rule, click the rule in the **Rules** list. A description of the rule and design recommendations for correction appear.

Figure 27. Design Assistant Rule Violation Recommendation

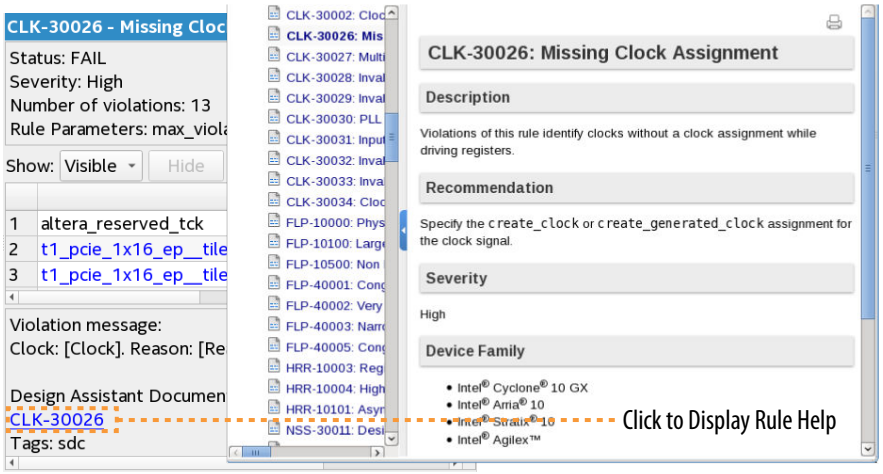


2.5.2.1. Opening Design Assistant Rule Help

Each rule report contains a link to the rule's Intel Quartus Prime Help topic. Help includes full rule descriptions and diagrams. To link to the Design Assistant rule Help:

1. In the Design Assistant report, click any rule in the **Rule** list. The right pane shows the rule description.

Figure 28. Linking in Design Assistant Reports to Rule Help



2. Click the rule ID link under **Design Assistant Document**. The rule displays in Help.

2.5.3. Running Design Assistant in Analysis Mode

You can launch Design Assistant in analysis mode directly from the Timing Analyzer or Chip Planner to rapidly run the specific rule checks that relate to those tools. For example, when you launch Design Assistant from the Chip Planner, Design Assistant is preset to check only a subset of the FLP (floorplanning) Design Assistant rules.

Similarly, when you launch Design Assistant from the Timing Analyzer, Design Assistant is preset to check only a subset of rules that are helpful during timing analysis. You can cross-probe to the Timing Analyzer and design visualization tools to determine the root cause of violations.

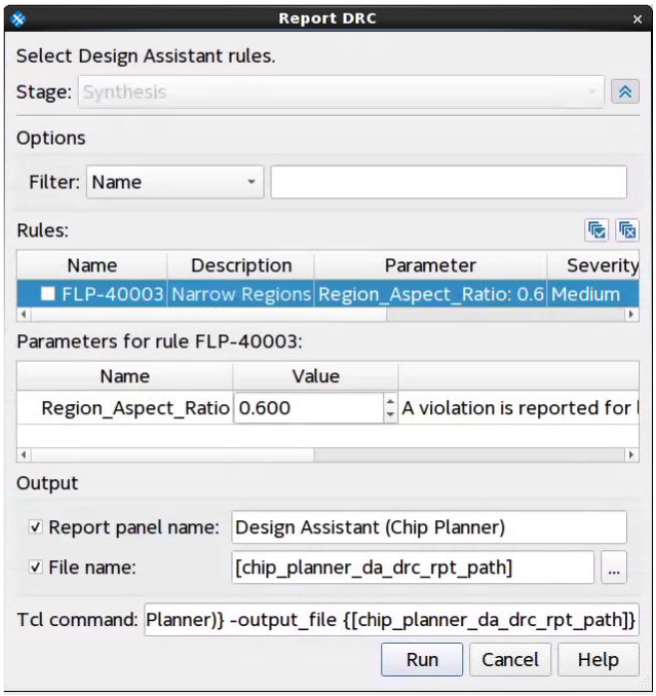
When you enable or specify parameters for a rule check in analysis mode, those specifications do not apply to running Design Assistant in compilation mode. The rule settings for analysis mode are independent from the rule settings in compilation mode.

2.5.3.1. Launching Design Assistant from Chip Planner

You can run Design Assistant directly from Chip Planner to assist when optimizing the floorplan in the tool. When you launch Design Assistant from the Chip Planner, Design Assistant is preset to check only the FLP (floorplanning) Design Assistant rules. Follow these steps to run the Design Assistant from the Chip Planner:

1. Run any stage of the Compiler. You must run at least the Analysis & Elaboration stage before running Design Assistant from Chip Planner.
2. Click **Tools ➤ Chip Planner**.

Figure 29. Report DRC Dialog Box in Chip Planner



3. In Chip Planner **Tasks** pane, click **Report DRC** under **Design Assistant**. The **Report DRC** (design rule check) dialog box appears.

4. Under **Rules**, disable any rules that are not important to your analysis by removing the check mark.
5. Consider whether to adjust rule parameter values in the **Parameters** field.
6. Under **Output**, confirm the **Report panel name** and optionally specify an output **File name**.
7. Click **Run**. The Results reports generate and appear in the **Report** pane and in the Compilation Report.

Figure 30. Rule Violations in Chip Planner Reports Pane

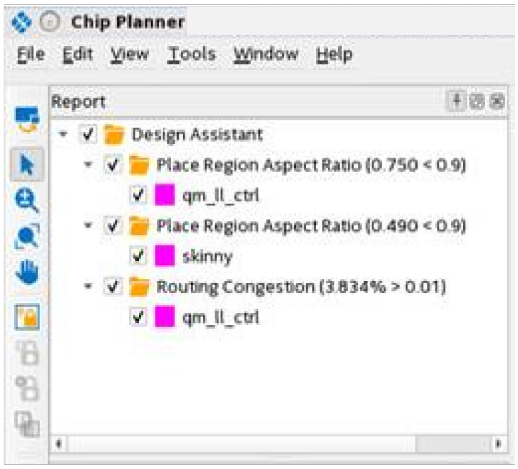
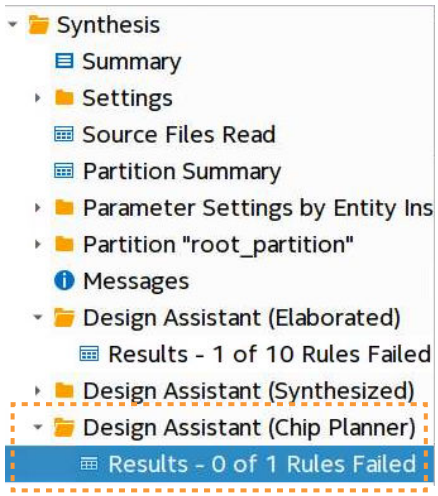


Figure 31. Chip Planner Rule Violations in Main Compilation Report



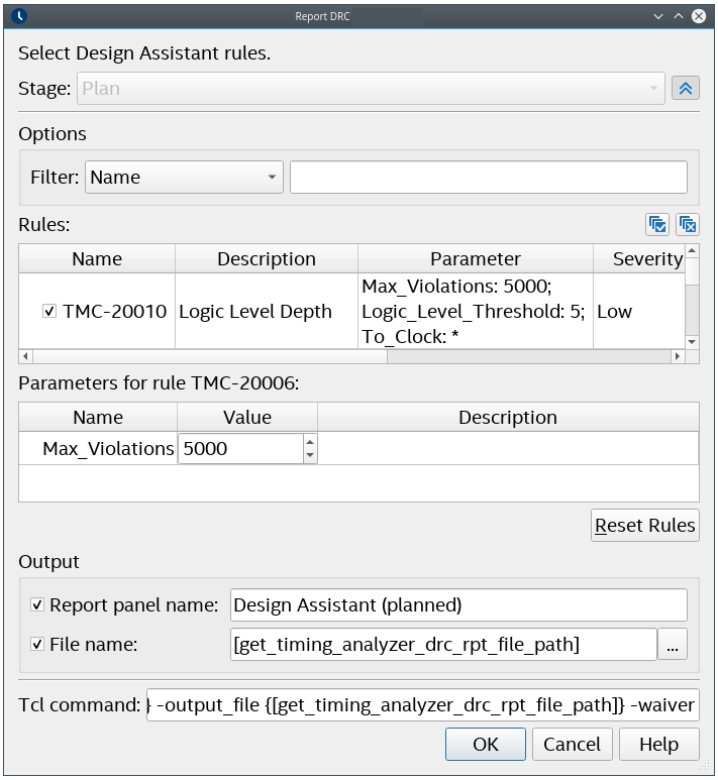
2.5.3.2. Launching Design Assistant from Timing Analyzer

You can run Design Assistant directly from the Timing Analyzer to assist when optimizing timing paths and other timing conditions. When you launch Design Assistant from the Timing Analyzer, Design Assistant is preset to check only rules that relate to timing analysis.

Follow these steps to run the Design Assistant from the Timing Analyzer:

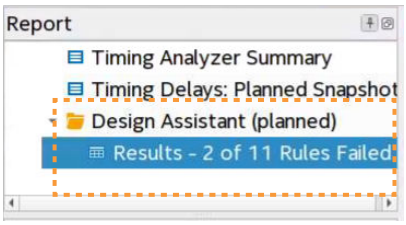
1. Compile the design through at least the Compiler's Plan stage.
2. Open the Timing Analyzer for the Compiler stage from the Compilation Dashboard.
3. In the Timing Analyzer, click **Reports ➤ Design Assistant ➤ Report DRC...**. The **Report DRC** (design rule check) dialog box opens.
4. Under **Rules**, disable any rules that are not important to your analysis by removing the check mark.
5. Consider whether to adjust rule parameter values in the **Parameters** field.

Figure 32. Report DRC (Design Rule Check) Dialog Box



6. Confirm the **Report panel name** and optionally specify an output **File name**.
7. Click **Run**. The Results reports generate and appear in the **Report** pane, as well as the main Compilation Report.

Figure 33. Design Assistant Reports in Timing Analyzer Report Pane



2.5.4. Cross-Probing from Design Assistant

In addition to the Locate Node command, Design Assistant allows you to cross-probe individual design objects relevant to the violation. For select high-value rules, Design Assistant provides full violation cross-probing ability into the Intel Quartus Prime Timing Analyzer and other design visualization tools.

For example, for rule TMC-20210 - Paths Failing Setup Analysis with High Routing Delay Added for Hold, you can right-click the violation, and then click **Report Timing (Extra Info)** to locate the path in the Timing Analyzer GUI.

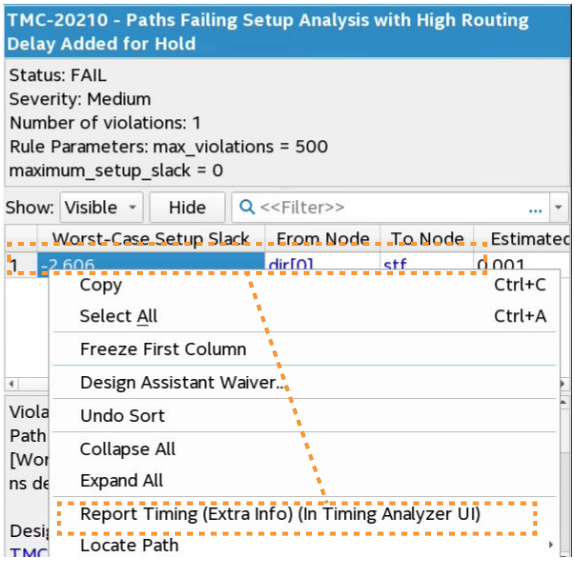
You can also locate from a rule violation instance to the source of the violation in Intel Quartus Prime design visualization tools, such as RTL Viewer, Resource Property Viewer, Technology Map Viewer, and Chip Planner. You can also locate to the violation source in the design file.

Cross-probing with Design Assistant can help you to more rapidly identify the root cause and resolve any rule violations negatively impacting your design.

2.5.4.1. Cross-Probing from Design Assistant to Timing Analyzer

Some Design Assistant rule violations allow cross-probing into Timing Analyzer. For example, for a path that Design Assistant flags with a setup analysis violation due to delay added for hold, you can cross-probe into the Timing Analyzer to view more information on the affected path and edge.

Figure 34. Cross Probing from Design Assistant Rule TMC-20210 Violations to Timing Analyzer



Follow these steps to cross-probe from such Design Assistant rule violations to the Timing Analyzer:

1. Compile the design through at least the Compiler's Plan stage.
2. Locate a rule violation in the Design Assistant folder of the Compilation Report.
3. Right-click the rule violation to display any **Report Timing** commands available for the violation.
4. Click the **Report Timing** command. The Timing Analyzer opens and reports the timing data for the violation path. **Report Timing (Extra Info)** includes Estimated Delay Added for Hold and Route Stage Congestion Impact extra data.

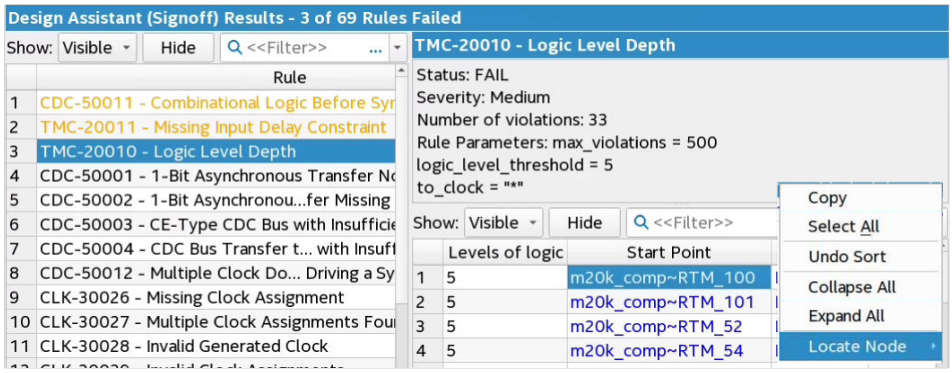
2.5.4.2. Cross-Probing from Design Assistant to Visualization Tools

Design Assistant can cross-probe from rule violations to the source in various Intel Quartus Prime design visualization tools. The following example demonstrates expanding from the cross-probing location for violation analysis.

The following example illustrates cross probing for the TMC-20010 Logic Level Depth rule violation to the RTL Viewer:

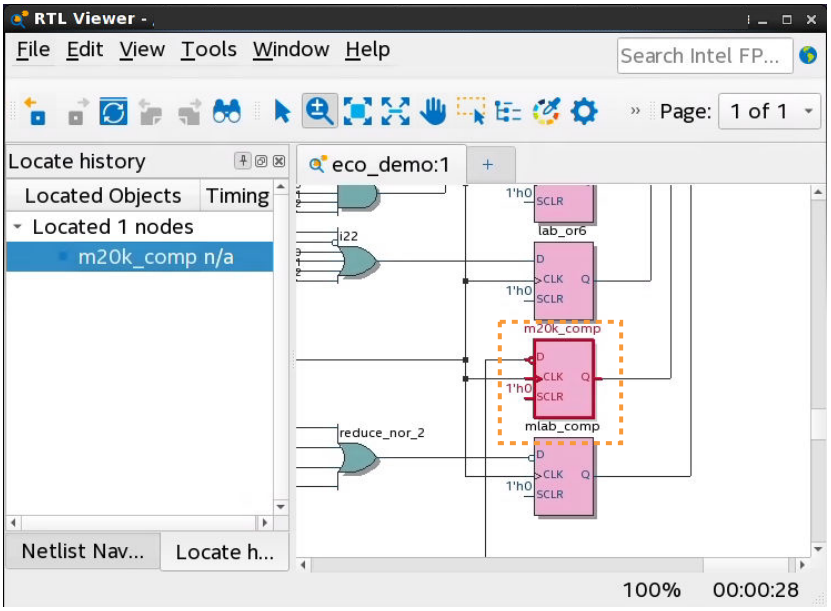
1. When Design Assistant reports FAIL status for rule TMC-20010, you can right-click any of the rule violations in the Design Assistant report, and then click **Locate Node ➤ Locate in RTL Viewer**.

Figure 35. Locate in RTL Viewer



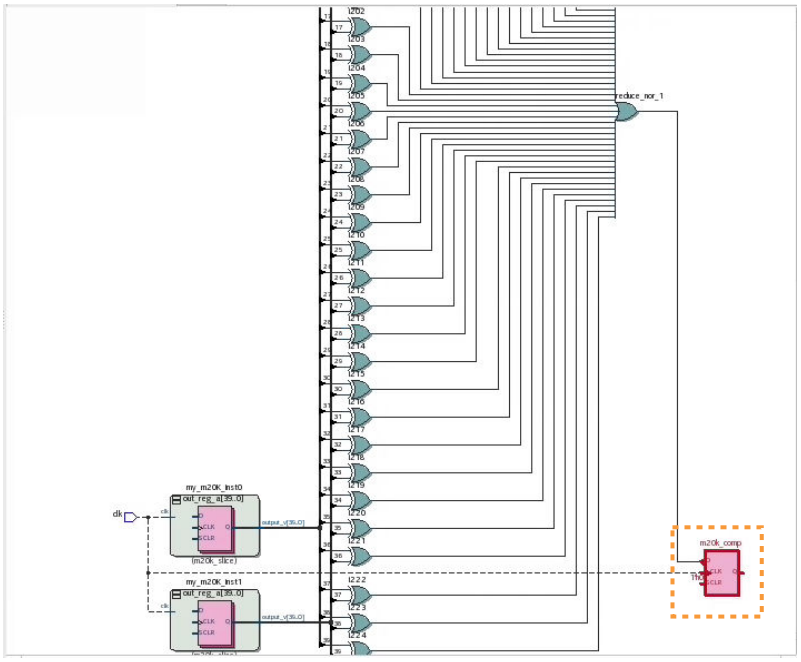
Cross-probing allows you to locate the driver register in the RTL Viewer.

Figure 36. Driver Register in RTL Viewer



2. To then fully visualize the logic level depth, right-click the register and click **Filter** to display **Sources and Destinations** of the register.

Figure 37. Expanded Connections



2.5.5. Managing Design Assistant Rules

The Design Assistant provides functions to help you manage the rules that are important for your design characteristics. You can use these features to help ensure that you are only checking rules that are important for your design at the relevant stage of compilation.

Design Assistant provides the following functionality to help you manage rule checking and reporting:

- Waiving Design Assistant Rules on page 109
- Enabling Rules for Specific Compiler Stages on page 107
- Specifying Rule Parameters for a Specific Compiler Stage on page 108
- Modifying Rule Severity Levels on page 108
- Changing the Default Number of Violations per Rule on page 107

2.5.5.1. Changing the Default Number of Violations per Rule

The default number of violations per rule is set to 5000 to limit runtime that rule processing incurs. The default limit is set to 5000 to help ensure that every violation is presented in all but the most exceptional cases. In typical designs, most rule violations do not approach this limit.

You can change the default number of violations that Design Assistant reports per rule, to show more or less violations per rule. Specify the following assignment in the project `.qsf` to change the default number of violations per rule:

```
set_global_assignment -name DESIGN_ASSISTANT_MAX_VIOLATIONS_PER_RULE <number>
```

To specify no limit on the number of violations per rule, at the possible expense of increased runtimes, enter `-1` for the `<number>` of the `DESIGN_ASSISTANT_MAX_VIOLATIONS_PER_RULE` assignment.

Change the default number of violations for individual rules in the **Design Assistant Settings** page, as Setting Up Design Assistant on page 97 describes.

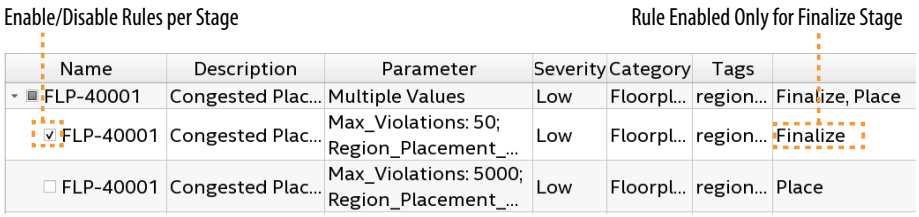
2.5.5.2. Enabling Rules for Specific Compiler Stages

You can enable or disable checking of Design Assistant rules on a per stage basis. This feature allows you to disable rule checks that are not important during one or more stage of compilation, while leaving the rule enabled for other stages.

To enable or disable rules for specific Compiler stages, follow these steps:

1. Specify initial Design Assistant Settings, as Setting Up Design Assistant on page 97 describes.
2. In the **Design Assistant Rule Settings** page, click the arrow next to a rule that supports multiple Compiler stages to expand the subrules for each stage.
3. For the subrule, enable or disable the **checkbox** to enable or disable checking for that rule during that Compiler stage.

Figure 38. Enable or Disable Rules per Stage



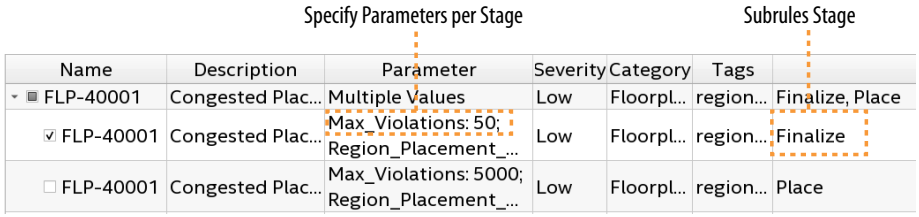
2.5.5.3. Specifying Rule Parameters for a Specific Compiler Stage

You can specify different parameters for Design Assistant rules for each Compiler stages. This feature allows you apply a set of rule parameters to a specific stage of compilation, and have a different set of rule parameters for another stage.

To enable or disable parameters for specific Compiler stages, follow these steps:

1. Specify initial Design Assistant Settings, as Setting Up Design Assistant on page 97 describes.
2. In the **Design Assistant Rule Settings** page, click the arrow next to a rule that supports multiple Compiler stages to expand the subrules for each stage.
3. Select the subrule and then specify the parameters in **Parameters for rule**.

Figure 39. Specifying Parameters Per Stage



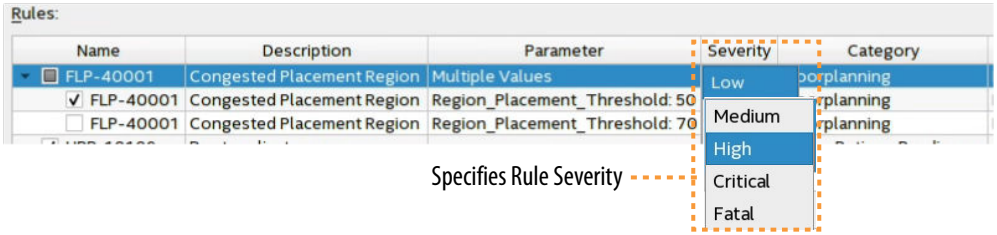
2.5.5.4. Modifying Rule Severity Levels

You can increase the severity level of a Design Assistant rule to match the importance of the rule for your design. You cannot decrease the severity level below the default. Design Assistant messages and reports reflect the rule severity level. You can filter and hide rule messages based on the severity level that you specify.

To customize rule severity level, follow these steps:

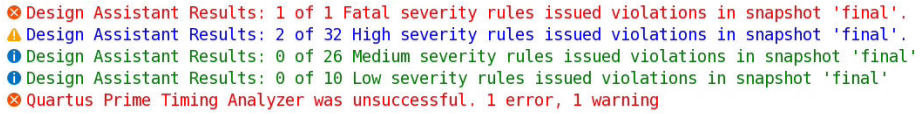
1. Specify initial Design Assistant Settings, as Setting Up Design Assistant on page 97 describes.
2. In the **Design Assistant Rule Settings** page, select the rule with a **Severity** that you want to change. You can only change the severity level of parent rules. Subrules for each stage must reflect the parent rule **Severity** level.
3. Click the **Severity** cell and select **Low**, **Medium**, **High**, **Critical**, or **Fatal**. Design Assistant reports the severity level you specify for the rule violations. Fatal violations cause failure of the Compiler stage.

Figure 40. Modifying Rule Severity Level



Note: Fatal violations indicate conditions that cause design failure and therefore cause the Compiler stage to be unsuccessful. You must correct the fatal condition, reduce the rule **Severity**, or create a rule waiver before proceeding to the next Compiler stage.

Figure 41. Messages Report Fatal Rule Violation Causes Compiler Failure



2.5.5.5. Waiving Design Assistant Rules

After running an initial design rule check, you can waive (ignore) design rule violations that you determine are unimportant for one or more iterations of design rule checking. When you create a waiver, Design Assistant does not check for compliance with the rules that match the violation conditions you specify, nor report results for the rule. For teams or individual designers, rule waivers also provide an audit trail that tracks the user, description, and reason for the design rule waiver.

You can create rule waivers to ignore violations for which you already have identified the root cause and correction, for violations that occur in a block that another developed owns, or to waive specific rules that you determine are not an issue for your design.

Initially, run Design Assistant checks without rule waivers to evaluate the complete list of violations. As you begin root cause analysis and violation correction, you can consider creating waivers to eliminate one or more rule violations from obscuring the rule violations that are still relevant.

After creating a design rule waiver, you can modify the rule parameters to fine tune rule checks, or you can delete waivers. For example, if a first pass rule check reports 800 violations with the **Max_Violations** per rule parameter set to default of 500, Design Assistant reports only the first 500 of the 800 total violations. You could then create rule waivers to omit the first 100 rule violations that you correct, thereby reporting rule violations number 501 and higher the next time you run Design Assistant.

When a Design Assistant waiver becomes completely unneeded, you can delete the waiver in the **Design Assistant Manage Waivers** dialog box or directly from the Design Assistant Waivers (`.dawf`) file.

Creating Design Assistant Waivers on page 110

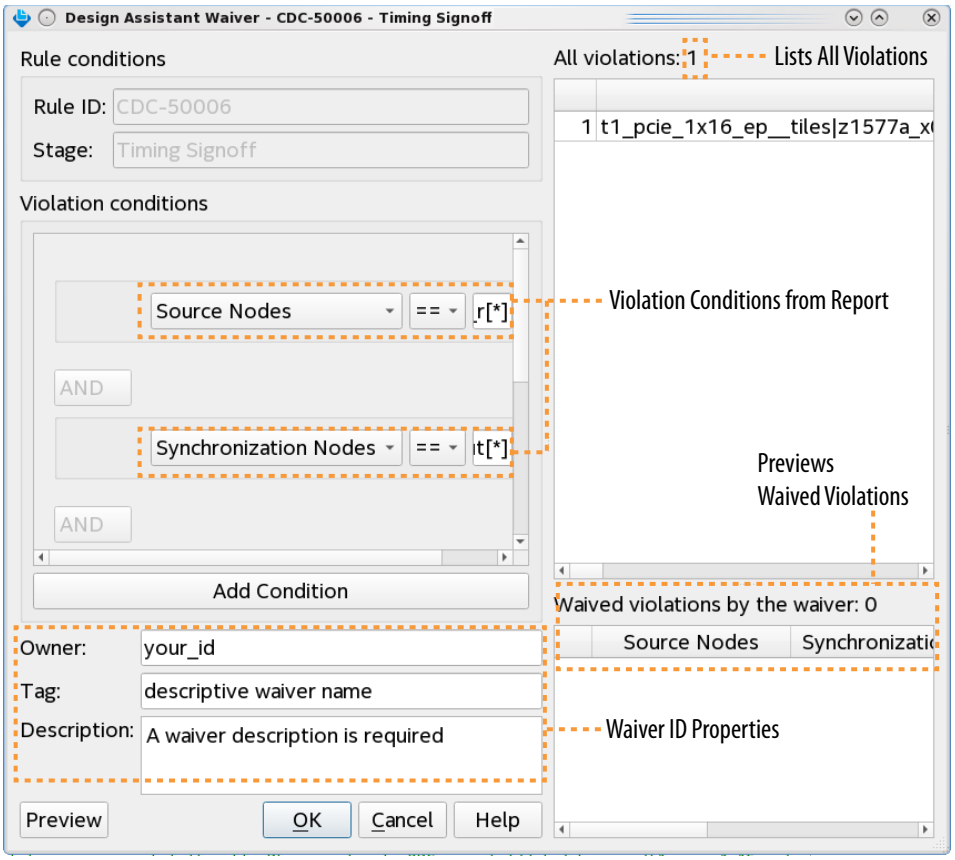
Design Assistant Waiver Dialog Box on page 111

2.5.5.5.1. Creating Design Assistant Waivers

To create a design rule waiver, follow these steps:

1. Run one or more stages of the Compiler to generate Design Assistant reports for the rules that you enable for your design.
2. In the Design Assistant report, right-click one or more rule violations, or right-click an entire rule category in the rule summary list, and then click **Design Assistant Waiver**. The **Design Assistant Waiver** dialog box opens preset with values from your violation selection.

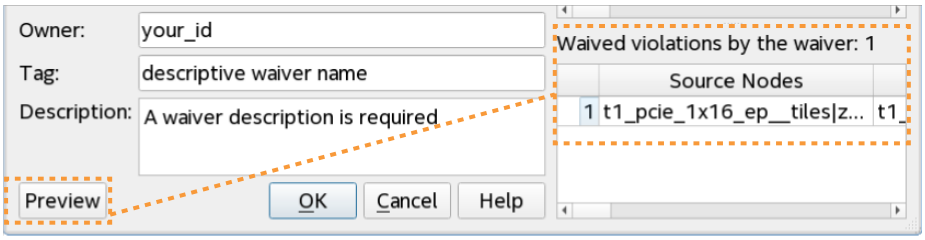
Figure 42. Right-Click Rule Violation in Report to Create Waiver for Violation



3. Modify any of the default **Violation conditions** that define when the waiver applies. The default settings are the most descriptive, using all applicable fields. The comparison operator is always **==** (equal to) by default for all conditions. Refer to Design Assistant Waiver Dialog Box on page 111 for all available options.

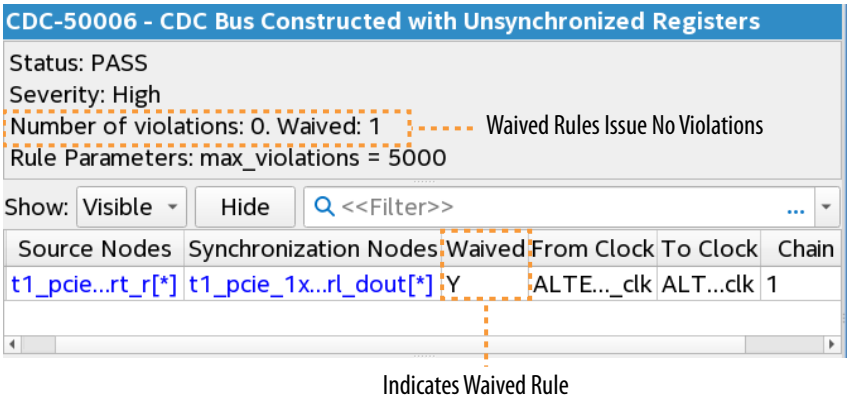
4. Click the **X** button to delete a sub-condition and simplify the query. Click **Add Condition** to add a violation sub-condition.
5. For waiver identification and audit tracking, optionally specify the waiver **Owner** name, a descriptive **Tag**, and a text **Description**.

Figure 43. Design Assistant Waiver Dialog Box



6. To preview the waived violations, click the **Preview** button. The **Waived violations by the waiver** list shows the waived rule violations during the next Design Assistant run. When you create a waiver after running Design Assistant, the newly added waiver specifies **To be Waived** in the **Waived** column. For any waivers that you delete, the **Waiver** column specifies **Y + To be unwaived**.
7. When waiver definition is complete, click **OK** to apply the waiver the next time you run Design Assistant. Design Assistant does not check for compliance with the rules that match waiver conditions, nor report results for the rules you waive. The Design Assistant reports indicate waived violations following compilation.

Figure 44. Applied Waiver Reported in Compilation Report



The report's **Waived** column specifies **Y** (for yes) for waived violations.

Design Assistant saves the rule waiver to a `da_drc.dawf` file in the project directory.

2.5.5.5.2. Design Assistant Waiver Dialog Box

You can define and apply waivers to Design Assistant rule violations that are not of concern in the **Design Assistant Waiver** dialog box.

The following table shows the **Design Assistant Waiver** dialog box options:

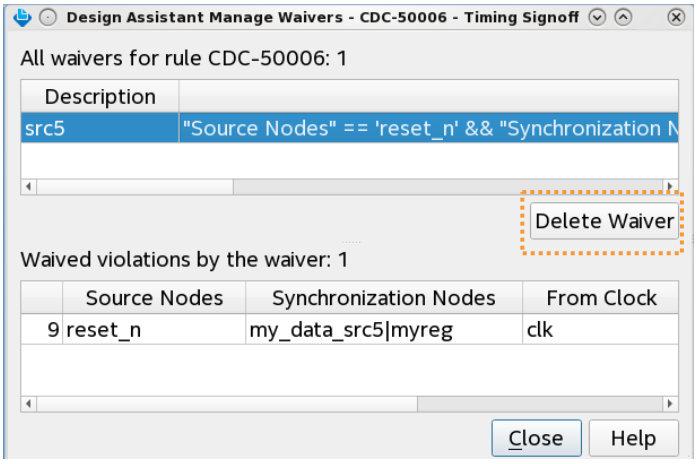
Table 6. Design Assistant Waiver Dialog Box Options

Setting	Description
Rule conditions	Automatically specifies the alphanumeric **Rule ID** and Compiler **Stage** of the rule violation.
Violation conditions	Specifies the conditions that define a rule violation waiver. The default **Violation conditions** automatically reflect the currently selected rule violation from the report. The available condition attributes are context-sensitive: • **==**is equal to • **!=** is not equal to • **<** is less than • **>** is greater than • **<=** is less than or equal to • **>=** is greater than or equal to • **!** is a negative unary operator for boolean expressions • **=~**—string contains • **!~**—string does not contain The join operators between conditions are always **AND**.
Add Condition	Click the **Add Condition** button to add more conditions to the rule waiver definition. Clicking the button adds a condition of default type.
Owner	Specifies the waiver owner's ID.
Tag	Specifies an identifying tag for the rule waiver.
Description	Required value that specifies a text description of the waiver for tracking and identifying.
Preview button	Previews the rule violations waived during the next Design Assistant run in the **Waived by the waiver** field.
Waived violations by the waiver	Upon clicking the **Preview** button, lists the rule violations that are waived during the next Design Assistant run.
All violations	Lists all violations of the currently selected Design Assistant rule.

2.5.5.5.3. Deleting Design Assistant Waivers

To fully remove (delete) any Design Assistant waivers that are no longer useful, delete the waiver in the **Design Assistant Manage Waivers** dialog box.

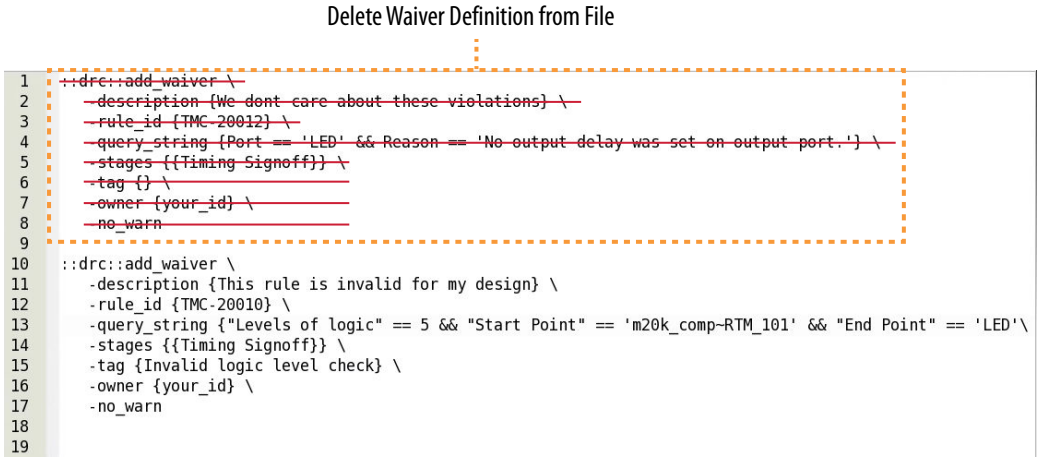
Figure 45. Design Assistant Manage Waivers Dialog Box



1. Right-click a rule violation with a waiver, and click **Waivers ➤ Manage Waivers**.
2. In the **Design Assistant Manager Waivers** dialog box, select the waiver you want to delete under **All waivers for rule**.
3. Click the **Delete Waiver** button to delete the waiver.

As an alternative to the GUI, you can also create and modify Design Assistant waivers by editing the Design Assistant Waivers File (`da_drc.dawf`), or by using Tcl commands within an interactive Tcl shell. The `da_drc.dawf` stores the waiver definitions you specify in the GUI. To avoid unintended results, do not edit the `da_drc.dawf` file while Compiler processes are running.

Figure 46. Deleting Waiver from Design Waivers File



2.5.5.5.4. Design Assistant Waiver Tcl Commands

As an alternative to the Design Assistant GUI, you can create, query, and report Design Assistant waivers by specifying the following Tcl commands within an interactive Tcl shell:

2.5.5.5.5. drc::add_waiver Command

Description

Creates a new Design Assistant waiver for the rule, Compiler stages, and query string that you specify. Optionally maintain audit tracking with the `timestamp` and `owner` arguments. Specify wildcard characters with the `stages` argument to create generic waivers across multiple rules. Specify any subset of violation arguments, comparison operators, and join operators to build any generic query string.

Syntax

```
drc::add_waiver
-description <description text>
-owner <userid>
-rule <rule_id>
query_string
-stages <compiler stages>
[-tag <string>]
```

Arguments

description Text description explaining the reason for the waiver.

owner User ID of waiver creator for audit trail.

rule Alpha-numeric rule ID.

query_string Query string specifying violation column arguments to define the violation patterns the waiver ignores. You can specify all or a subset of all violation column arguments, depending on the scope of the waiver.

stages Subset of Compiler stage(s) to which the rule waiver applies.

tag Short text description for tracking different types of violations across the entire project.

2.5.5.5.6. drc::get_waivers Command

Description

Returns a Design Assistant waiver for the rule, Compiler stages, owner, and query string that you specify. If you specify no option, `drc::get_waivers` returns all existing waivers.

Syntax

```
drc::get_waivers
-owner <userid>
-rule <rule_id>
query_string
-stages <compiler stages>
[-tag <string>]
```

Arguments

owner User ID of waiver creator for audit trail.

rule Alpha-numeric rule ID.

query_string Query string specifying violation column arguments to define the violation patterns the waiver ignores. The query string must exactly match the `query_string` you specify during waiver creation with the `add_waiver` command.

stages Subset of Compiler stage(s) to which the rule waiver applies.

tag Short text description for tracking different types of violations across the entire project.

2.5.5.5.7. drc::report_waivers Command

Description

Returns a collection of Design Assistant waivers for the rule, Compiler stages, and owner that you specify, or adds those waivers to a waiver file that you specify.

Syntax

```
drc::report_waivers
-file <waiver_file>
-owner <userid>
-rule <rule_id>
-stages <compiler stages>
[-tag <string>]
```

Arguments

file A waiver output file that design assistant creates and appends the returned waivers.

owner User ID of waiver creator for audit trail.

rule Alpha-numeric rule ID.

stages Subset of Compiler stage(s) to which the rule waiver applies.

tag Short text description for tracking different types of violations across the entire project.

2.5.5.6. Design Assistant Tags

Different Design Assistant **Tags** apply to each rule to extend search or filter capabilities based on the following facets of the rule. Refer to the **Design Assistant Rule Settings** to view which tags apply to each rule.

Table 7. Design Assistant Tags

Tag	Description
cdc-bus	Design rule checks related to topologies that use a bus to transfer multiple bits of data between clock domains at once.
clock-skew	Design rule checks related to clock skew.
design-partition	Design rule checks which check design partitions.
dsp	Design rule checks related to DSP blocks inside the FPGA fabric.
false-positive-synchronizer	Design rule checks related to automatically-detected synchronizer chains that may have been over-zealously detected.
global-signal	Design rule checks related to global signals.
impossible-requirements	Design rule checks which check the requirements on failing timing paths and flag those which fail by construction.

continued...

Tag	Description
ip-parameterization	Design rule checks which look for parameterizable IP modules which may need to be adjusted to meet performance specifications.
intrinsic-margin	Design rule checks which use the Intrinsic Margin metric (slack ignoring cell delay, IC delay and clock skew) to diagnose potential timing issues on failing paths.
latch	Design rule checks related to latches.
logic-levels	Design rule checks which flag potentially problematic amounts of logic on a timing path.
minimum-pulse-width	Design rule checks related to minimum pulse width.
nonstandard-timing	Design rule checks related to topologies which have unique timing analysis methodologies and may prove problematic.
partial-reconfiguration	Design rule checks which check Partial Reconfiguration designs.
place	Design rule checks which pertain to the Compiler's Place stage.
project-settings	Design rule checks related to validating the project settings.
ram	Design rule checks related to M20k blocks inside the FPGA fabric.
region-constraints	Design rule checks related to region constraints in the design (both placement and routing).
register-duplication	Design rule checks related to duplication of registers in the design, either manually or automatically.
register-spread	Design rule checks related to measuring the spread of a register's sinks, as found in the "Report Register Spread" command.
reset-usage	Design rule checks related to safe resets or appropriate use of reset modes.
reset-reachability	Design rule checks related to reachability analysis of reset signals, including convergence of multiple reset signals.
resource-usage	Design rule checks related to managing the resource usage of the design.
retime	Design rule checks which pertain to the Compiler's Retime stage.
route	Design rule checks which pertain to the Compiler's Route stage.
sdc	Design rule checks related to SDC validity checking.
synchronizer	Design rule checks related to synchronizer chains.
synthesis	Design rule checks which pertain to the Compiler's Analysis & Synthesis stage.
system	Design rule checks which validate full-system design.

2.5.6. Design Assistant Rule Categories

Each Design Assistant rule has a unique alphanumeric ID that reflects the rule category. You can enable or disable Design Assistant rules for specific stages of compilation. The following lists all categories of Design Assistant rules, and provides a link to rule documentation in Intel Quartus Prime Help. Some categories are device-specific.

Table 8. Design Assistant Rule Categories

Rule Category and Help Link	Acronym	Description
Clock Domain Crossing Rules	CDC	Rule checks for clock domain crossings
Clock Rules	CLK	Rule checks for clocks

continued...

Rule Category and Help Link	Acronym	Description
Floorplanning Rules	FLP	Rule checks for floorplanning, such as Logic Lock regions and congestion
Linting Rules	LNT	Rules checks of source code for programmatic and stylistic errors
Project Rules	PRJ	Rule checks related to Intel Quartus Prime projects
Reset Domain Crossing	RDC	Rule checks for reset domain crossings
Reset Rules	RES	Rule checks for resets
Timing Closure Rules	TMC	Rule checks for timing closure

Related Information

Design Assistant Rules Help

2.6. Recommended Design Practices Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2023.04.14	23.1	• Updated the recommendation in *Optimizing Clocking Schemes*.
2022.06.20	22.2	• Removed obsolete Block Based Design rule category from *Design Assistant Design Rule Checking* topic.
2021.10.04	21.3	• Updated *Design Assistant Design Rule Checking* topic for latest rule categories.
2021.03.29	21.1	• Updated *Design Assistant Design Rule Checking* topic with screenshot, minor wording changes, and link to *AN 919: Improving Quality of Results with Design Assistant*. • Updated *Setting Up Design Assistant* topic for rule tags. • Updated *Running Design Assistant During Compilation* step 4 wording. • Updated *Opening Design Assistant Rule Help* topic to show rule included in this release. • Updated screenshots in *Cross-Probing from Design Assistant to Timing Analyzer* topic. • Removed *Filtering and Hiding Rule Violations* section as waivers are most effective method in current version. • Updated *Changing the Default Number of Violations per Rule* for minor wording changes. • Created new *Design Assistant Tags* topic to define all tags. • Created new *Design Assistant Waiver Dialog Box* topic to define all settings in this dialog box. including new waiver features. • Updated *Creating Design Assistant Waivers* topic for waiver preview and reporting. • Updated *Deleting Design Assistant Waivers* topic for new GUI method. • Updated *Design Assistant Rule Categories* topic for latest rule categories. • Revised *Design Assistant Rule Severity Levels* descriptions.
2020.09.28	20.3	• Added new "Waiving Design Assistant Rules" section. • Added new "Cross-Probing with Design Assistant" section. • Added description of new Fatal severity level Design Assistant rule to "Modifying Rule Severity Levels" topic. • Updated "Setting Up Design Assistant" for new Fatal severity level and RDC rule category.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Updated "Running Design Assistant in Analysis Mode" with screenshots and detailed steps. • Added RDC to "Design Assistant Rule Categories" topic. • Removed individual Design Assistant rule descriptions from document and linked to updated rule description in Help. • Replaced obsolete rule screenshots throughout.
2020.04.13	20.1	• Added support for Design Assistant during Timing Signoff to "Setting Up Design Assistant" and "Running Design Assistant During Compilation" topics. • Added new Design Assistant rules. • Revised Design Assistant rules HRR-10201, HRR-10201, and RES-30131. • Removed various obsolete Design Assistant rules.
2019.11.01	19.3.0	• Revised "Changing Design Assistant Rule Scope or Number of Violations" topic for clarity. • Created separate "Clock Domain Crossing (CDC) Rules" category and topic. • Added "CLK-30026: Missing Clock Assignment" Design Assistant rule. • Added "CLK-30027: Multiple Clock Assignment" Design Assistant rule. • Added "CLK-30028: Invalid Generated Clock" Design Assistant rule. • Added "CLK-30029: Invalid Clock Assignment" Design Assistant rule. • Added "CLK-30030: PLL Setting Violation" Design Assistant rule. • Added "CLK-30031: Input Delay Assigned to Clock" Design Assistant rule.
2019.09.30	19.3.0	• Added new "Setting Up Design Assistant" topic. • Added new "Managing Design Assistant Rules" topic. • Added new "Enabling Rules for Specific Compiler Stages" topic. • Added new "Specifying Rule Parameters for Specific Compiler Stages" topic. • Added new "Modifying Rule Severity Levels" topic. • Added new "Filtering and Hiding Rule Violations" topic. • Added new "Filter Options Dialog Box" topic. • Added new "Linking to Design Assistant Rule Documentation" topic.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Updated screenshots for latest GUI elements. • Added the following new Design Assistant rules: — CDC-50001: Missing 1-Bit Synchronizer — CDC-50002: 1-Bit Synchronized Missing Constraint — CDC-50003: CE-Type CDC No Constraints — HRR-10015: High Fan-out Signal — HRR-10201: Registers Cannot Power Up with Don't Care Logic Level — HRR-10204: Reset Release Instance Count Check — RES-30132: Registers May Not Be Properly Reset — TMC-20500: Hierarchical Tree Duplication was Shallower than Possible — TMC-20501: Hierarchical Tree Duplication was Shallower than Requested — TMC-20550: Duplication Candidate Rejected for Placement Constraint — TMC-20551: Automatically-Discovered Duplication Candidate Likely Requires More Duplication — TMC-20552: User-Directed Duplication Candidate was Rejected — TMC-20601: Registers with High Immediate Fan-Out Tension — TMC-20602: Registers with High Timing Path Endpoint Tension — TMC-20603: Registers with High Immediate Fan-Out Span — TMC-20604: Registers with High Timing Path Endpoint Span • Removed obsolete Design Assistant rules: — HRR-10014: High Fan-out Nets Driving Clock-Enable Pins — HRR-10016: Registers Cannot Power-Up With Dont Care Logic Level
2019.04.01	19.1.0	• Described new Design Assistant design rule checking tool. • Added new topics describing each of the Design Assistant rules, under the *Recommended Design Practices > Checking for Design Rule Violations* section.
2018.09.24	18.1.0	• Created subtopics: *Clock Region Assignments in Intel Arria 10 and Older Device Families* and *Clock Region Assignments in Intel Stratix 10 Devices* from content in topic: *Use Clock Region Assignments to Optimize Clock Constraints*.
2017.11.06	17.1.0	• Updated topic: *Optimizing Timing Closure*. • Updated topic *Use Global Clock Network Resources* and added topic *Use Clock Region Assignments to Optimize Clock Constraints* for Intel Stratix 10 support.
2017.05.08	17.0.0	• Removed information about Integrated Synthesis. • Removed information about quartus_drc.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	• Replaced Internally Synchronized Reset code sample with corrected version. • Removed information about deprecated physical synthesis options. • Removed information about unsupported Design Assistant.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
June 2014	14.0.0	Removed references to obsolete MegaWizard Plug-In Manager.
November 2013	13.1.0	Removed HardCopy device information.
May 2013	13.0.0	Removed PrimeTime support.

continued...

Document Version	Intel Quartus Prime Version	Changes
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	Added information to Reset Resources .
December 2010	10.1.0	• Title changed from Design Recommendations for Altera Devices and the Quartus II Design Assistant. • Updated to new template. • Added references to Quartus II Help for "Metastability" on page 9–13 and "Incremental Compilation" on page 9–13. • Removed duplicated content and added references to Help for "Custom Rules" on page 9–15.
July 2010	10.0.0	• Removed duplicated content and added references to Quartus II Help for Design Assistant settings, Design Assistant rules, Enabling and Disabling Design Assistant Rules, and Viewing Design Assistant reports. • Removed information from "Combinational Logic Structures" on page 5–4 • Changed heading from "Design Techniques to Save Power" to "Power Optimization" on page 5–12 • Added new "Metastability" section • Added new "Incremental Compilation" section • Added information to "Reset Resources" on page 5–23 • Removed "Referenced Documents" section
November 2009	9.1.0	• Removed documentation of obsolete rules.
March 2009	9.0.0	• No change to content.
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size • Added new section "Custom Rules Coding Examples" on page 5–18 • Added paragraph to "Recommended Clock-Gating Methods" on page 5–11 • Added new section: "Design Techniques to Save Power" on page 5–12
May 2008	8.0.0	• Updated Figure 5–9 on page 5–13; added custom rules file to the flow • Added notes to Figure 5–9 on page 5–13 • Added new section: "Custom Rules Report" on page 5–34 • Added new section: "Custom Rules" on page 5–34 • Added new section: "Targeting Embedded RAM Architectural Features" on page 5–38 • Minor editorial updates throughout the chapter • Added hyperlinks to referenced documents throughout the chapter

3. Managing Metastability with the Intel Quartus Prime Software

You can use the Intel Quartus Prime software to analyze the average mean time between failures (MTBF) due to metastability caused by synchronization of asynchronous signals, and optimize the design to improve the metastability MTBF.

All registers in digital devices, such as FPGAs, have defined signal-timing requirements that allow each register to correctly capture data at its input ports and produce an output signal. To ensure reliable operation, the input to a register must be stable for a minimum amount of time before the clock edge (register setup time or t_{SU}) and a minimum amount of time after the clock edge (register hold time or t_H). The register output is available after a specified clock-to-output delay (t_{CO}).

If the data violates the setup or hold time requirements, the output of the register might go into a metastable state. In a metastable state, the voltage at the register output hovers at a value between the high and low states, which means the output transition to a defined high or low state is delayed beyond the specified t_{CO}. Different destination registers might capture different values for the metastable signal, which can cause the system to fail.

In synchronous systems, the input signals must always meet the register timing requirements, so that metastability does not occur. Metastability problems commonly occur when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the signal can arrive at any time relative to the destination clock.

The MTBF due to metastability is an estimate of the average time between instances when metastability could cause a design failure. A high MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design. You should determine an acceptable target MTBF in the context of your entire system and taking in account that MTBF calculations are statistical estimates.

The metastability MTBF for a specific signal transfer, or all the transfers in a design, can be calculated using information about the design and the device characteristics. Improving the metastability MTBF for your design reduces the chance that signal transfers could cause metastability problems in your device.

The Intel Quartus Prime software provides analysis, optimization, and reporting features to help manage metastability in Intel designs. These metastability features are supported only for designs constrained with the Intel Quartus Prime Timing Analyzer. Both typical and worst-case MBTF values are generated for select device families.

3.1. Metastability Analysis in the Intel Quartus Prime Software

When a signal transfers between circuitry in unrelated or asynchronous clock domains, the first register in the new clock domain acts as a synchronization register.

To minimize the failures due to metastability in asynchronous signal transfers, circuit designers typically use a sequence of registers (a synchronization register chain or synchronizer) in the destination clock domain to resynchronize the signal to the new clock domain and allow additional time for a potentially metastable signal to resolve to a known value. Designers commonly use two registers to synchronize a new signal, but a standard of three registers provides better metastability protection.

The timing analyzer can analyze and report the MTBF for each identified synchronizer that meets its timing requirements, and can generate an estimate of the overall design MTBF. The software uses this information to optimize the design MTBF, and you can use this information to determine whether your design requires longer synchronizer chains.

Related Information

- Metastability and MTBF Reporting on page 124
- MTBF Optimization on page 127

3.1.1. Data Synchronization Register Chains

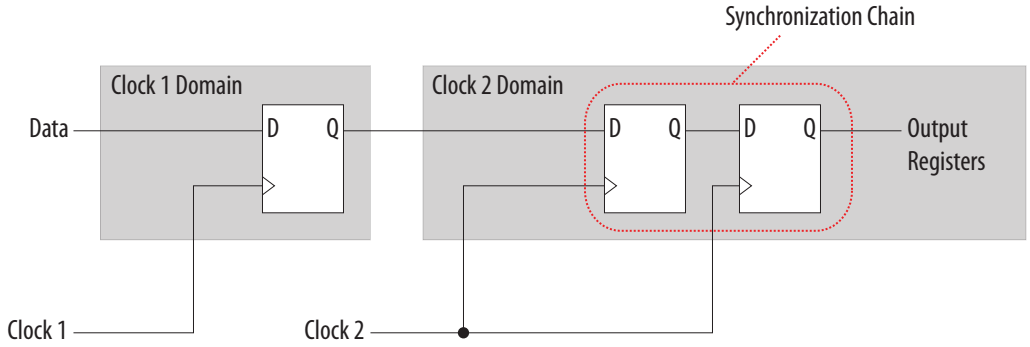
A synchronization register chain, or synchronizer, is defined as a sequence of registers that meets the following requirements:

- The registers in the chain are all clocked by the same clock or phase-related clocks.
- The first register in the chain is driven asynchronously or from an unrelated clock domain.
- Each register fans out to only one register, except the last register in the chain.

For Intel Quartus Prime software to identify a synchronization register chain, the registers in the chain must not include any resets.

The length of the synchronization register chain is the number of registers in the synchronizing clock domain that meet the requirements listed earlier. The following figure shows a sample two-register synchronization chain.

Figure 47. Sample Synchronization Register Chain



The timing slack available in the register-to-register paths of the synchronizer allows a metastable signal to settle, and is referred to as the available settling time. The available settling time in the MTBF calculation for a synchronizer is the sum of the output timing slacks for each register in the chain. Adding available settling time with additional synchronization registers improves the metastability MTBF.

Related Information

- How Timing Constraints Affect Synchronizer Identification and Metastability Analysis on page 123
- Force the Identification of Synchronization Registers on page 129
- Identify Synchronizers for Metastability Analysis on page 123

3.1.2. Identify Synchronizers for Metastability Analysis

The first step in enabling metastability MTBF analysis and optimization in the Intel Quartus Prime software is to identify which registers are part of a synchronization register chain. You can apply synchronizer identification settings globally to automatically list possible synchronizers with the **Synchronizer identification** option on the **Timing Analyzer** page in the **Settings** dialog box.

Synchronization chains are already identified within most Intel FPGA intellectual property (IP) cores.

Related Information

Force the Identification of Synchronization Registers on page 129

3.1.3. How Timing Constraints Affect Synchronizer Identification and Metastability Analysis

The timing analyzer can analyze metastability MTBF only if a synchronization chain meets its timing requirements. The metastability failure rate depends on the timing slack available in the synchronizer's register-to-register connections, because that slack is the available settling time for a potential metastable signal. Therefore, you must ensure that your design is correctly constrained with the real application frequency requirements to get an accurate MTBF report.

In addition, the **Auto** and **Forced If Asynchronous** synchronizer identification options use timing constraints to automatically detect the synchronizer chains in the design. These options check for signal transfers between circuitry in unrelated or asynchronous clock domains, so clock domains must be related correctly with timing constraints.

The timing analyzer views input ports as asynchronous signals unless they are associated correctly with a clock domain. If an input port fans out to registers that are not acting as synchronization registers, apply a `set_input_delay` constraint to the input port; otherwise, the input register might be reported as a synchronization register. Constraining a synchronous input port with a `set_max_delay` constraint for a setup (t_{SU}) requirement does not prevent synchronizer identification because the constraint does not associate the input port with a clock domain.

Instead, use the following command to specify an input setup requirement associated with a clock:

`set_input_delay -max -clock` *<clock name> <latch – launch – tsu requirement> <input port name>*

Registers that are at the end of false paths are also considered synchronization registers because false paths are not timing-analyzed. Because there are no timing requirements for these paths, the signal may change at any point, which may violate the t_{SU} and t_H of the register. Therefore, these registers are identified as

synchronization registers. If these registers are not used for synchronization, you can turn off synchronizer identification and analysis. To do so, set **Synchronizer Identification** to **Off** for the first synchronization register in these register chains.

3.2. Metastability and MTBF Reporting

The Intel Quartus Prime software reports the metastability analysis results in the Compilation Report and Timing Analyzer reports.

The MTBF calculation uses timing and structural information about the design, silicon characteristics, and operating conditions, along with the data toggle rate.

If you change the **Synchronizer Identification** settings, you can generate new metastability reports by rerunning the timing analyzer. However, you should rerun the Fitter first so that the registers identified with the new setting can be optimized for metastability MTBF.

Related Information

- Metastability Reports on page 124
- MTBF Optimization on page 127
- Synchronizer Data Toggle Rate in MTBF Calculation on page 126

3.2.1. Metastability Reports

Metastability reports summarize the results of the metastability analysis. In addition to the MTBF Summary and Synchronizer Summary reports, the Timing Analyzer tool reports additional statistics for each synchronizer chain.

If the design uses only the **Auto Synchronizer Identification** setting, the reports list likely synchronizers but do not report MTBF. To obtain an MTBF for each register chain you must force identification of synchronization registers.

If the synchronizer chain does not meet its timing requirements, the reports list identified synchronizers but do not report MTBF. To obtain MTBF calculations, ensure that the design is constrained correctly, and that the synchronizer meets its timing requirements.

Related Information

- Identify Synchronizers for Metastability Analysis on page 123
- How Timing Constraints Affect Synchronizer Identification and Metastability Analysis on page 123

3.2.1.1. MTBF Summary Report

The MTBF Summary reports an estimate of the overall robustness of cross-clock domain and asynchronous transfers in the design. This estimate uses the MTBF results of all synchronization chains in the design to calculate an MTBF for the entire design.

3.2.1.1.1. Typical and Worst-Case MTBF of Design

The MTBF Summary Report shows the **Typical MTBF of Design** and the **Worst-Case MTBF of Design** for supported fully-characterized devices. The typical MTBF result assumes typical conditions, defined as nominal silicon characteristics for the selected device speed grade, as well as nominal operating conditions. The worst-case MTBF result uses the worst case silicon characteristics for the selected device speed grade.

When you analyze multiple timing corners in the timing analyzer, the MTBF calculation may vary because of changes in the operating conditions, and the timing slack or available metastability settling time. Intel recommends running multi-corner timing analysis to ensure that you analyze the worst MTBF results, because the worst timing corner for MTBF does not necessarily match the worst corner for timing performance.

Related Information

Timing Analyzer
In *Intel Quartus Prime Help*

3.2.1.1.2. Synchronizer Chains

The MTBF Summary report also lists the **Number of Synchronizer Chains Found** and the length of the **Shortest Synchronizer Chain**, which can help you identify whether the report is based on accurate information.

If the number of synchronizer chains found is different from what you expect, or if the length of the shortest synchronizer chain is less than you expect, you might have to add or change **Synchronizer Identification** settings for the design. The report also provides the **Worst Case Available Settling Time**, defined as the available settling time for the synchronizer with the worst MTBF.

You can use the reported **Fraction of Chains for which MTBFs Could Not be Calculated** to determine whether a high proportion of chains are missing in the metastability analysis. A fraction of 1, for example, means that MTBF could not be calculated for any chains in the design. MTBF is not calculated if you have not identified the chain with the appropriate **Synchronizer identification** option, or if paths are not timing-analyzed and therefore have no valid slack for metastability analysis. You might have to correct your timing constraints to enable complete analysis of the applicable register chains.

3.2.1.1.3. Increasing Available Settling Time

The MTBF Summary report specifies how an increase of 100ps in available settling time increases the MTBF values. If your MTBF is not satisfactory, this metric can help you determine how much extra slack would be required in your synchronizer chain to allow you to reach the desired design MTBF.

3.2.1.2. Synchronizer Summary Report

The **Synchronizer Summary** lists the synchronization register chains detected in the design depending on the Synchronizer Identification setting.

The **Source Node** is the register or input port that is the source of the asynchronous transfer. The **Synchronization Node** is the first register of the synchronization chain. The **Source Clock** is the clock domain of the source node, and the **Synchronization Clock** is the clock domain of the synchronizer chain.

This summary reports the calculated **Worst-Case MTBF**, if available, and the **Typical MTBF**, for each appropriately identified synchronization register chain that meets its timing requirement.

Related Information

Synchronizer Chain Statistics Report in the Timing Analyzer on page 126

3.2.1.3. Synchronizer Chain Statistics Report in the Timing Analyzer

The timing analyzer provides an additional report for each synchronizer chain.

The **Chain Summary** tab matches the Synchronizer Summary information described in the Synchronizer Summary Report, while the **Statistics** tab adds more details. These details include whether the **Method of Synchronizer Identification** was **User Specified** (with the **Forced if Asynchronous** or **Forced** settings for the **Synchronizer Identification** setting), or **Automatic** (with the **Auto** setting). The **Number of Synchronization Registers in Chain** report provides information about the parameters that affect the MTBF calculation, including the **Available Settling Time** for the chain and the **Data Toggle Rate Used in MTBF Calculation**.

The following information is also included to help you locate the chain in your design:

- **Source Clock** and **Asynchronous Source** node of the signal.
- **Synchronization Clock** in the destination clock domain.
- Node names of the **Synchronization Registers** in the chain.

Related Information

Synchronizer Data Toggle Rate in MTBF Calculation on page 126

3.2.2. Synchronizer Data Toggle Rate in MTBF Calculation

The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency. That is, the arriving data is assumed to switch once every eight source clock cycles.

If multiple clocks apply, the highest frequency is used. If no source clocks can be determined, the data rate is taken as 12.5% of the synchronization clock frequency.

If you know an approximate rate at which the data changes, specify it with the **Synchronizer Toggle Rate** assignment in the Assignment Editor. You can also apply this assignment to an entity or the entire design. Set the data toggle rate, in number of transitions per second, on the first register of a synchronization chain. The timing analyzer takes the specified rate into account when computing the MTBF of that register chain. If a data signal never toggles and does not affect the reliability of the design, you can set the **Synchronizer Toggle Rate** to **0** for the synchronization chain so the MTBF is not reported. To apply the assignment with Tcl, use the following command:

```
set_instance_assignment -name SYNCHRONIZER_TOGGLE_RATE <toggle rate in
transitions/second> -to <register name>
```

In addition to **Synchronizer Toggle Rate,** there are two other assignments associated with toggle rates, which are not used for metastability MTBF calculations. The I/O Maximum Toggle Rate is only used for pins, and specifies the worst-case

toggle rates used for signal integrity purposes. The Power Toggle Rate assignment is used to specify the expected time-averaged toggle rate, and is used by the Power Analyzer to estimate time-averaged power consumption.

3.3. MTBF Optimization

In addition to reporting synchronization register chains and MTBF values found in the design, the Intel Quartus Prime software can also protect these registers from optimizations that might negatively impact MTBF and can optimize the register placement and routing if the MTBF is too low.

Synchronization register chains must first be explicitly identified as synchronizers. Intel recommends that you set **Synchronizer Identification** to **Forced If Asynchronous** for all registers that are part of a synchronizer chain.

Optimization algorithms, such as register duplication and logic retiming in physical synthesis, are not performed on identified synchronization registers. The Fitter protects the number of synchronization registers specified by the **Synchronizer Register Chain Length** option.

In addition, the Fitter optimizes identified synchronizers for improved MTBF by placing and routing the registers to increase their output setup slack values. Adding slack in the synchronizer chain increases the available settling time for a potentially metastable signal, which improves the chance that the signal resolves to a known value, and exponentially increases the design MTBF. The Fitter optimizes the number of synchronization registers specified by the **Synchronizer Register Chain Length** option.

Metastability optimization is **on** by default. To view or change the **Optimize Design for Metastability** option, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**. To turn the optimization on or off with Tcl, use the following command:

```
set_global_assignment -name OPTIMIZE_FOR_METASTABILITY <ON|OFF>
```

Related Information

Identify Synchronizers for Metastability Analysis on page 123

3.3.1. Synchronization Register Chain Length

The **Synchronization Register Chain Length** option specifies how many registers should be protected from optimizations that might reduce MTBF for each register chain, and controls how many registers should be optimized to increase MTBF with the **Optimize Design for Metastability** option.

For example, if the **Synchronization Register Chain Length** option is set to **2**, optimizations such as register duplication or logic retiming are prevented from being performed on the first two registers in all identified synchronization chains. The first two registers are also optimized to improve MTBF when the **Optimize Design for Metastability** option is turned on.

The default setting for the **Synchronization Register Chain Length** option is **3**. The first register of a synchronization chain is always protected from operations that might reduce MTBF, but you should set the protection length to protect more of the

synchronizer chain. Intel recommends that you set this option to the maximum length of synchronization chains you have in your design so that all synchronization registers are preserved and optimized for MTBF.

Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** to change the global **Synchronization Register Chain Length** option.

You can also set the **Synchronization Register Chain Length** on a node or an entity in the Assignment Editor. You can set this value on the first register in a synchronization chain to specify how many registers to protect and optimize in this chain. This individual setting is useful if you want to protect and optimize extra registers that you have created in a specific synchronization chain that has low MTBF, or optimize less registers for MTBF in a specific chain where the maximum frequency or timing performance is not being met. To make the global setting with Tcl, use the following command:

```
set_global_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers>
```

To apply the assignment to a design instance or the first register in a specific chain with Tcl, use the following command:

```
set_instance_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers> -to <register or instance name>
```

3.4. Reducing Metastability Effects

You can check your design's metastability MTBF in the Metastability Summary report, and determine an acceptable target MTBF given the context of your entire system and the fact that MTBF calculations are statistical estimates. A high metastability MTBF (such as hundreds or thousands of years between metastability failures) indicates a more robust design.

This section provides guidelines to ensure complete and accurate metastability analysis, and some suggestions to follow if the Intel Quartus Prime metastability reports calculate an unacceptable MTBF value.

Related Information

Metastability Reports on page 124

3.4.1. Apply Complete System-Centric Timing Constraints for the Timing Analyzer

To enable the Intel Quartus Prime metastability features, make sure that the timing analyzer is used for timing analysis.

Ensure that the design is fully timing constrained and that it meets its timing requirements. If the synchronization chain does not meet its timing requirements, MTBF cannot be calculated. If the clock domain constraints are set up incorrectly, the signal transfers between circuitry in unrelated or asynchronous clock domains might be identified incorrectly.

Use industry-standard system-centric I/O timing constraints instead of using FPGA-centric timing constraints.

You should use `set_input_delay` constraints in place of `set_max_delay` constraints to associate each input port with a clock domain to help eliminate false positives during synchronization register identification.

Related Information

How Timing Constraints Affect Synchronizer Identification and Metastability Analysis on page 123

3.4.2. Force the Identification of Synchronization Registers

Use the guidelines in *Identifying Synchronizers for Metastability Analysis* to ensure the software reports and optimizes the appropriate register chains.

Identify synchronization registers by setting **Synchronizer Identification** to **Forced If Asynchronous** in the Assignment Editor. If there are any registers that the software detects as synchronous, but you want to analyze for metastability, apply the **Forced** setting to the first synchronizing register. Set **Synchronizer Identification** to **Off** for registers that are not synchronizers for asynchronous signals or unrelated clock domains.

To help you find the synchronizers in your design, you can set the global **Synchronizer Identification** setting on the **Timing Analyzer** page of the **Settings** dialog box to **Auto** to generate a list of all the possible synchronization chains in your design.

Related Information

Identify Synchronizers for Metastability Analysis on page 123

3.4.3. Set the Synchronizer Data Toggle Rate

The MTBF calculations assume the data being synchronized is switching at a toggle rate of 12.5% of the source clock frequency.

To obtain a more accurate MTBF for a specific chain or all chains in your design, set the **Synchronizer Toggle Rate**.

Related Information

Synchronizer Data Toggle Rate in MTBF Calculation on page 126

3.4.4. Optimize Metastability During Fitting

Ensure that the **Optimize Design for Metastability** setting is turned on.

Related Information

MTBF Optimization on page 127

3.4.5. Increase the Length of Synchronizers to Protect and Optimize

Increase the Synchronizer Chain Length parameter to the maximum length of synchronization chains in your design. If you have synchronization chains longer than 2 identified in your design, you can protect the entire synchronization chain from operations that might reduce MTBF and allow metastability optimizations to improve the MTBF.

Related Information

Synchronization Register Chain Length on page 127

3.4.6. Increase the Number of Stages Used in Synchronizers

Designers commonly use two registers in a synchronization chain to minimize the occurrence of metastable events, and a standard of three registers provides better metastability protection. However, synchronization chains with two or even three registers may not be enough to produce a high enough MTBF when the design runs at high clock and data frequencies.

If a synchronization chain is reported to have a low MTBF, consider adding an additional register stage to your synchronization chain. This additional stage increases the settling time of the synchronization chain, allowing more opportunity for the signal to resolve to a known state during a metastable event. Additional settling time increases the MTBF of the chain and improves the robustness of your design. However, adding a synchronization stage introduces an additional stage of latency on the signal.

If you use the Intel FPGA IP core with separate read and write clocks to cross clock domains, increase the metastability protection (and latency) for better MTBF. In the DCFIFO parameter editor, choose the **Best metastability protection, best fmax, unsynchronized clocks** option to add three or more synchronization stages. You can increase the number of stages to more than three using the **How many sync stages?** setting.

3.4.7. Select a Faster Speed Grade Device

The design MTBF depends on process parameters of the device used. Faster devices are less susceptible to metastability issues. If the design MTBF falls significantly below the target MTBF, switching to a faster speed grade can improve the MTBF substantially.

3.5. Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run procedures at a command prompt.

For detailed information about scripting command options, refer to the Intel Quartus Prime Command-Line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt and then press Enter:

```
quartus_sh --qhelp
```

Related Information

- Intel Quartus Prime Pro Edition Settings File Reference Manual
- Intel Quartus Prime Pro Edition User Guide: Scripting

3.5.1. Identifying Synchronizers for Metastability Analysis

To apply the global Synchronizer Identification assignment, use the following command:

```
set_global_assignment -name SYNCHRONIZER_IDENTIFICATION <OFF|AUTO|"FORCED IF
ASYNCHRONOUS">
```

To apply the **Synchronizer Identification** assignment to a specific register or instance, use the following command:

```
set_instance_assignment -name SYNCHRONIZER_IDENTIFICATION <AUTO|"FORCED IF
ASYNCHRONOUS"|FORCED|OFF> -to <register or instance name>
```

3.5.2. Synchronizer Data Toggle Rate in MTBF Calculation

To specify a toggle rate for MTBF calculations as described on page "R**Synchronizer Data Toggle Rate in MTBF Calculation", use the following command:

```
set_instance_assignment -name SYNCHRONIZER_TOGGLE_RATE <toggle rate in
transitions/second> -to <register name>
```

Related Information

Synchronizer Data Toggle Rate in MTBF Calculation on page 126

3.5.3. report_metastability and Tcl Command

If you use a command-line or scripting flow, you can generate the metastability analysis reports described in "C**Metastability Reports" outside of the Intel Quartus Prime and user interfaces.

The table describes the options for the `report_metastability` and Tcl command.

Table 9. report_metastabilty Command Options

Option	Description
`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten.
`-file <name>`	Sends the results to an ASCII or HTML file. The extension specified in the file name determines the file type — either **`*.txt`** or **`*.html`**.
`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel.
`-stdout`	Indicates the report be sent to the standard output, via messages. This option is required only if you have selected another output format, such as a file, and would also like to receive messages.

Related Information

Metastability Reports on page 124

3.5.4. MTBF Optimization

To ensure that metastability optimization described on page "C**MTBF Optimization" is turned on (or to turn it off), use the following command:

```
set_global_assignment -name OPTIMIZE_FOR_METASTABILITY <ON|OFF>
```

Related Information

MTBF Optimization on page 127

3.5.5. Synchronization Register Chain Length

To globally set the number of registers in a synchronization chain to be protected and optimized as described on page "C**Synchronization Register Chain Length", use the following command:

```
set_global_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers>
```

To apply the assignment to a design instance or the first register in a specific chain, use the following command:

```
set_instance_assignment -name SYNCHRONIZATION_REGISTER_CHAIN_LENGTH <number of registers> -to <register or instance name>
```

Related Information

Synchronization Register Chain Length on page 127

3.6. Managing Metastability

Intel's Intel Quartus Prime software provides industry-leading analysis and optimization features to help you manage metastability in your FPGA designs. Set up your Intel Quartus Prime project with the appropriate constraints and settings to enable the software to analyze, report, and optimize the design MTBF. Take advantage of these features in the Intel Quartus Prime software to make your design more robust with respect to metastability.

3.7. Managing Metastability with the Intel Quartus Prime Software Revision History

The following revisions history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2022.03.28	22.1	• Renamed "Synchronization Register Chains" to "Data Synchronization Register Chains" and revised the topic. • Removed references to obsolete Advisors.
2018.05.07	18.0.0	• First release as part of the stand-alone *Design Recommendations User Guide*
2017.11.06	17.1.0	• Corrected broken links to other documents.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
June 2014	14.0.0	Updated formatting.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Changed to new document template.

continued...

Document Version	Intel Quartus Prime Version	Changes
July 2010	10.0.0	Technical edit.
November 2009	9.1.0	Clarified description of synchronizer identification settings. Minor changes to text and figures throughout document.
March 2009	9.0.0	Initial release.

4. Intel Quartus Prime Pro Edition User Guide: Design Recommendations Archive

For the latest and previous versions of this user guide, refer to *Intel Quartus Prime Pro Edition User Guide: Design Recommendations*. If a software version is not listed, the user guide for the previous software version applies.

A. Intel Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Intel Quartus Prime Pro Edition FPGA design flow.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Intel Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Intel Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Intel Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Intel Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Intel Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Design Compilation

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

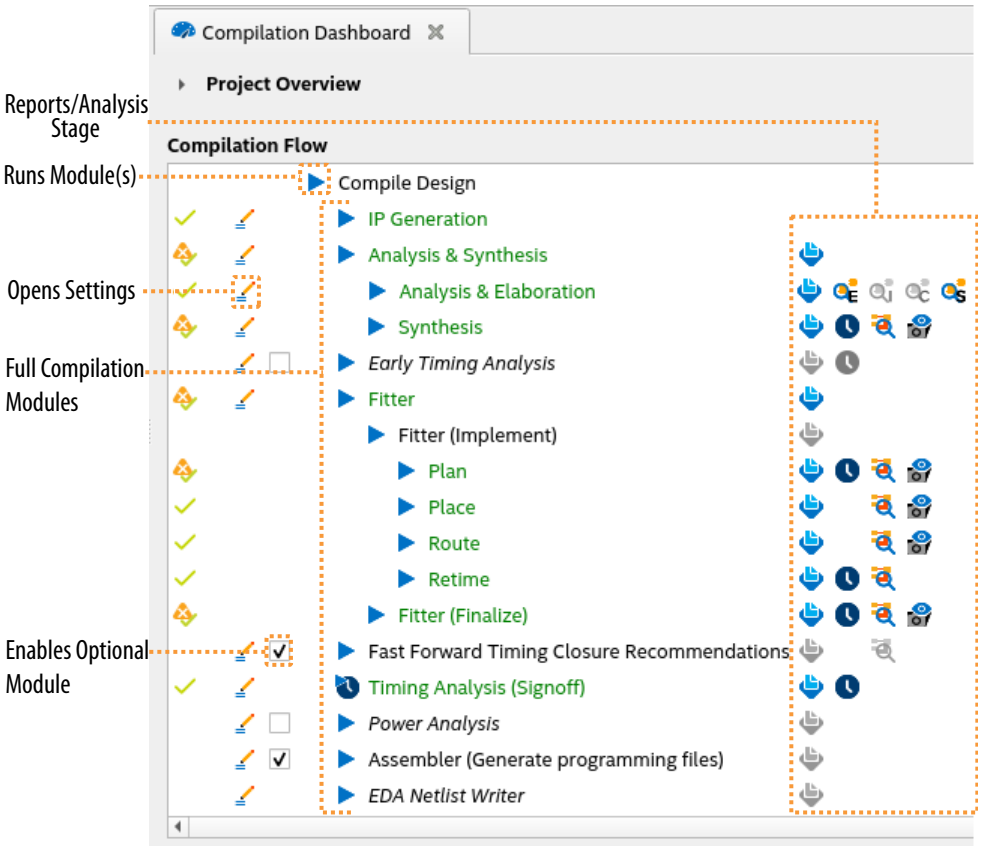
Answers to Top FAQs:

Contents

1. Design Compilation

The Quartus® Prime Compiler synthesizes, places, and routes your design before generating device programming files. The Compiler supports a variety of high-level, HDL, and schematic design entry methods. The modules of the Compiler include IP Generation, Analysis & Synthesis, Fitter, Timing Analyzer, and Assembler.

Figure 1. Compilation Dashboard



The Quartus Prime Pro Edition version of the Compiler supports these advanced features:

- Supports Arria® 10, Cyclone® 10 GX, Stratix® 10, and Agilex™ 7 devices.
- Incremental Fitter optimization—analyze and optimize after each Fitter stage to maximize performance and shorten total compilation time.
- Hyper-Aware Design Flow—use Hyper-Retiming and Fast Forward compilation for the highest performance in Stratix 10 and Agilex 7 devices.
- Partial Reconfiguration—dynamic reconfiguration of a portion of the FPGA, while the remaining FPGA continues to function.
- Block-Based Design Flows—preservation and reuse of design blocks.

1.1. Compilation Overview

The Compiler is modular, allowing you to run only the process that you need. Each Compiler module performs a specific function in the full compilation process. When you run any module, the Compiler runs any prerequisite modules automatically and generates detailed reports at each stage. The Compiler can preserve a "snapshot" of the compilation results after each stage.

Table 1. Compilation Modules

Compilation Process	Description
IP Generation	Identifies the status and version of IP components in the project, reports outdated IP that require upgrade, and generates Intel FPGA IPs in the project.
Analysis & Synthesis	• Analysis & Elaboration—a stage of Analysis & Synthesis that checks for design file and projects errors. It provides different checkpoints or previews (elaborated, instrumented, constrained, and swept) of your design early in the compilation flow and serves as a platform to better analyze your design and improve it. • Synthesis—Synthesizes, optimizes, minimizes, and maps design logic to device resources. The "synthesized" snapshot preserves the results of this stage.
Early Timing Analysis	Combines Synopsys* Design Constraint (SDC) on RTL and post-synthesis static timing analysis. The SDC-on-RTL allows you to integrate SDC constraints that target nodes using the same names as in your RTL design early in the compilation flow and uses them in the later stages of the Quartus Prime compilation. However, you can run the module even without RTL SDCs where you can view the synthesized timing netlist.
Fitter (Place & Route)	Assigns the placement and routing of the design to specific device resources, while honoring timing and placement constraints. The Fitter includes the following stages: • Plan—places all periphery elements (such as I/Os and PLLs) and determines a legal clock plan, without core placement or routing. The "planned" snapshot preserves the stage results. • Place—places all core elements in a legal location. The "placed" snapshot preserves the stage results. • Route—creates all routing between the elements in the design. The "routed" snapshot preserves the stage results. • Retime—moves (retimes) existing registers into Hyper-Registers for fine-grained performance improvement. The "retimed" snapshot preserves the stage results. (1) • Fitter (Finalize)—for Arria 10 and Cyclone 10 GX devices, converts unnecessary tiles to High-Speed or Low-Power. For Stratix 10 and Agilex 7 devices, performs post-Route fix-up. The "final" snapshot preserves the stage results.
Fast Forward Timing Closure Recommendations	Generates detailed reports that estimate performance gains achievable by making specific RTL modifications.

continued...

(1) Retiming and Fast Forward compilation available only for Stratix 10 and Agilex 7 devices.

Compilation Process	Description
Timing Analysis	Analyzes and validates the timing performance of all design logic with the Timing Analyzer.
Power Analysis	Optional module that estimates device power consumption. Specify the electrical standard on each I/O cell and the board trace model on each I/O standard in your design.
Assembler	Converts the Fitter's placement and routing assignments into a programming image for the FPGA device.
EDA Netlist Writer	Generates output files for use in other EDA tools, as Integrating Other EDA Tools on page 147 describes.

Note: Each successive release of the Quartus Prime software typically includes:

- Added support for new features in supported FPGA devices.
- Added support for new devices.
- Efficiency and performance improvements.
- Improvements to compilation time and resource use of the design software.

Due to these improvements, different versions of the Quartus Prime Pro Edition, Quartus Prime Standard Edition, and Quartus Prime Lite Edition software can produce different programming files from release to release.

1.1.1. Compilation Flows

The Quartus Prime Pro Edition Compiler supports a variety of flows to help you maximize performance and minimize compilation processing time. The modular Compiler is flexible and efficient, allowing you to run all modules in sequence with a single command, or to run and optimize each stage of compilation separately.

As you develop and optimize your design, run only the Compiler stages that you need, rather than waiting for full compilation. Run full compilation only when your design is complete and you are ready to run all Compiler modules and generate a device programming image.

Table 2. Compilation Flows

Compiler Flow	Function
Full Compilation Flow	Launches all Compiler modules in sequence to synthesize, fit, analyze final timing, and generate a device programming file. By default, the Compiler generates and preserves only the synthesized and final snapshots during a full compilation. You can optionally **Enable Intermediate Fitter Snapshots** to preserve the planned, placed, routed, and retimed snapshots.
Block-Based Design Flows	Supports preservation and reuse of design blocks in one or more projects. You can reuse synthesized or final design blocks in other projects. Reusable design blocks can include device core or periphery resources.
Incremental Optimization Flow	Incremental optimization allows you to stop processing after each Fitter stage, analyze the results, and adjust settings or RTL before proceeding to the next compilation stage. This iterative flow optimizes at each stage, without waiting for full compilation results.

continued...

Compiler Flow	Function
Hyper-Aware Design Flow	Combines automated register retiming (Hyper-Retiming), with implementation of targeted timing closure recommendations (Fast Forward Compilation), to maximize use of Hyper-Registers and drive the highest performance in Stratix 10 and Agilex 7 devices.
Partial Reconfiguration	Reconfigures a portion of the FPGA dynamically, while the remaining FPGA design continues to function.
ECO Compilation Flow	The Quartus Prime Pro Edition software supports last-minute, targeted design changes (also known as engineering change orders (ECOs)), even after you fully compile the design. ECOs typically occur during the design verification stage. Refer to the *Quartus Prime Pro Edition User Guide: Design Optimization*.

Related Information

- Incremental Optimization Flow on page 105
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
- Full Compilation Flow on page 132
- Running the Fitter on page 96
- Fast Forward Compilation Flow on page 116
- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration
- Intel Quartus Prime Pro Edition User Guide: Design Optimization

1.1.2. Compilation Hierarchy

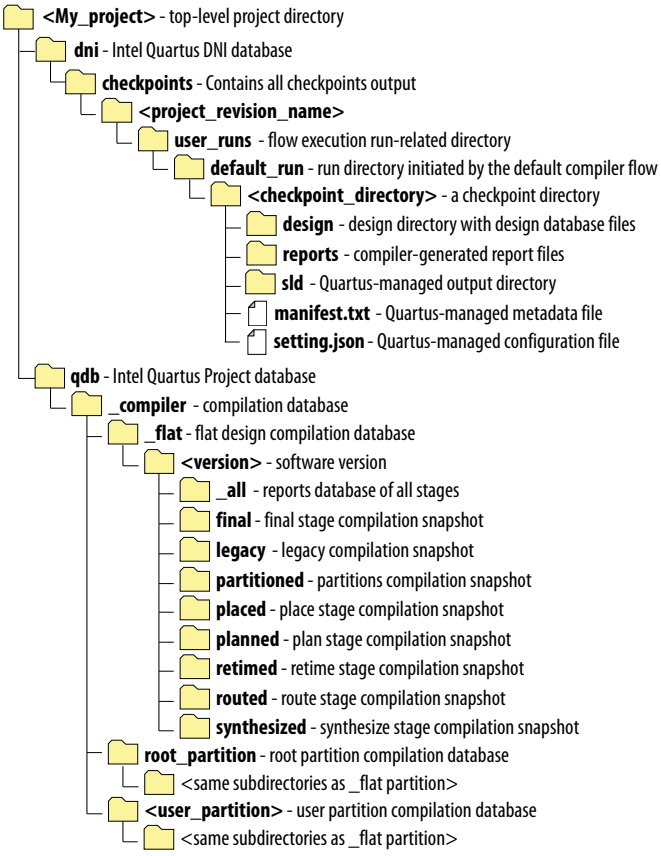
The Quartus Prime Pro Edition Compiler generates a hierarchical project structure that isolates results of each compilation stage, for each design entity.

For example, the `dni` database directory contains checkpoint directories pertaining to each of the Analysis and Elaboration stages, such as elaborated, instrumented, swept, and constrained. Similarly, the `synthesized` directory contains a snapshot of the Synthesis stage of the compilation.

If you use design partitions, such as in a block-based design, the Compiler also isolates the results for each design partition. The Compiler fully preserves routing and placement within a partition. Changes to other portions of the design hierarchy do not impact the partition.

This hierarchical structure allows you to optimize specific design elements without impacting placement and routing in other partitions. The hierarchical project structure also supports distributed work groups and compilation processing across multiple machines.

Figure 2. Hierarchical Project Structure



Related Information

- Exporting Compilation Results on page 136
- Creating a Design Partition on page 140

1.1.3. Compilation on a Compute Farm

If you want to run Quartus Prime Pro Edition compiles on a compute farm, use the Design Space Explorer II or manually submit jobs with your regular job submission commands.

For Design Space Explorer II, refer to the following topics:

- Design Space Explorer II Tool
  - Setup Page
  - Project Page
  - Exploration Page
  - Status Page
  - Report Page
- Using Design Space Explorer
- Optimize Settings with Design Space Explorer II
- Optimize Settings with Project Revisions
- Design Space Explorer II Computing Resources
- Design Space Explorer II Optimization Parameters
- Design Space Explorer II Result Management
- Running Design Space Explorer II
- How to Set Up a Remote Farm Machine in the Design Space Explorer II

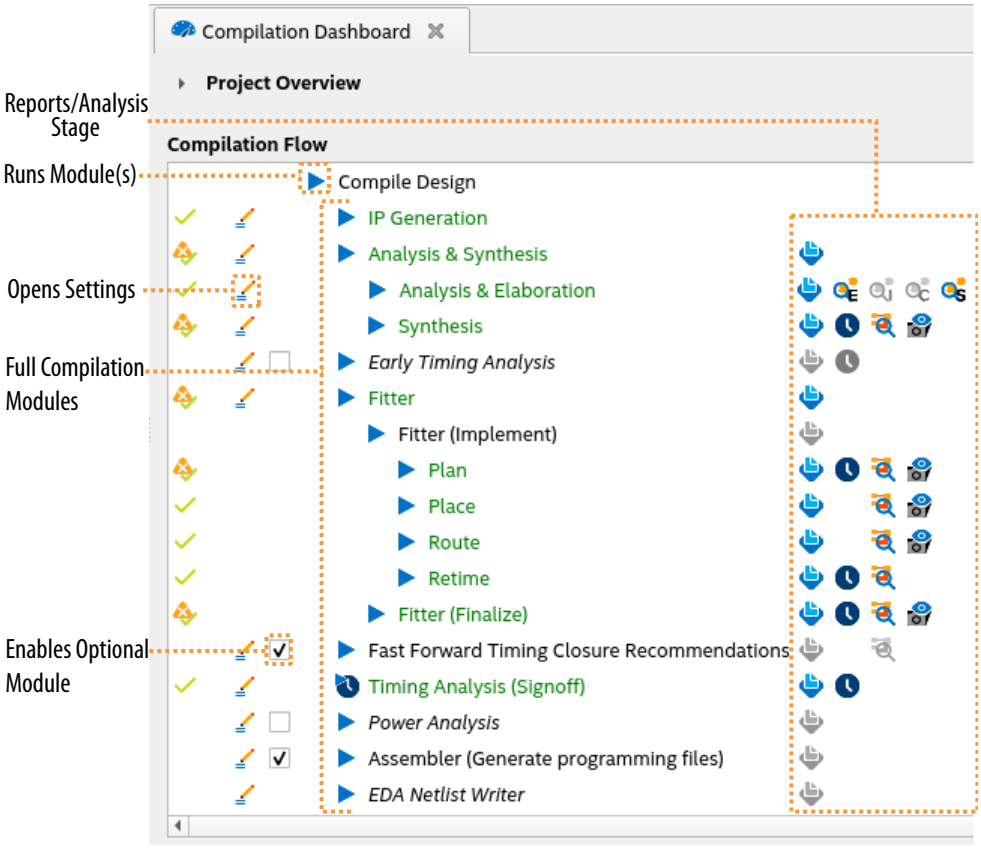
Refer to the command-line commands in the *Quartus Prime Pro Edition User Guide: Scripting* to run a Quartus compilation.

1.2. Using the Compilation Dashboard

The Compilation Dashboard provides immediate access to settings, controls, and reporting for each stage of the compilation flow.

The Compilation Dashboard appears by default when you open a project, or you can click **Compilation Dashboard** in the Tasks window to re-open it.

Figure 3. Compilation Dashboard



- Click the **Pencil** icon to edit settings for that stage of the compilation flow.
- Click any Compiler stage to run one or more Compiler stage.

  You can click a Compiler stage to resume an interrupted compilation flow provided no compilation settings have changed from the initial start of the compilation flow.
- Click the **Report**, **RTL Viewer**, **Technology Map Viewer**, **Timing Analyzer**, or **Snapshot Viewer** icons for analysis of stage results.

As the Compiler progresses through the flow, the dashboard updates the status of each module, and enables icons that you can click for reports and analysis. The dashboard is also updated if you run your compilation flow from a command line with the `quartus_sh --flow` command.

Related Information

- Analyzing Compiler Snapshots on page 107
- Compilation with quartus_sh --flow

1.3. Design Netlist Infrastructure

Design Netlist Infrastructure (DNI) is a major foundational evolution of the Quartus Prime software. It enables new features for faster design convergence and a better user experience.

As a first step, applications and flow for Early Design Analysis have been enabled that unlock the following significant benefits:

- Comprehensive and interactive schematic visualization of an unaltered view of your design (RTL).
- Deeper and advanced design analysis with an intuitive and rich Tcl scripting interface.
- Faster design interactions with granular synthesis.
- Simplified and user-friendly constraint authoring by allowing SDC-on-RTL targets.
- Faster design iterations by SDC cleanup and early timing analysis with post-synthesis timing.

Starting from the 23.3 release, DNI compilation flow is available by default and it is compatible with all components of the design flow. It supports the Assembler to generate and download bit stream to the hardware. It is compatible with Signal Tap, various design flows (Partial Reconfiguration, black box incremental compile, and import/export flows), and simulation model generation.

Related Information

- Use Case Examples on page 50
- TCL Commands and Packages Summary
- ::quartus::dcmd_dni
- ::quartus::dni_sdc

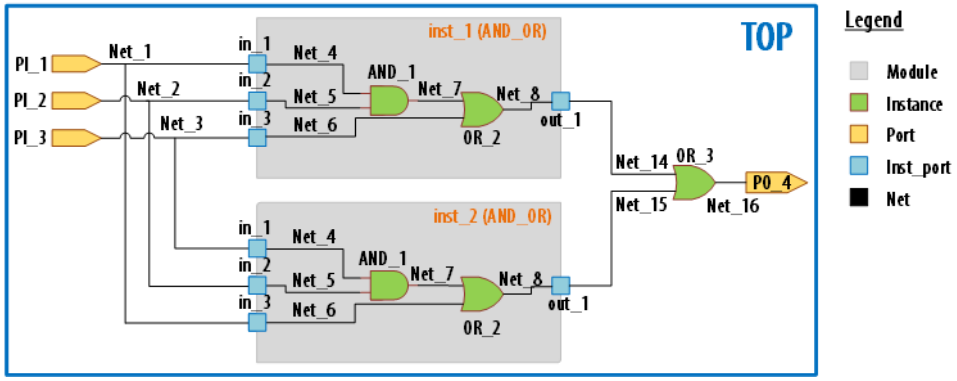
1.3.1. DNI Netlist Five-Box Data Model

DNI introduces a conventional netlist five-box data model used in most Electronic Design Automation (EDA) tools and uses Tcl commands to traverse the netlist. Consider the following two instances example:

```
module top (input PI_1,
            input PI_2,
            input PI_3,
            output PO_4);

    wire net_2;
    wire net_3;

    AND_OR inst_1(PI_1, PI_2, PI_3, net_2);
    AND_OR inst_2(PI_3, PI_2, PI_1, net_3);
    assign PO_4 = net_2 | net_3;
endmodule

module AND_OR (input in_1,
               input in_2,
               input in_3,
               output out_1);

    wire net_1;

    assign net_1 = in_1 & in_2;
    assign out_1 = net_1 | in_3;

endmodule
```

A DNI netlist consists of modules, instances, ports, instance ports, and nets, as shown in the following color-coded diagram:

Figure 4. DNI Netlist Five-box Data Model



The following table describes the core elements of this netlist data model:

Table 3. Core Elements of the Netlist Data Model

Data Model Elements	Description	Tcl Command
Module	A collection of connected netlist objects, such as instances, ports, nets, and instance ports. It is similar to the Verilog module or VHDL entity. Each design has only a single top module containing a group of instances of other modules or library cells. *Note:* A library cell instance is also referred to as a leaf instance.	–
Port	An I/O interface of a module. A design can have input ports, output ports, or bidirectional ports. In the DNI Data Model, `PI_1`, `PI_2`, `PI_3`, and `PO_4` are ports.	`dni::get_ports`
Instance	An instantiation of a module or primitive. A module instance is also referred to as a hierarchical instance, submodule, or a child of the top-level module. A module instance (submodule) may also contain a group of instances of other modules or library cells. These nested module-children instances are referred to as design hierarchies. You can build a hierarchy tree for the entire design with the root as the top module. In the DNI Data Model, `inst_1`, `inst_2`, `AND_1`, `OR_2`, and `OR_3` are instances. *Note:* Multiple unique objects can reference an instance if it exists in a netlist instantiated more than once.	`dni::get_cells`
Instance port (inst_port)	A terminal of an instance. The hierarchical I/O interfaces or leaf instances are referred to as instance ports. Their directions can be input or output. *Note:* FPGA hardware does not support bidirectional signals. Hence, your design must not include any bidirectional instance ports.	`dni::get_pins`

continued...

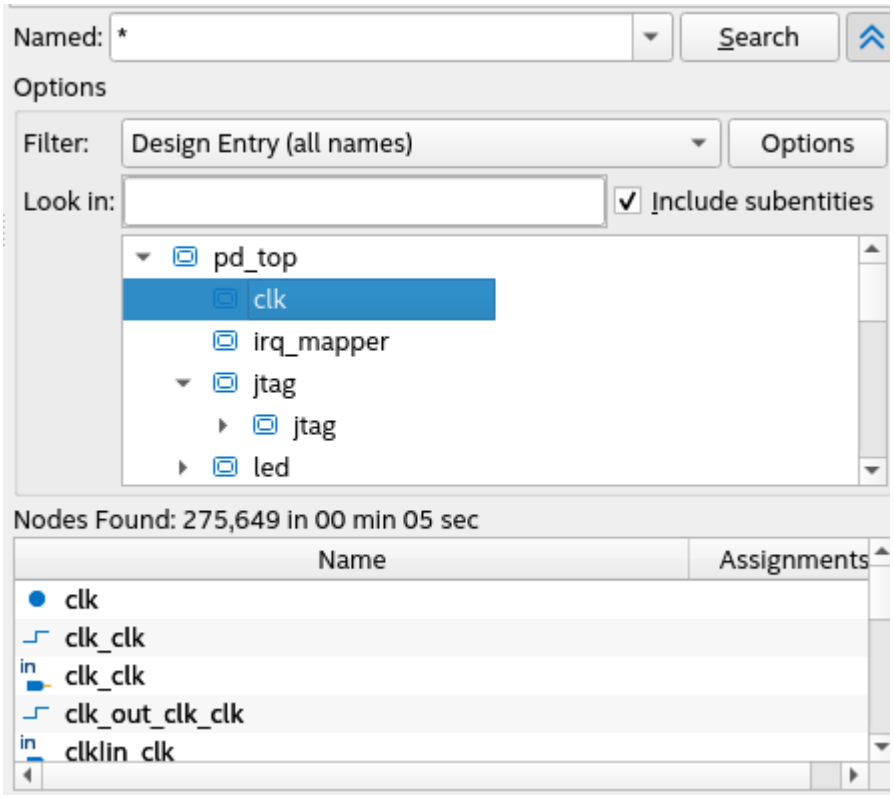
Data Model Elements	Description	Tcl Command
	In the DNI Data Model, `inst_1|in_1`, `inst_2|AND_1|in_1` are few examples of instance ports.	
Net	A wire that connects terminals of instantiations or a netlist. A local net or a hierarchical net is an object within a submodule or top module to hold the connection between instance ports of child instances, instance ports of the submodule boundary, or primary ports of the top module. Hierarchical nets in the submodule and outside the top module of a hierarchical instance port have the same corresponding global or flat net. A global net or a flat net represents the connection of leaf instances or primary ports. In the DNI Data Model, `Net_1`, `Net_2`, `Net_3`, `Net_4` and so on are nets.	`dni::get_nets`

1.4. Using the Node Finder

The Node Finder allows you to search objects in the design netlist based on your search criteria. It returns a list of matching nodes. From the nodes list, you can also locate a node using the RTL Analyzer and other Quartus Prime software tools. In terms of the functionality, it is similar to the Object Finder in the RTL Analyzer. You must complete the Analysis & Elaboration compilation stage to perform a search.

Note: This version of the Node Finder is available only in Quartus Prime Pro Edition software versions 23.3 and later.

To launch the Node Finder, on the Quartus Prime software menu, click **View ➤ Node Finder**. In the following example, the Node Finder has found all user-entered names in your design files:



The **Named** field accepts partial or full-text characters and standard wildcard characters. When you click **Search**, the Node Finder searches all node names matching the specified text. Your search strings are saved, so you can access previously searched strings using the drop-down. The Node Finder provides additional search options (described in the section below) to refine your search.

Node Finder Search Options

Use the **Show More Search Options** button in the Node Finder to apply filters and refine your search. The following search options are available:

Important: Search options available in the Node Finder vary based on the filter you select in the **Filter** drop-down list.

Table 4. Node Finder Search Options

Node Finder Search Options	Description
Filter	Provides a set of default filters. For more information about each filter, refer to the Node Finder Search Filters table. Depending on the filter selected, when you click the **Options** button, some options are enabled, disabled, or grayed out by default.

continued...

Node Finder Search Options	Description
	To refine your search, you can either use one of the default filters or create a custom filter based on one of the default filters using the **Customize** button.
Options	• **Case-insensitivity**: Allows searching case-insensitive node names. By default, the search is case-sensitive. With this option enabled, the search returns the same result irrespective of whether you use lowercase or uppercase search strings. • **Object Type**: Specifies the object type to search for. You can choose between **instance**, **instance_bus**, **instance_port**, **port**, **port_bus**, **net**, and **net_bus**. • **Properties**: Allows adding or removing object properties, such as object name, name of an object's parent, number of ports, source file or line number in the source, and so on. You can type the property values (integer or string) manually.
Look in	Allows you to refine your search hierarchy path. Use the browse button to select the search hierarchy level. It displays the **Select Hierarchy Level** dialog box, allowing you to navigate and select the desired hierarchy level.
Include subentities	Includes node names below the current search hierarchy level in the **Nodes Found** list.
Hierarchy view	Allows you to view nodes in the **Nodes Found** list in hierarchy levels.

Node Finder Search Filters

The following search filters are available in the Node Finder:

Table 5. Node Finder Filters and Search Mode

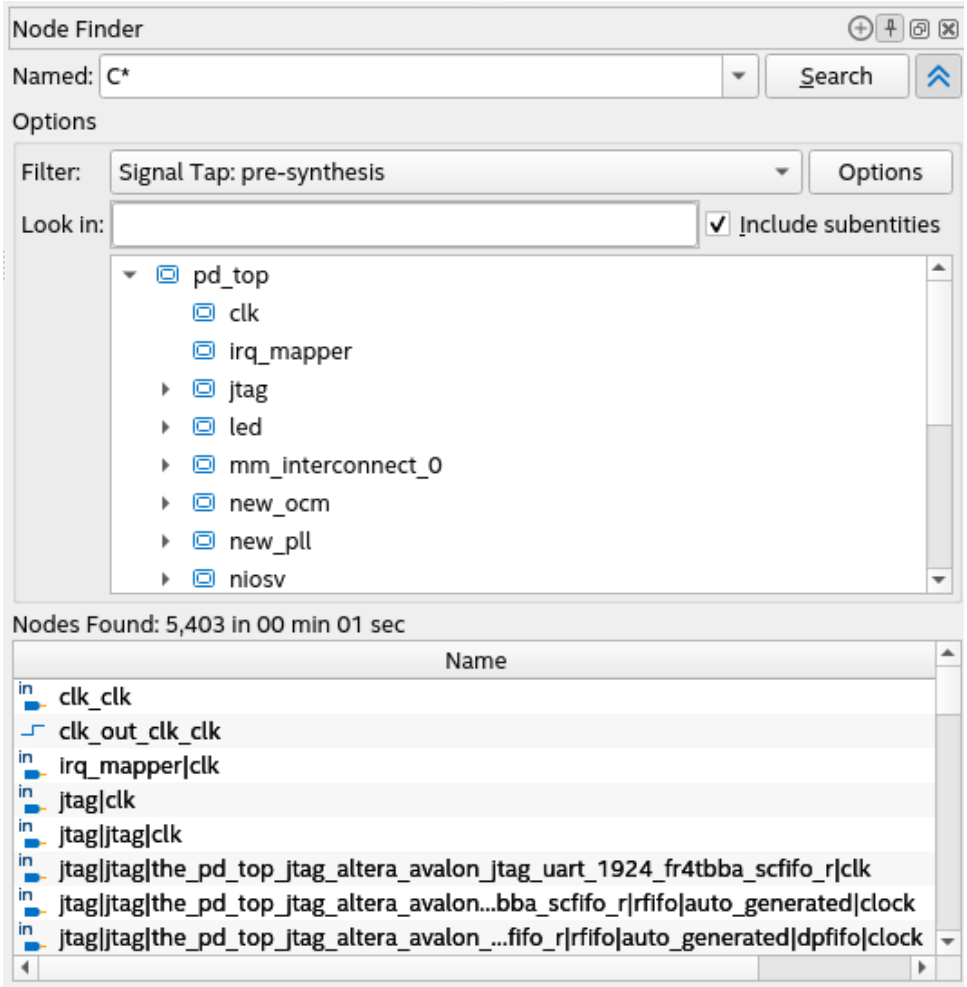
Filter Name	Description
Design Entry (all names)	Finds all user-entered names in the search hierarchy path.
Pins: assigned	Finds all assigned pins in the search hierarchy path.
Pins: unassigned	Finds all unassigned pins in the search hierarchy path.
Pins: input	Finds all input pins in the search hierarchy path.
Pins: output	Finds all output pins in the search hierarchy path.
Pins: bidirectional	Finds all bidirectional pins in the search hierarchy path.
Pins: virtual	Finds all I/O elements mapped to logic elements with a virtual pin logic option assignment.
Pins: all	Finds all pins in the search hierarchy path.
Pins: all & Registers: post-fitting	Finds all pins and registers that persist after physical synthesis and fitting within the search hierarchy path. *Note:* This filter is a combination of the **Pins: all** and **Registers: post-fitting** filters.
Ports: partition	Finds all user-entered and compiler-generated partition ports within the post-fit netlist and search hierarchy path.
Entity instance: pre-synthesis	Finds all entity instances within the pre-synthesis netlist and search hierarchy path.
Registers: pre-synthesis	Finds all user-entered register names contained in the design after Analysis & Elaboration, but before physical synthesis performs any synthesis optimizations.
Registers: post-fitting	Finds all user-entered registers in the search hierarchy path that survived physical synthesis and fitting.
Post-synthesis	Finds all user-entered and synthesis-generated nodes contained in the design after design elaboration and physical synthesis.
Post-synthesis: preserved for debug	Finds all internal device nodes that you have designated with preserve for debug in the post-synthesis netlist.

continued...

Filter Name	Description
Post-Compilation	Finds all user-centered and compiler-generated names that persist post-fit and do not have location assignments.
Signal Tap: pre-synthesis	Finds all internal device nodes in the pre-synthesis netlist that are preserved for analysis by the Signal Tap Logic Analyzer. *Note:* **Signal Tap: pre-synthesis user defined** filter option is unsupported in the DNI mode because functionally, it is same as the **Signal Tap: pre-synthesis**.
Signal Tap: post-fitting	Finds all internal device nodes in the post-fit netlist that are preserved for analysis by the Signal Tap Logic Analyzer.
Signal Tap: post-fitting user defined	Finds all user defined internal device nodes in the post-fit netlist that are preserved for analysis by the Signal Tap Logic Analyzer.
Signal Tap: pre-synthesis preserved for debug	Finds all internal device nodes in the pre-synthesis netlist that are preserved for analysis by the Signal Tap Logic Analyzer.
Signal Tap: post-fitting preserved for debug	Finds all internal device nodes in the post-fit netlist that are preserved for analysis by the Signal Tap Logic Analyzer.

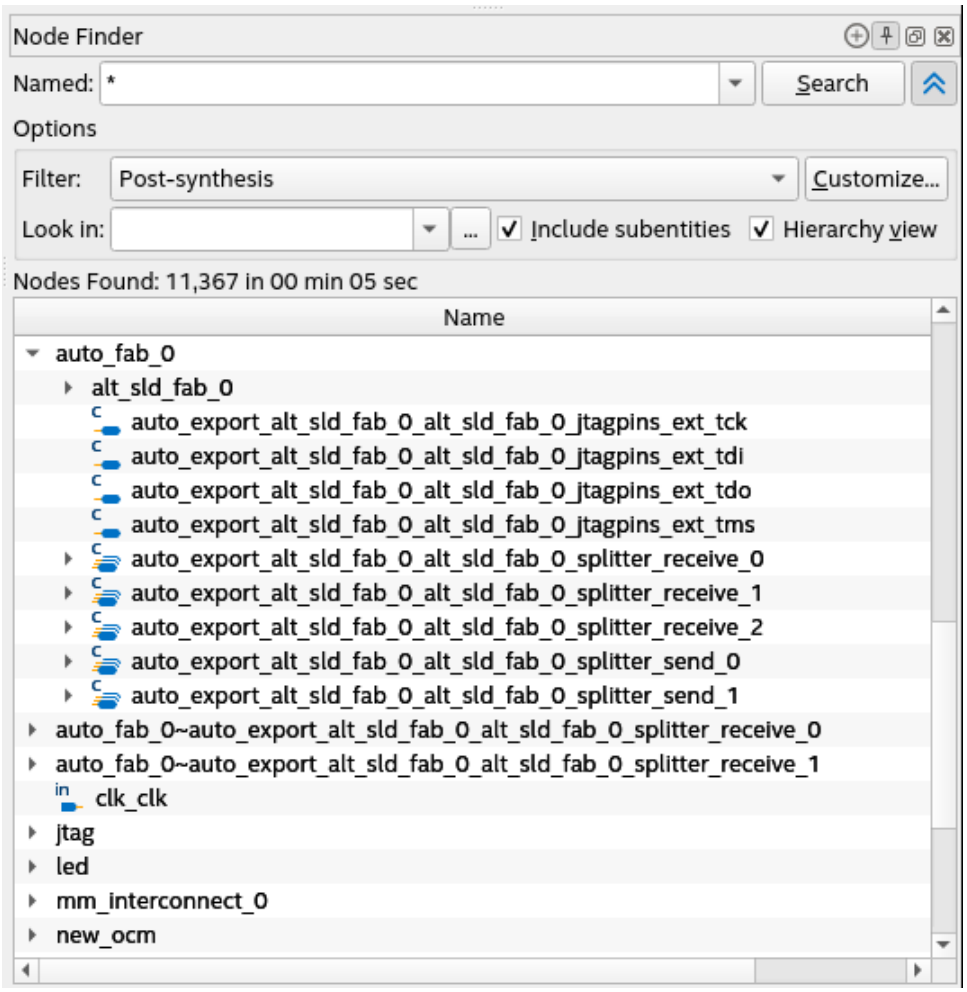
The following image illustrates an example of the Node Finder finding case-insensitive internal device nodes in the pre-synthesis netlist:

Figure 5. Example of Finding Case-insensitive Internal Device Nodes



The following image illustrates an example of the Node Finder using the Post-synthesis filter to find user-entered synthesis-generated nodes in the design:

Figure 6. Example of Finding User-entered Synthesis-generated Nodes



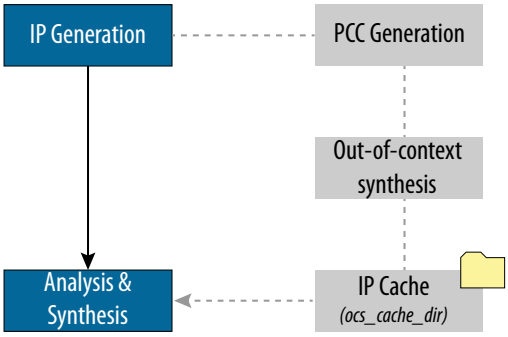
1.5. Precompiled Component (PCC) Generation Flow

Precompiled Components (PCC) generation flow is a compilation stage that runs between IP Generation and Analysis & Synthesis stages on the compilation dashboard. It synthesizes and caches the results of the Intel® FPGA IP components in your design.

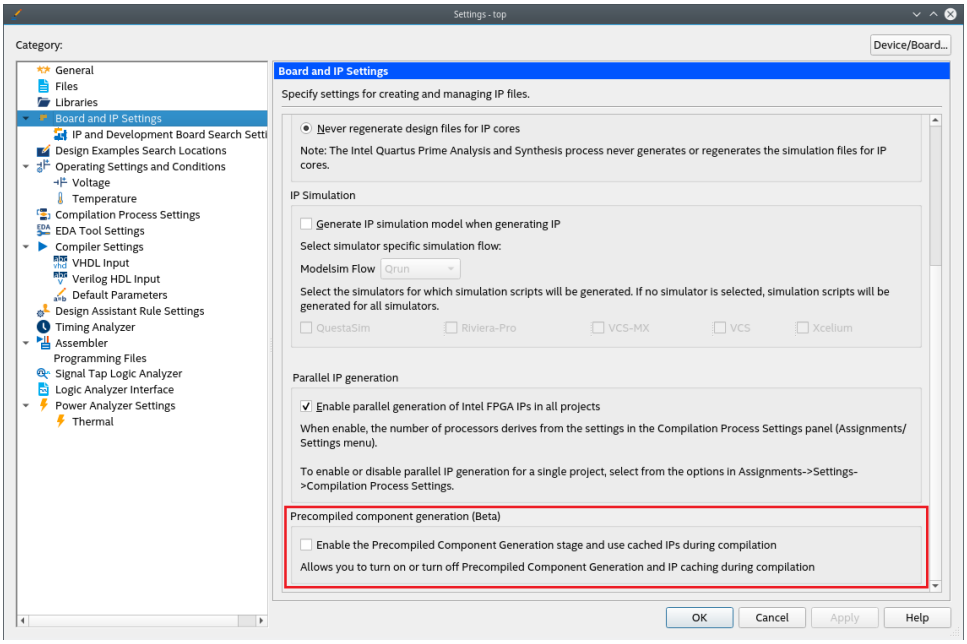
PCC generation reduces Synthesis compile time through the following steps and allows you to iterate and go to post-synthesis timing analysis stages faster and reduce the development cycle time:

1. Synthesizes Intel FPGA IP components in your design.
2. Generates and caches IP compilation results in your project's `ocs_cache_dir` (IP cache) directory.
3. Reuses the IP cache in subsequent Synthesis runs.

Figure 7. PCC Generation



By default, the PCC generation flow is disabled. You can enable it from **Assignments ➤ Settings ➤ Board and IP Settings** dialog as shown in the following:



If the PCC generation flow compiles successfully, you can view the **Precompiled Components** compilation report for each IP that participates in the flow under the **Compilation Reports** dialog.

Advantages of PCC Generation Flow

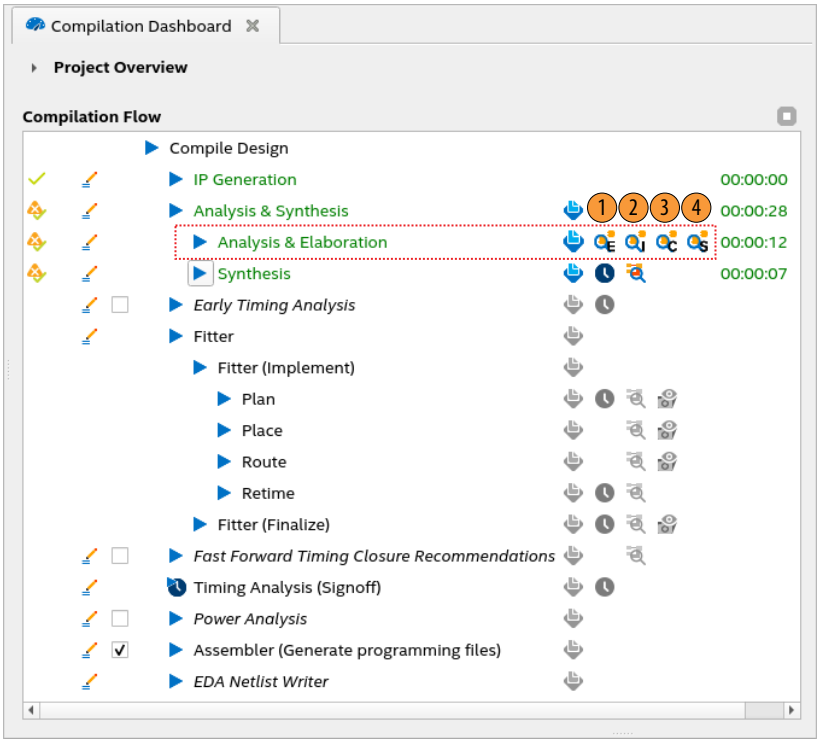
- Each unique IP is generated only once and cached, so the subsequent Synthesis run does not perform PCC generation if you have not modified any IP in your design.
- If you modify any IP, the PCC generation flow synthesizes only that IP and skips synthesizing the remaining IPs.
- The PCC generation process occurs in parallel for each IP.
- Synthesis compile time savings scales with the proportion of the design that is IP.

1.6. Analysis & Elaboration Flow

The Analysis & Elaboration compilation flow provides a complete and unmodified view of your design early in the compilation flow. It serves as a platform to better analyze your design and improve it. With this feature, the monolithic Analysis & Synthesis stage splits into Analysis & Elaboration and Synthesis stages.

This flow allows you to access the preview modes of the Analysis & Elaboration stage, as shown in the following image:

Figure 8. Analysis & Elaboration Checkpoints

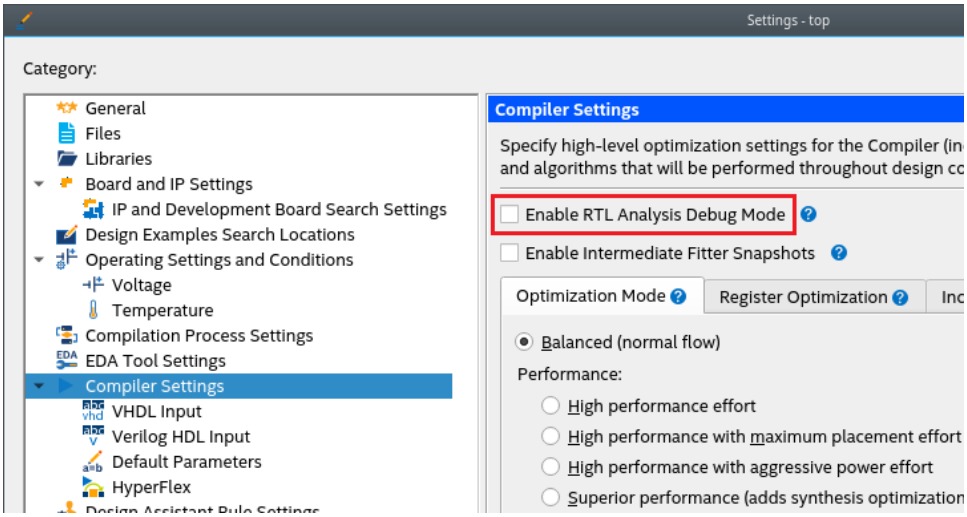


The Analysis & Elaboration stage is composed of a series of checkpoints and you can preview your design at each checkpoint as shown in Analysis & Elaboration Checkpoints, where:

1. **Elaborated**: Provides an unmodified preview of your design captured directly from RTL.
2. **Instrumented**: Provides an instrumented preview with system-level debugging (debug fabric and Signal Tap logic analyzer inserted in your design). This checkpoint is disabled by default (**Hint**: See Note below to enable it).
3. **Constrained**: Provides a design preview with SDC-on-RTL constraints shown on the target nodes. This checkpoint is disabled by default (**Hint**: See Note below to enable it).
4. **Swept**: Provides a design preview with unnecessary logic removed from your design.

Note: You can control the number of checkpoints generated using the **RTL Analysis Debug Mode** option under **Project ➤ Settings**. This mode is off by default, which means only Elaborated and Swept checkpoints are available, and Instrumented and Constrained checkpoints are unavailable. When you enable this mode, all four checkpoints become available.

When the mode is off, you can obtain information about the **Hierarchies Optimized Away** and **Top Causes for Logic Optimized Away During Sweep**, under **Synthesis Compilation Reports ➤ Synthesis ➤ Analysis & Elaboration**.



For information about the Synthesis stage, refer to Design Synthesis on page 58.

Caution: **Incompatibility with older software versions**

A design compiled with the DNI-based compilation flow is not compatible with the Quartus Prime software versions older than 23.3 due to the new DNI database. Ensure the following:

- If you compile your design with the software version 23.3 and launch the project in the Quartus Prime software GUI, a message displays indicating that you compiled your design with a different compilation engine. If you continue to launch the project, then you must recompile the design. This also applies for designs compiled with the Quartus Prime software versions older than 23.3.
- For any partition-based design (for example, Partial Reconfiguration), mixing 23.3 version with 23.2 or older version compilation flow among the user-defined partitions is not supported. You must compile all partitions with the same compilation flow.
- For version-compatible design export or import feature, a design compiled with and exported from an older Quartus Prime version requires the Quartus Prime 23.2 or older version. Similarly, the Quartus Prime 23.3 version is necessary for version-compatible databases compiled with the DNI flow, and this feature is not supported yet.

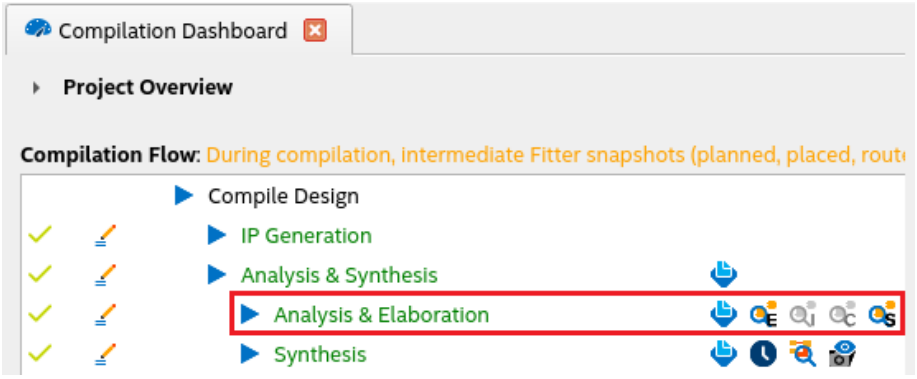
Executing Tcl Commands

The Quartus Prime software GUI (`quartus`) and Synthesis tool (`quartus_syn`) support Tcl commands.

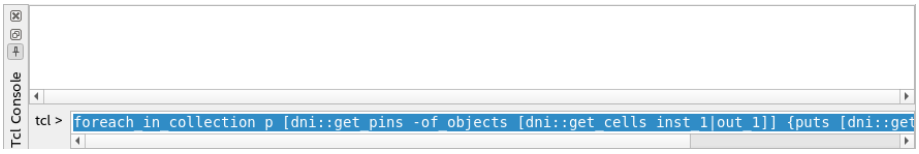
Use one of the following suitable methods to execute your Tcl commands:
Quartus Prime Software GUI (`quartus`)

Perform the following steps in the GUI:

1. On the **Compilation Dashboard**, run **Analysis & Synthesis ➤ Analysis & Elaboration** task to generate the netlist.



2. Click the magnifier icon to Invoke the RTL Analyzer.
3. Execute your Tcl command in the **Tcl Console**.



Synthesis Tool (`quartus_syn`)

1. Enable the flow for your project with the following command:

```
quartus_syn --analysis_and_elaboration <project_name>
```

2. Load your design.

```
> quartus_syn -s
<... Quartus Info Message...>
tcl> project_open top
tcl> dni::load_design -checkpoint elaborated
dms_path::sandboxes::sandbox_1239_0::design
```

3. Execute your Tcl commands:

```
tcl> foreach_in_collection p [dni::get_pins -of_objects [dni::get_cells
inst_1|out_1]] {puts [dni::get_property -name name -object $p]}
a[0]
a[1]
o
tcl> foreach_in_collection p [dni::get_pins -of_objects [dni::get_cells
inst_1|out_1]] {puts [dni::get_property -name direction -object $p]}
input
input
output
tcl>
```

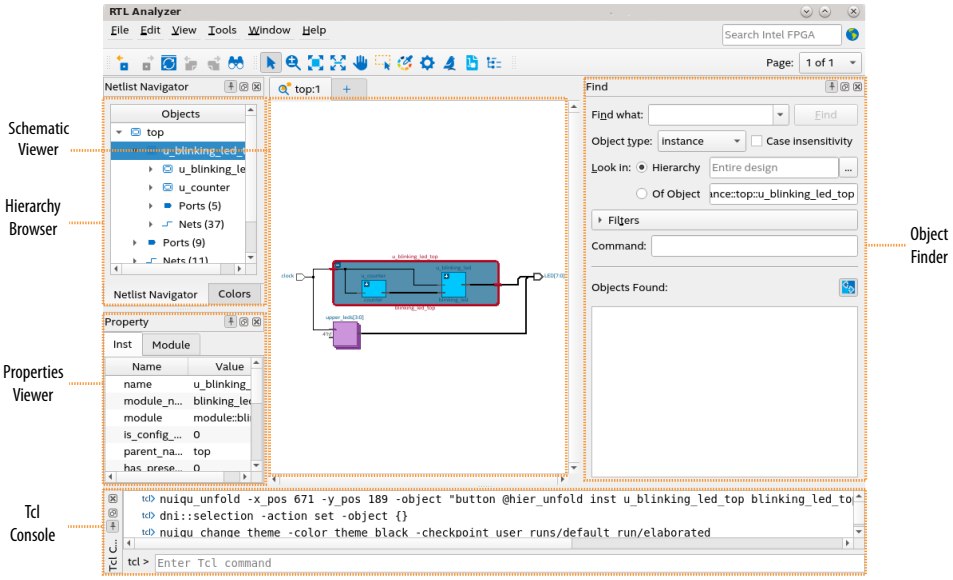
1.6.1. Exploring the RTL Analyzer

You can invoke the RTL Analyzer from the Elaborated checkpoint of the Analysis & Elaboration stage by clicking on the first magnifier icon in the compilation dashboard. For details about checkpoints, refer to the Analysis & Elaboration Flow on page 20.

The RTL Analyzer GUI has the following major components:

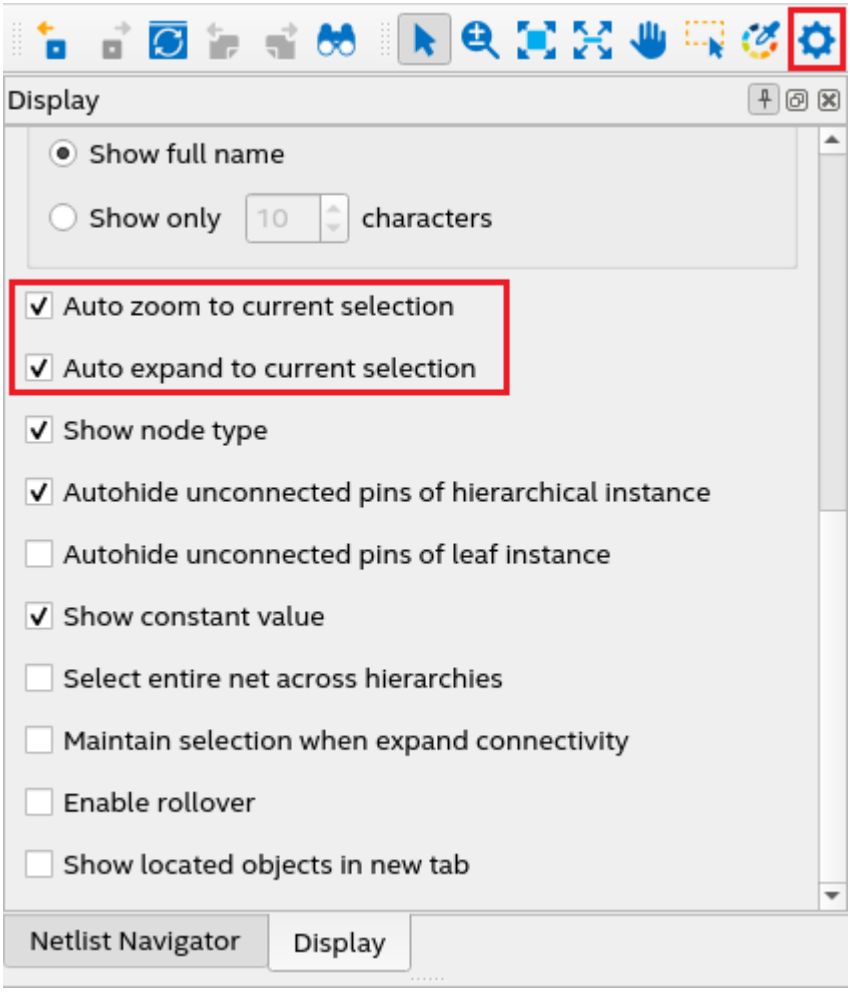
- Design Hierarchy Browser
- Schematic Viewer
- Object Property Viewer
- Tcl Console
- Object Finder

Figure 10. RTL Analyzer GUI



Note: Before exploring your design in the RTL Analyzer, select your desired viewing modes in the **Display** settings by navigating to **View ➤ Display Settings** menu. The **Auto zoom to current selection** and **Auto expand to current selection** modes are enabled by default in the **Display** settings. You can disable one or both of these modes. In the enabled mode, the object you select in the hierarchy browser gets automatically highlighted in the schematic viewer. The schematic viewer and hierarchy browser expands the current selected object's hierarchy and adjusts the view to ensure the object is visible. Disable both modes if you do not want your current view to change in the hierarchy browser and schematic viewer.

Figure 11. Display Settings



Note: RTL Analyzer automatically saves the last view of your design when you close it. So, when you relaunch the RTL Analyzer, the last view is restored, including the highlighted objects and nodes, and you can continue your work from where you left it previously. However, in the event you change your RTL design and recompile the project after exiting the RTL Analyzer, the saved view is invalidated and not restored.

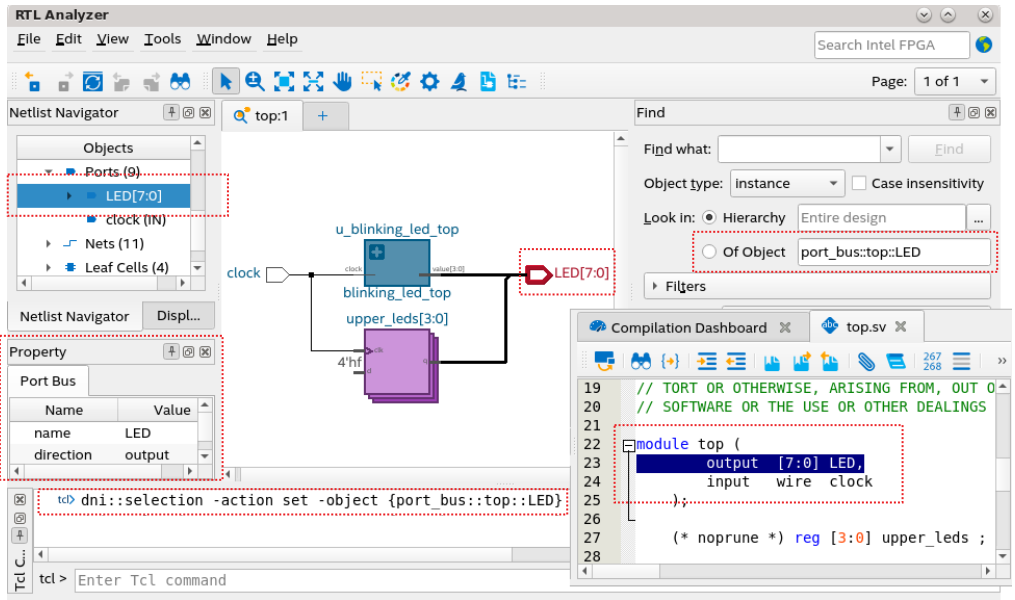
Design Hierarchy Browser

The design hierarchy browser renders the design netlist five-box data model. The core elements of this data model are modules, instances, ports, instance ports, and nets. For more information about this data model, refer to DNI Netlist Five-Box Data Model on page 11.

All objects in the hierarchy are organized by the object type. For example, I/Os are grouped based on the direction.

Various GUI elements of the RTL analyzer are synchronized and respond to the objects within the **Objects** pane. For example, for the currently selected object in the hierarchy browser, the schematic viewer brings the object into focus with a highlight, the **Tcl Console** emits an equivalent Tcl command, and the **Properties** viewer provides relevant information about the object. You can cross-probe from selected objects in the netlist to the source where they were defined by right-clicking on an object and selecting **Locate Node ➤ Locate in Design File** option in the context-sensitive menu. The source file displays in the Quartus main window showing the line that instantiates the object.

Figure 12. Cross-probing Selected Objects in the Netlist



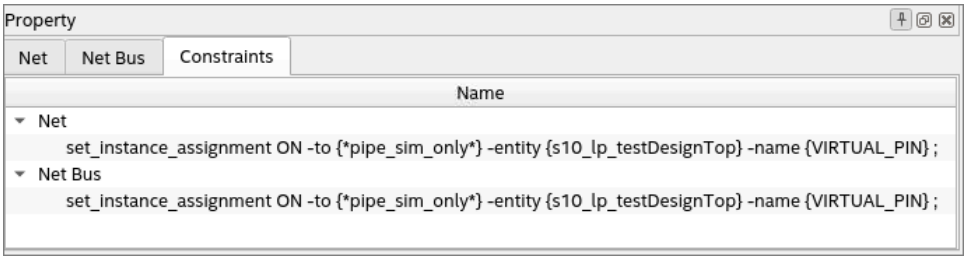
Schematic Viewer

The schematic viewer represents the design elements, such as modules, instances, ports, instance ports, and nets, schematically within the viewer. The viewer reacts and updates to bring different objects into focus as you navigate through the netlist.

Properties Viewer

The **Property** pane provides information about assignments and SDC constraints attached to the selected object in the hierarchy browser. You can cross-probe to the definition of the assignment in the source file.

Figure 13. Property Viewer Showing Constraints



Tcl Console

The Tcl Console provides a robust scripting interface that reports issues and displays relevant Tcl commands for GUI actions. It allows you to traverse and analyze your design. As you type the Tcl commands, notice how the schematic viewer and the hierarchy browser respond accordingly.

Object Finder

The object finder helps in locating an object in the design netlist. You can refine your search based on the object type (instance, instance bus, inst_port, port, port_bus, net, and net_bus) and modules. For complicated designs, you can further use filters to refine your search.

The RTL Analyzer provides the following tools and functionality:

Sweep Hints Viewer on page 26
Object Set Console on page 33
Module Interfaces on page 37
Bundled Instances on page 40
Auto-hide Unconnected Pins on page 45
Filtering Ports and Nets on page 46
Expand Connections on page 49

Related Information

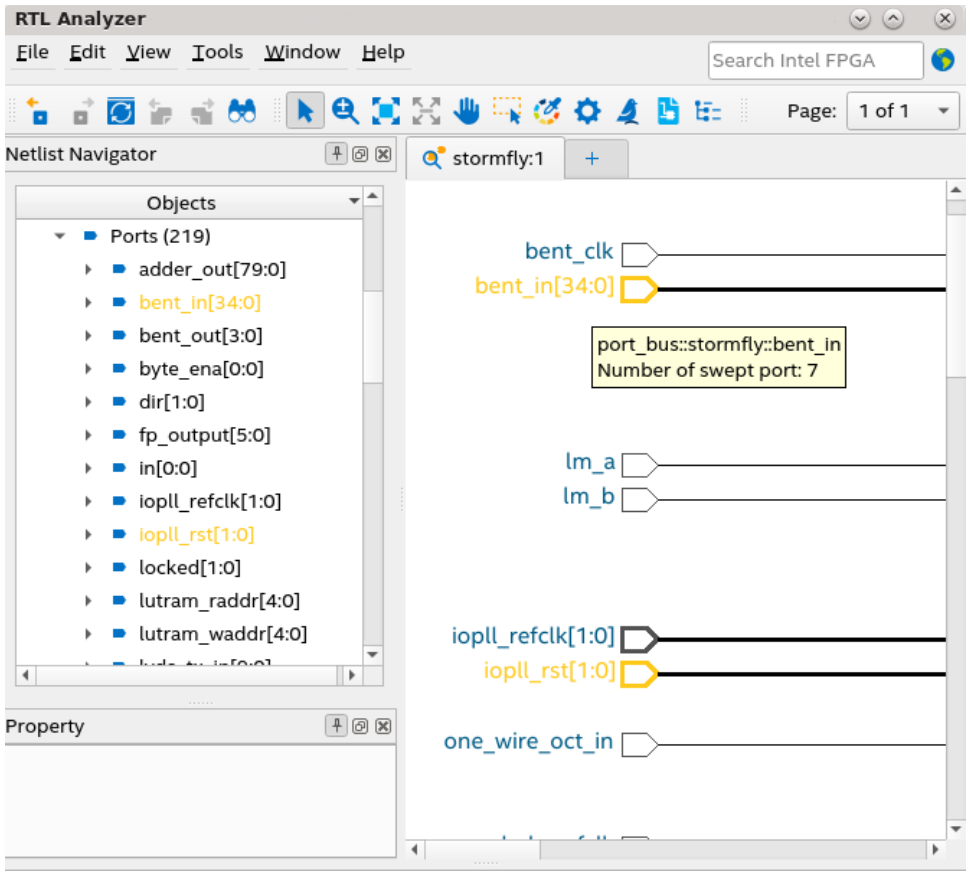
Overview Video: RTL Analyzer for Quartus Prime Software

1.6.1.1. Sweep Hints Viewer

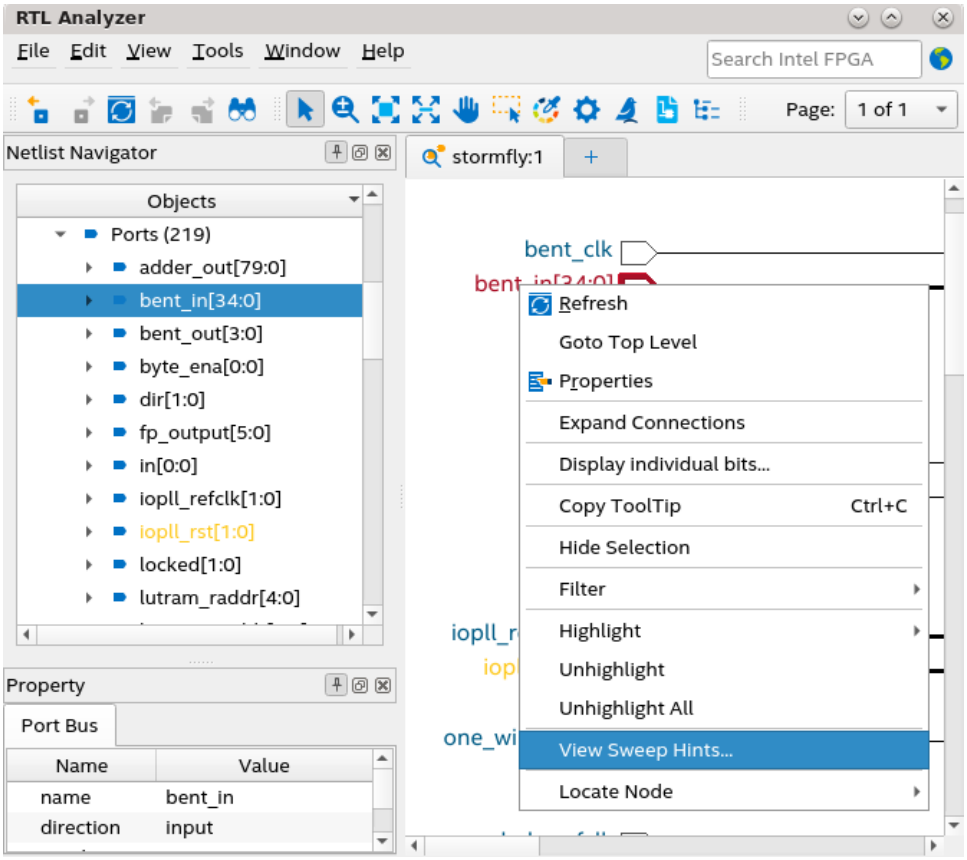
Sweep Hints Viewer allows you to visualize and identify why Synthesis removed logic in your design. It is one of the RTL Analyzer tools you can access from the menu by selecting **Tools ➤ View Sweep Hints**. You can also launch it from the context-sensitive menu of the swept instance.

The RTL Analyzer highlights the top-level module port and leaf instance with associated sweep hints in orange color. When you hover over the swept port or instance, the tooltip displays the number of swept ports, sweep hint type, and sweep hint reason. These highlights and tooltips are available in both the schematic view and the Netlist Navigator, as shown in the following image:

Figure 14. Sweep Hints View in the RTL Analyzer

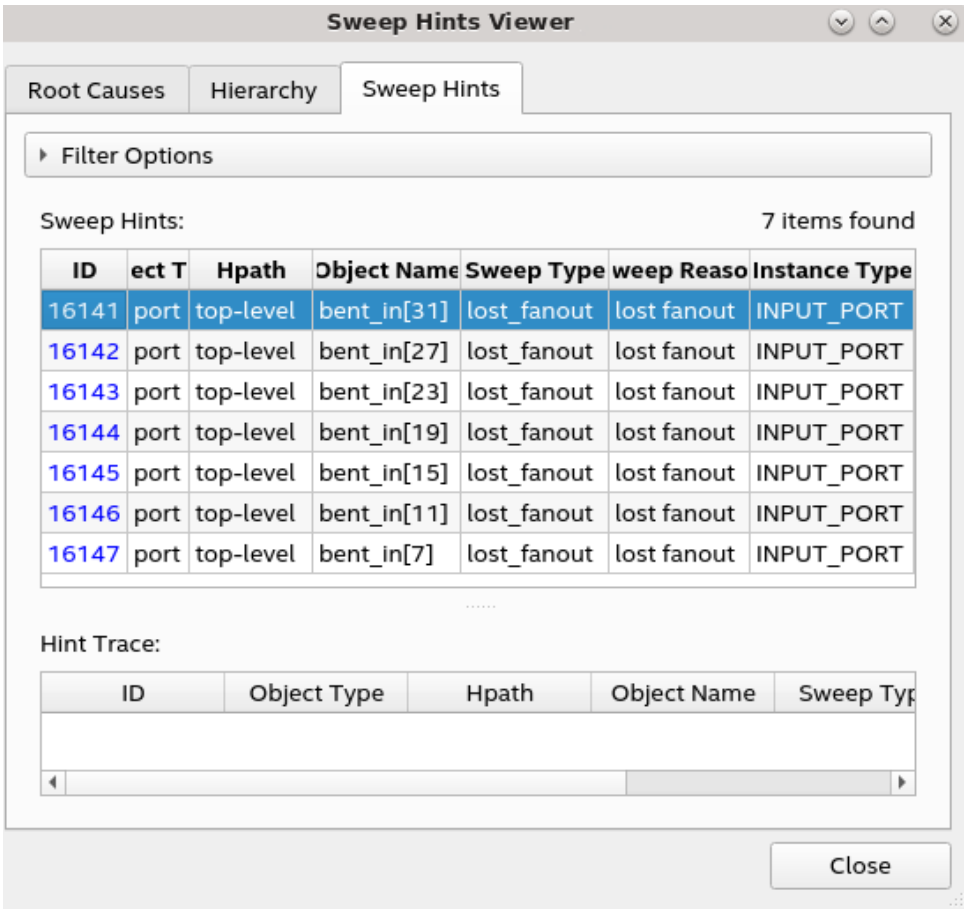


From the RTL Analyzer, you can launch Sweep Hints Viewer by selecting **View Sweep Hints** from the context-sensitive menu of the swept instance or the top-level module port in the netlist navigator or schematic view, as shown in the following image:



Upon selecting the **View Sweep Hints** option from the context-sensitive menu, the Sweep Hints Viewer auto-populates with sweep hints of the selected instance or port, as shown in the following image:

Figure 15. Auto-populated List of Sweep Hints for the Selected instance



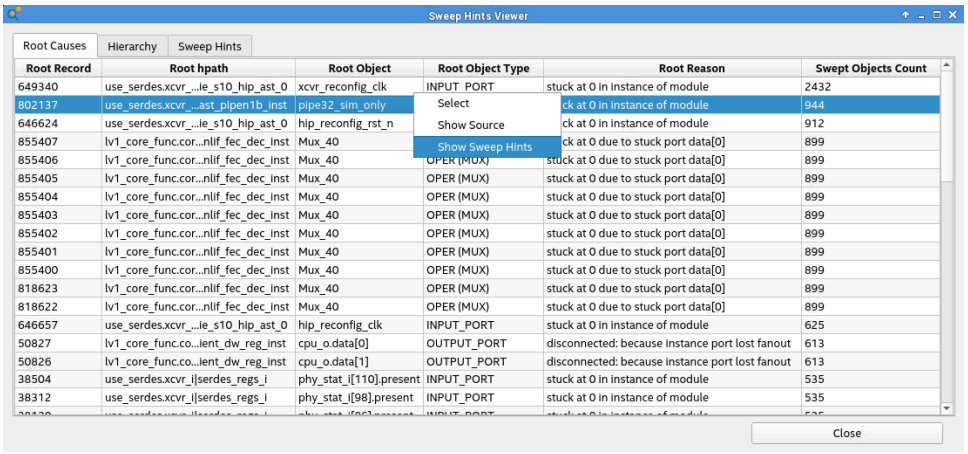
Root Causes View of the Sweep Hints Viewer

To identify the root cause for a particular node in the design being swept away, you must understand various reasons behind why a single node was marked during the sweep, as listed in the following:

- A node stuck at constant 0 or 1. For example, an input port of an AND gate stuck at 0 triggers a sweep of the AND gate since the output of the AND gate is also stuck at 0.
- A node behaved as a wire and was deleted. For example, a multiplexer whose select line is connected to constant acts as a wire.
- A node was modified during the sweep but it was not deleted.
- A node lost all fan-outs and it was deleted.
- A port or instance port got disconnected because the parent or child did not have a fan-out.

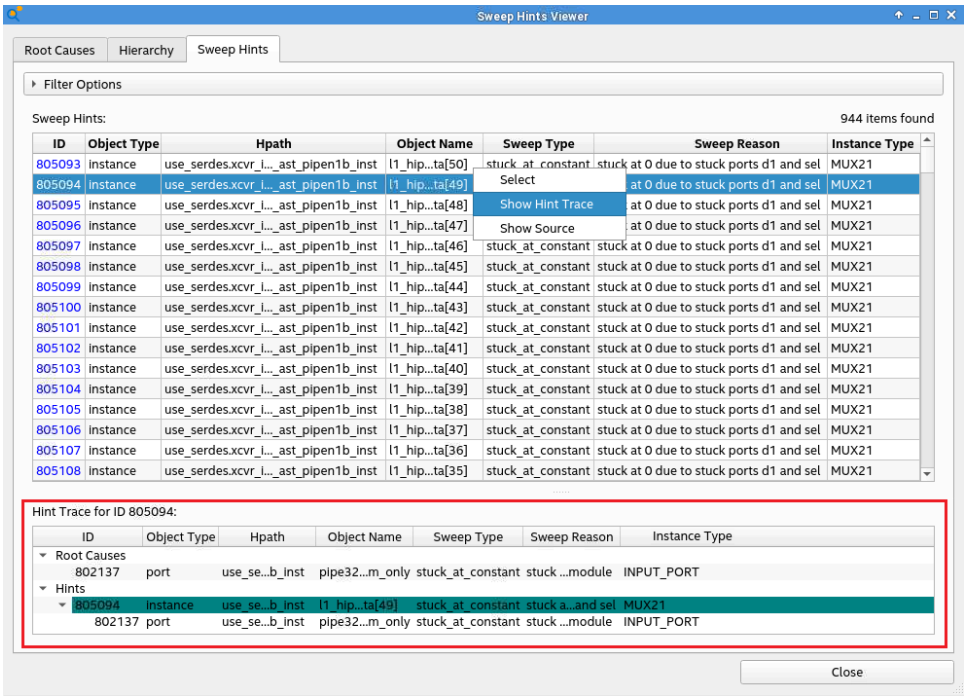
The **Root Causes** tab of the Sweep Hints Viewer provides a list of sweep records and their root cause for being swept away, as shown in the following example:

Figure 16. Root Causes Tab of the Sweep Hints Viewer



The Root Causes view shows the root reason and total number of swept objects of a root object in the **Swept Objects Count** column, which you can trace to a sweep record in the **Root Record** column (for instance, 802137 in the above image). Selecting the **Show Sweep Hints** option in the context-sensitive menu switches the interface to the **Sweep Hints** view to show all sweep records that identify the root record as the root cause, as shown in the following image:

Figure 17. Tracing Sweep Records of a Root Record



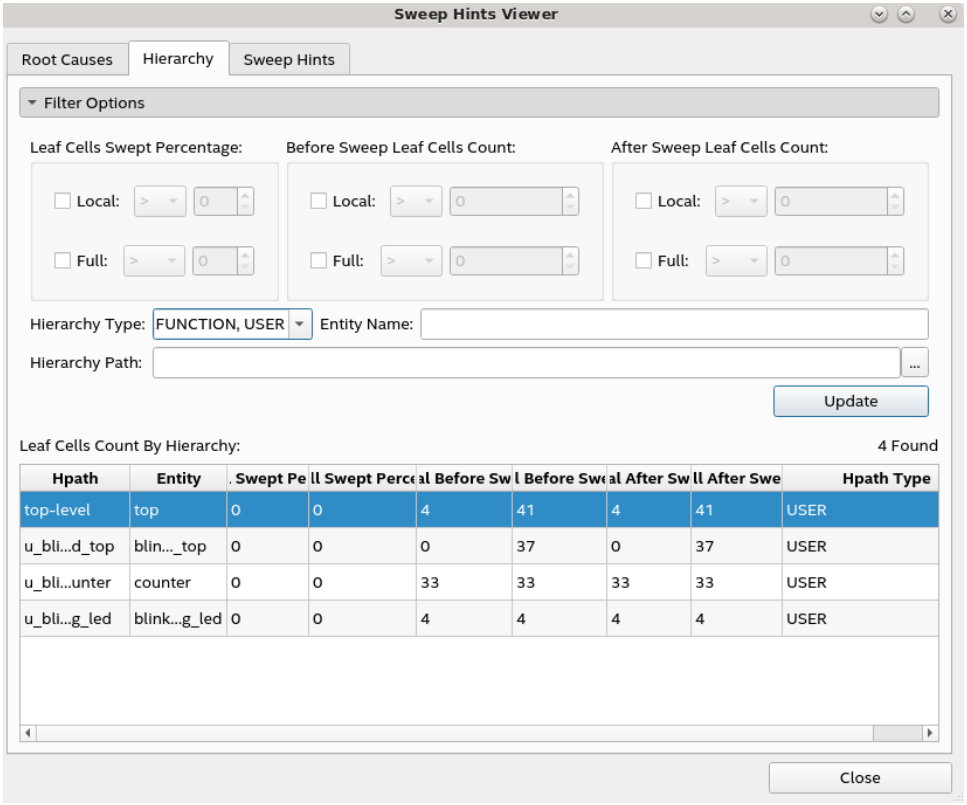
Note: For more information about the **Show Hint Trace**, refer to the Sweep Hints View of the Sweep Hints Viewer on page 32 section.

Hierarchy View of the Sweep Hints Viewer

Recall from the Analysis & Elaboration Flow on page 20 how to sweep optimization (Swept checkpoint) happens as part of the Analysis & Elaboration stage. The sweep optimization phase leaves behind a database of objects that are swept away along with the reason for their removal. The database also tracks relationships between the removed objects. For example, if an output of a gate is dangling, it is swept away. Few cells in the fan-in cone can also be swept away because their output does not drive anything useful. The Sweep Hints Viewer allows you to view the sweep hints database with a list of marked objects in the pre-swept design that are removed later.

Note: For large and complex designs, the total number of swept objects can be significant (in millions). Hence, the swept objects are hierarchically arranged so that you can quickly query, filter, and find regions of your interest.

Figure 18. Hierarchy Tab in the Sweep Hints Viewer



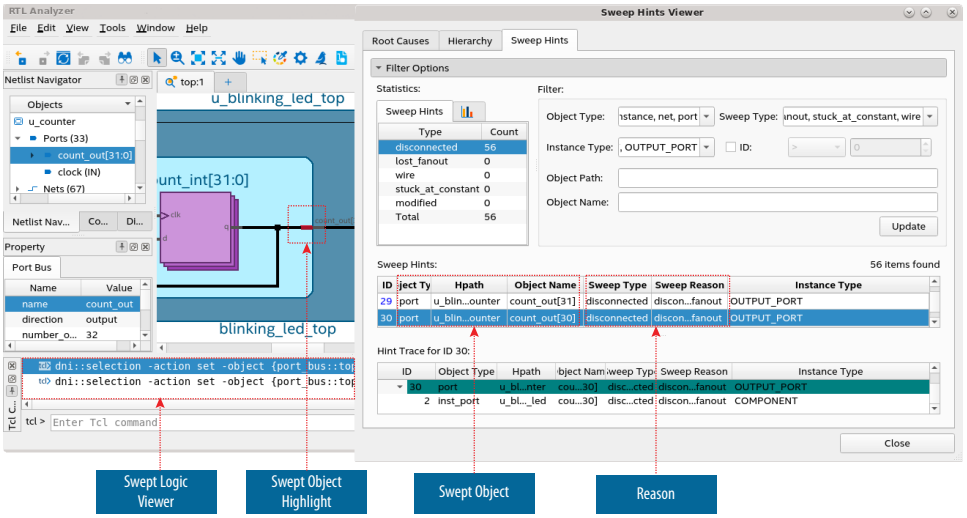
Within the **Hierarchy** tab of the Sweep Hints Viewer, you can observe various hierarchy levels and their sweep statistics. The logic unit here is a leaf cell, which refers to logical primitives, such as AND/OR gate or registers. A hierarchical cell is a composite of leaf cells and other hierarchical cells.

The **Leaf Cells Count by Hierarchy** table in the above image provides information about the percentage of the logic swept either locally or in the full hierarchy of a given hierarchical path. The table columns for the number of leaf cells indicate before and after a sweep and the type of hierarchical path, which can be USER (RTL you designed), Intel IP, or MEGAFUNCTION (Intel-provided library modules). If the hierarchy table is large, use the filter options to refine the regions of your design.

Note: Intel IP and MEGAFUNCTION types are hidden by default since you cannot modify them. However, you can view them in specific circumstances by selecting the desired option in the **Hierarchy Type** drop-down list. Encrypted modules are always hidden from this view.

Sweep Hints View of the Sweep Hints Viewer

Figure 19. Example of Swept Objects in the Sweep Hints Viewer



Notice the swept object highlighted with the teal color in the schematic view. If you hover over this object tooltip or view the sweep reason in the Sweep Hints Viewer, you can see that this instance was swept away because the instance port lost fanout. Similarly, you can select other swept objects to identify their root cause and view them in the schematic viewer and hierarchy view. All these objects are eventually swept away from the design when you proceed with the compilation flow.

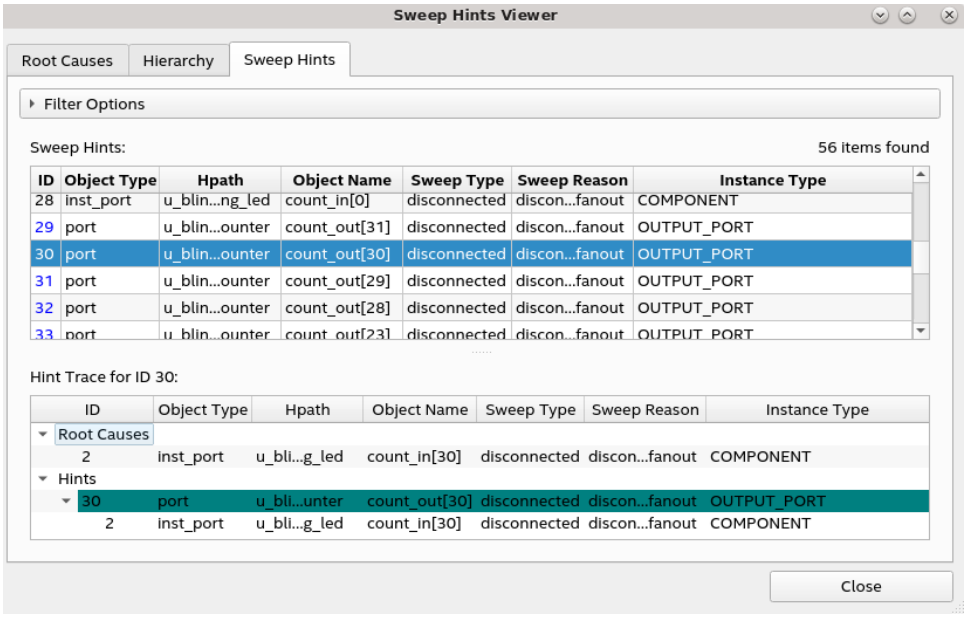
While the **Hierarchy** tab helps filter by the sweep statistics of an instance, the **Sweep Hints** tab allows you to filter and query the leaf objects swept away in a given hierarchy level. Various columns in the table describe the object involved in the sweep and provide ways to query data. For instance, you can look for specific instance types or limit the results to specific sweep reasons, types, and so on.

Viewing Related Swept Objects

You can also view related swept objects by right-clicking on a select object under **Sweep Hints** and selecting **Show Hint Trace** in the context-sensitive menu. **Hint Trace for ID <*number*>** section displays a hierarchical listing of swept objects that caused a particular object to be swept away and objects swept away as a result of one object. The ability to highlight and select various objects in the chain allows you to narrow down the desired sweep information.

In the following figure, when you trace the sweep hints for a port with object ID 30, under **Hint Trace for ID *30***, you notice an instance port with object ID 2 being related to the swept port 30. The sweep reason for both objects is disconnected fanout.

Figure 20. Related Swept Objects



1.6.1.2. Object Set Console

The Object Set Console helps you conveniently navigate and manipulate the currently selected, highlighted, or colored node object sets when you have multiple lists to manage in the console. To launch the Object Set Console dialog box, select **View ➤ Object Set Console Window** on the RTL Analyzer menu.

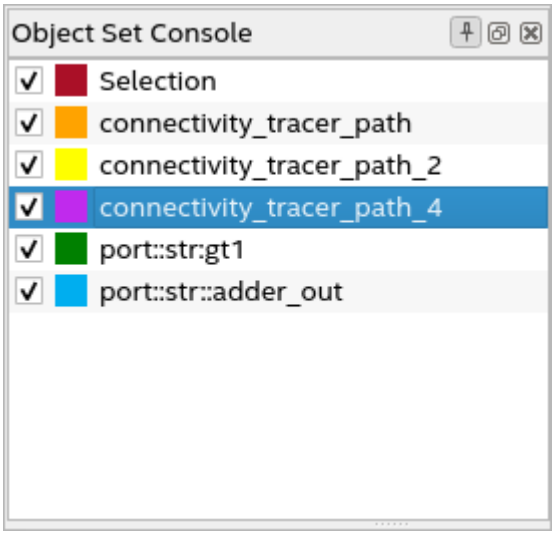
The Object Set Console is split into a top pane and a bottom pane, which are explained in the following sections:

Top Pane of the Object Set Console Window

The top pane displays the design's currently selected, highlighted, or colored nodes.

Consider an example where you traced fan-in or fan-out connectivity of your design. When you view a specific traced path, it gets highlighted in the schematic viewer. When you click **Show Path** for one or more traced paths, the top pane lists all paths in different color codes, as shown in the following image:

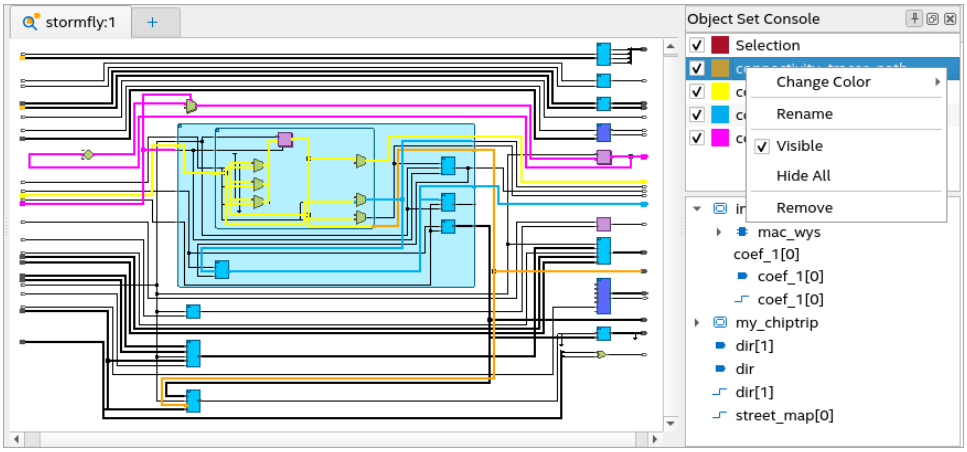
Figure 21. Top Pane of the Object Set Console



Note: You can save all highlighted paths and restore them anytime using the **Save** and **Restore** options, respectively, from the context-sensitive menu.

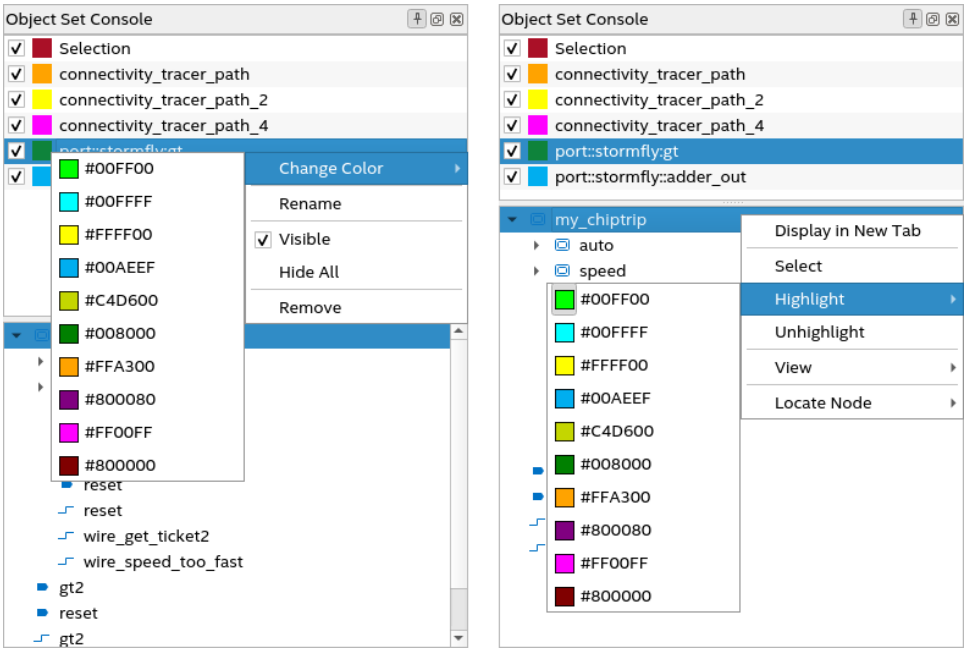
Use the context-sensitive menu to change node color, rename the traced path identifier, toggle a node's visibility, hide one or more traced paths. or delete a node from the list, as shown in the following image:

Figure 22. Object Set Console Window



To change the node color, use the **Change Color** option. Alternatively, you can directly use the **Highlight** option in the bottom pane to select a color and highlight the traced path in the desired color.

Figure 23. Changing Node Color from the Context-sensitive Menu



Bottom Pane of the Object Set Console Window

When you select one of the nodes in the top pane, the bottom pane displays a tree or list view of all netlist objects (ports, instances, instance ports, and nets) that are part of that node. Using the context-sensitive menu, you can select and view the desired node in a new tab within the schematic viewer and highlight/unhighlight the path with a different color.

With the **View** sub-options (List, Hierarchy, and Type), you can change the view of the traced paths. By default, the objects are organized hierarchically in the bottom pane.

- **List** lists out all the netlist objects.
- **Hierarchy** arranges the netlist objects in the traced path hierarchically. This is the default view.
- **Type** categorizes and lists netlist objects by type (for example, Ports and Nets).

Figure 24. Bottom Pane Context-sensitive Menu

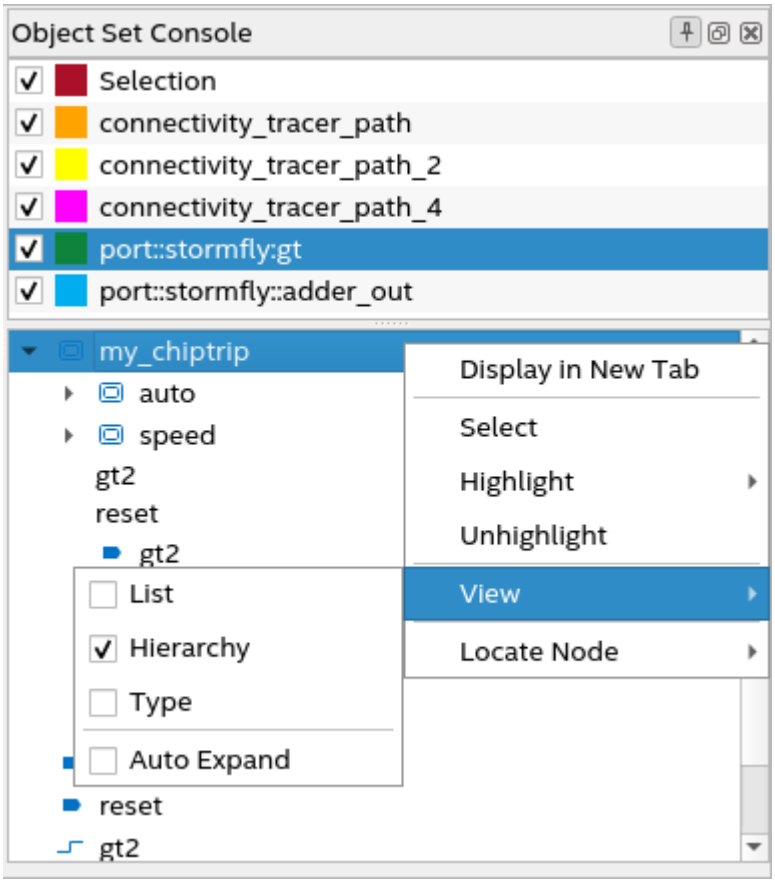
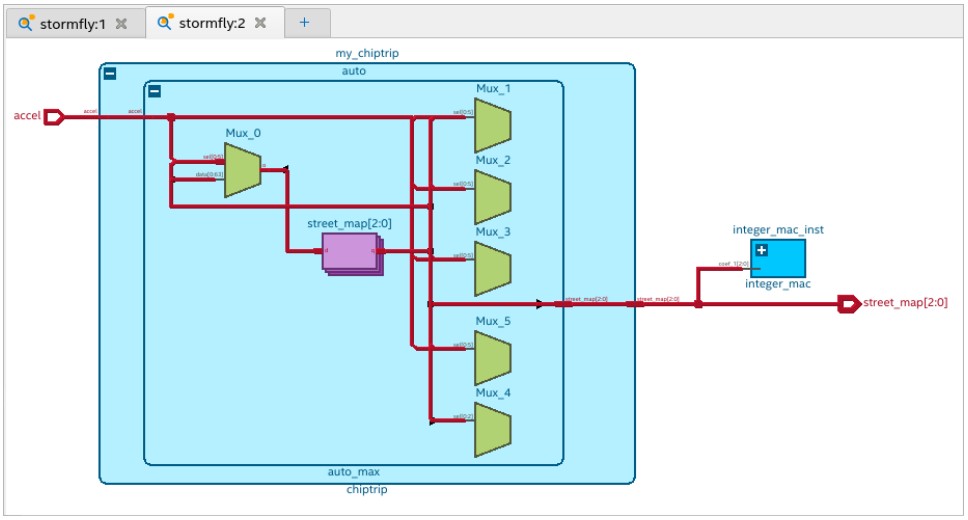


Figure 25. Example of Displaying a Node in a New Tab in the Schematic Viewer



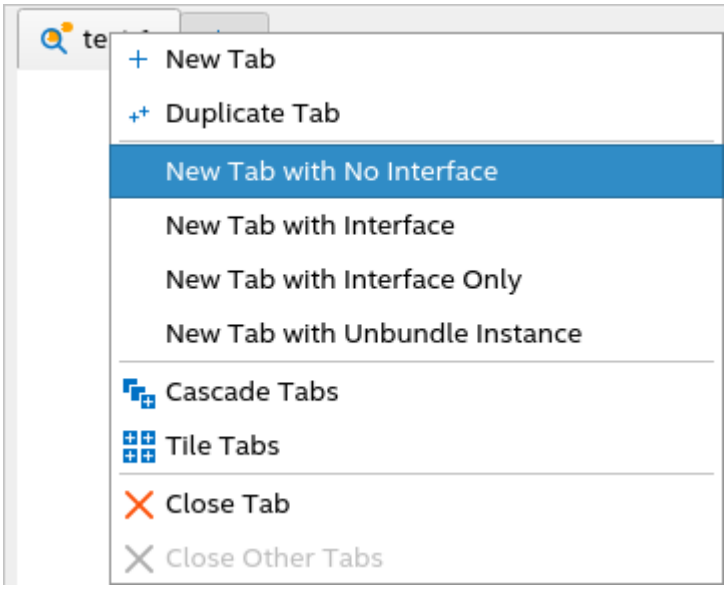
Bottom Pane

The bottom pane lists all netlist objects, such as ports, instances, instance ports, and nets, currently selected in the schematic viewer. Using the context-sensitive menu, you can view the object in a new tab or locate the node in the design file.

1.6.1.3. Module Interfaces

The schematic viewer in the RTL Analyzer supports viewing module interface for designs containing interfaces in the SystemVerilog. You can access the interface modes by launching your design in the schematic viewer and selecting one of the interface modes from the tab's context-sensitive menu. The RTL Analyzer supports three interface modes, as shown in the following image:

Figure 27. Interface Modes



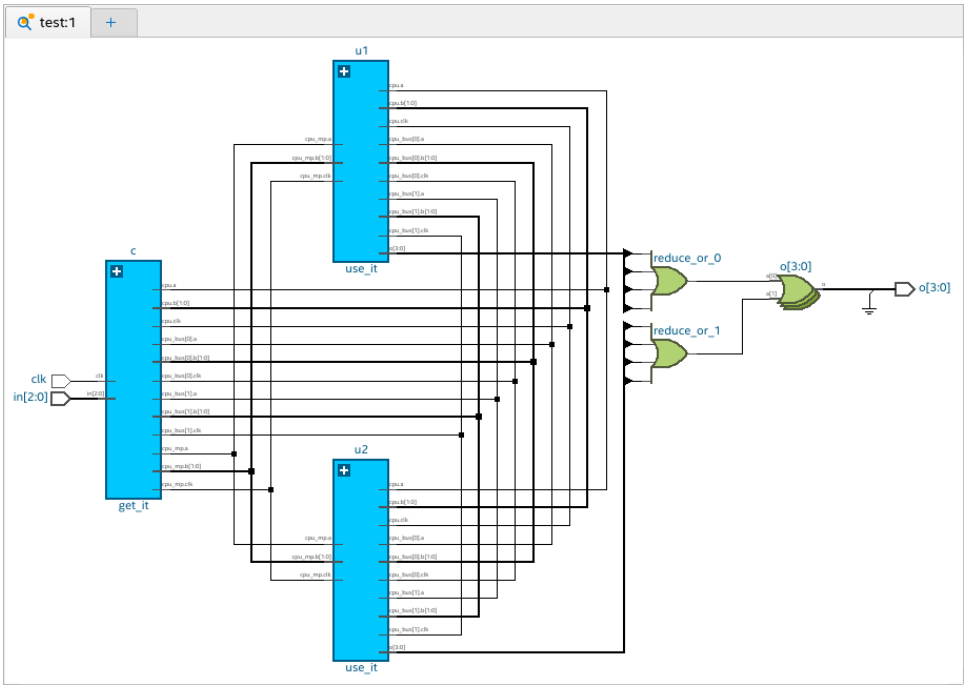
Note: For details about the unbundle instance mode, refer to Bundled Instances on page 40.

No Interface Mode (Default)

This is the default mode where all ports display individually, even if they are part of an interface.

In the following example, you can observe that all ports for c, u1, and u2 modules are listed without any grouping of ports.

Figure 28. No Interface Mode

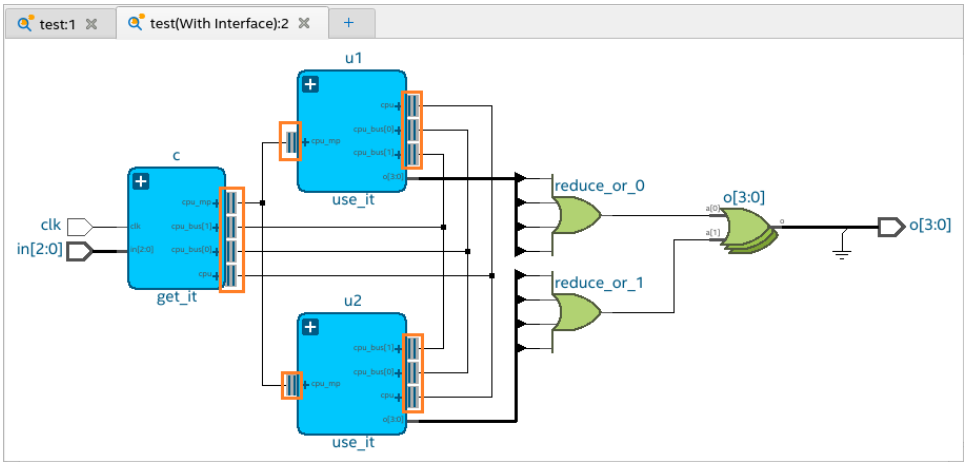


With Interface Mode

In this mode, all ports of an interface are grouped to show a compact and simplified view of the schematic. Interfaces have an expand button "+" and double vertical lines beside the port. You can expand each interface to view all ports under the interface.

In the following example, you can observe that in the u1 module, all ports that are part of cpu, cpu_bus[0], cpu_bus[1], and cpu_mp interfaces are grouped (indicated by the '+' symbol). Each interface has two dark gray lines marked beside them.

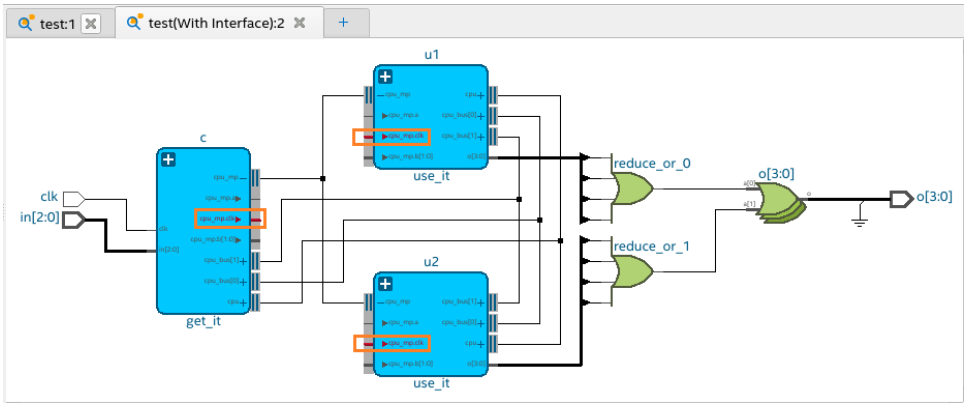
Figure 29. With Interface Mode



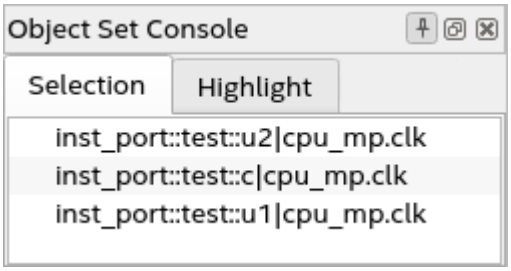
In this mode, all related ports (input/output) are automatically highlighted when you expand an interface and select any port under the interface.

In the following example, you can see that when you expand the `cpu_mp` interface in `c`, `u1`, and `u2` modules and select the `cpu_mp.clk` port in one of the modules, all related ports (highlighted in red) are automatically highlighted, as shown in the following image:

Figure 30. With an Interface Expanded



Note: You can also view these related ports in the Object Set Console Window.

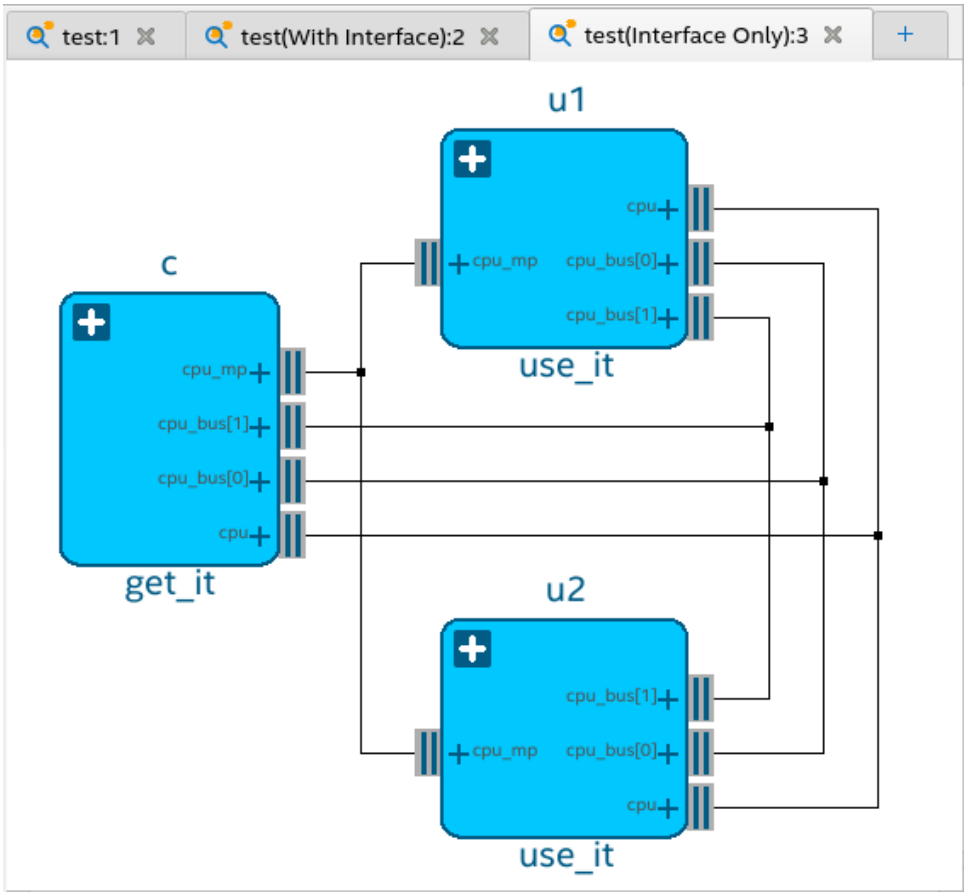


Interface Only Mode

In this mode, only modules with interfaces are shown in the schematic viewer. All non-interface ports and objects are removed. This mode is helpful in viewing the relationship between modules that are connected by an interface. You can expand each interface to view all ports under the interface.

In the following example, you can observe that the non-interface port `o[3:0]` is removed in `c` and `u1` modules, and all other objects are no longer visible in the schematic viewer.

Figure 31. Interface Only Mode



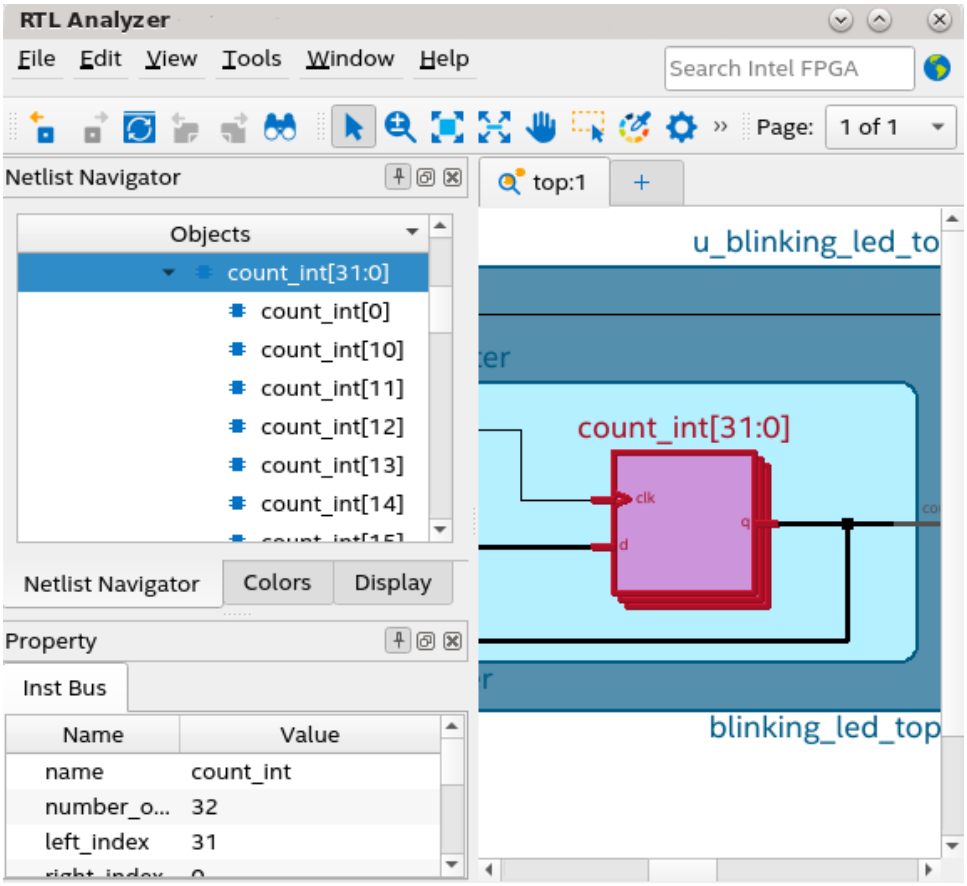
1.6.1.4. Bundled Instances

To improve schematic readability, the RTL Analyzer bundles single-bit instances into a collection known as a bundled instance and hides unconnected pins of instances by default.

Bundled instances appear stacked, and the indexes of the bits appear in the instance name as left and right indexes. You can expand bundled instances to view single-bit instance members in the hierarchy browser. You can view more information about a bundled instance, such as the number of instances, left and right index, and instance members from the Property viewer.

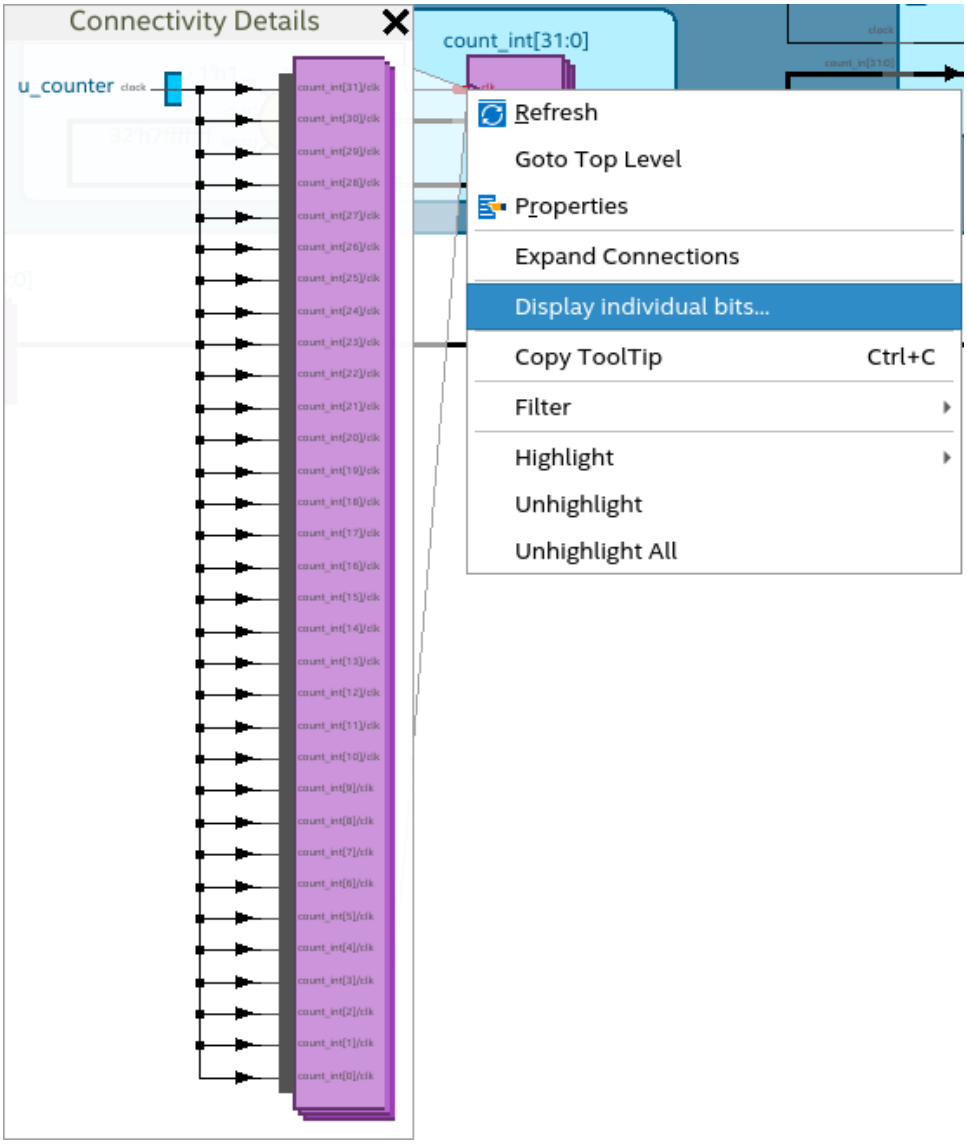
In the following figure, `count_int[31:0]` is a bundled instance, where 31 is the left index and 0 is the right index. Under the **Objects** pane (hierarchy browser), `count_int[0]`, `count_int[10]`, `count_int[11]`, and so on are single-bit instance members.

Figure 32. Bundled Instance



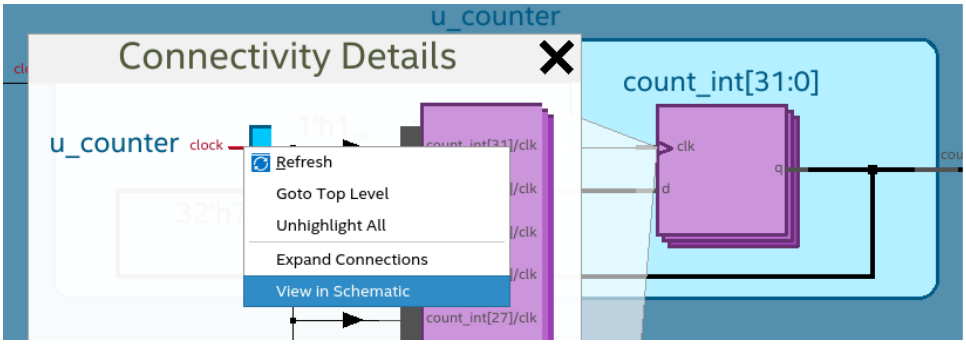
If you want to view individual bits of a bundled instance within the schematic viewer, select the pin bus, right-click to display the context-sensitive menu, and click **Display Individual Bits**. The **Connectivity Details** dialog displays connectivity details of the single-bit instances.

Figure 33. Connectivity Details of Single-bit Instances



Within the **Connectivity Details** dialog box, you can select the first individual instance, right-click to display the context-sensitive menu, and select **View in Schematic**. The first object in the hierarchy gets highlighted automatically.

Figure 34. View Connectivity Details in the Schematic Viewer

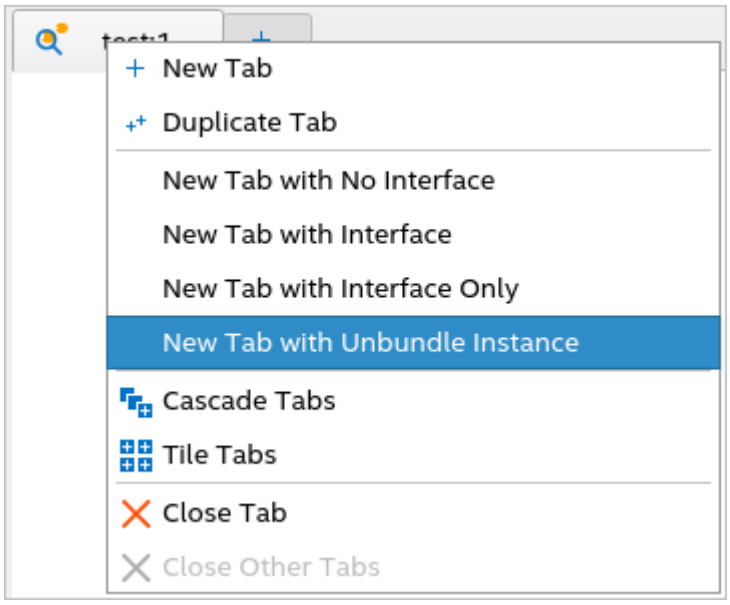


Viewing Unbundled Instances

To view unbundled instances, launch your design in the schematic viewer and use one of the following options:

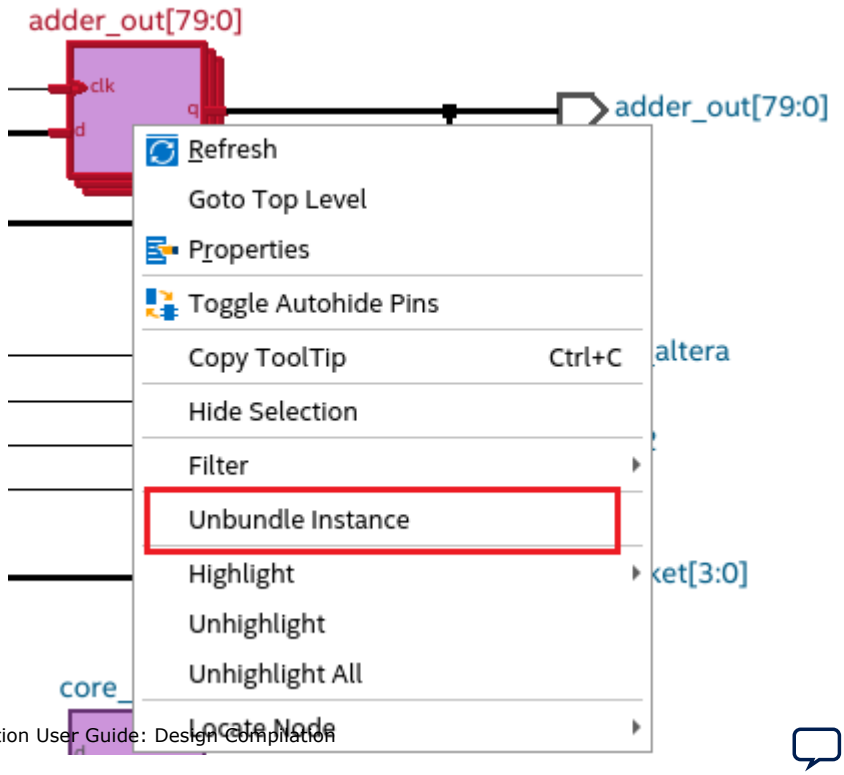
- Select **New Tab with Unbundle Instance** from the tab's context-sensitive menu.

Figure 35. Opening a New Tab with Unbundled Instances



- Select **Unbundle Instance** option from the context-sensitive menu.

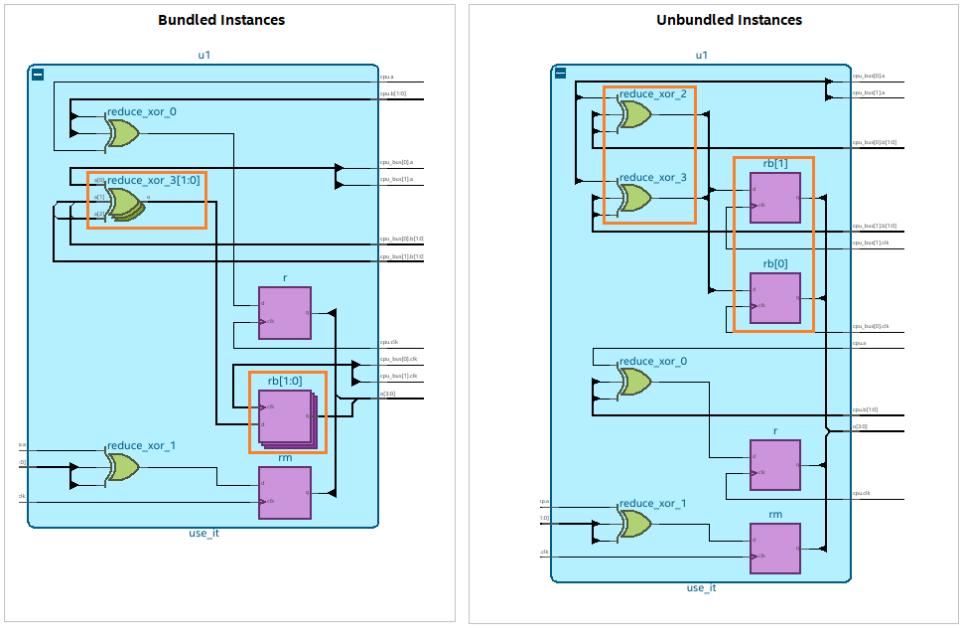
Figure 36. Viewing Unbundled Instances



Using either option, you get an unbundled view of the bundled instances in a new tab of the schematic viewer. This mode is helpful when you want to view single-bit instances of a bundled object.

In the following example, when you view the design in the default no interface mode (left), you observe that the instances `reduce_xor_3[1:0]` and `rb[1:0]` are bundled. However, within the unbundle instance mode (right), you can view single-bit instances `reduce_xor_3`, `reduce_xor_2`, `rb[1]`, and `rb[0]`.

Figure 37. Example of Bundled and Unbundled Instances



Related Information

Module Interfaces on page 37

1.6.1.5. Auto-hide Unconnected Pins

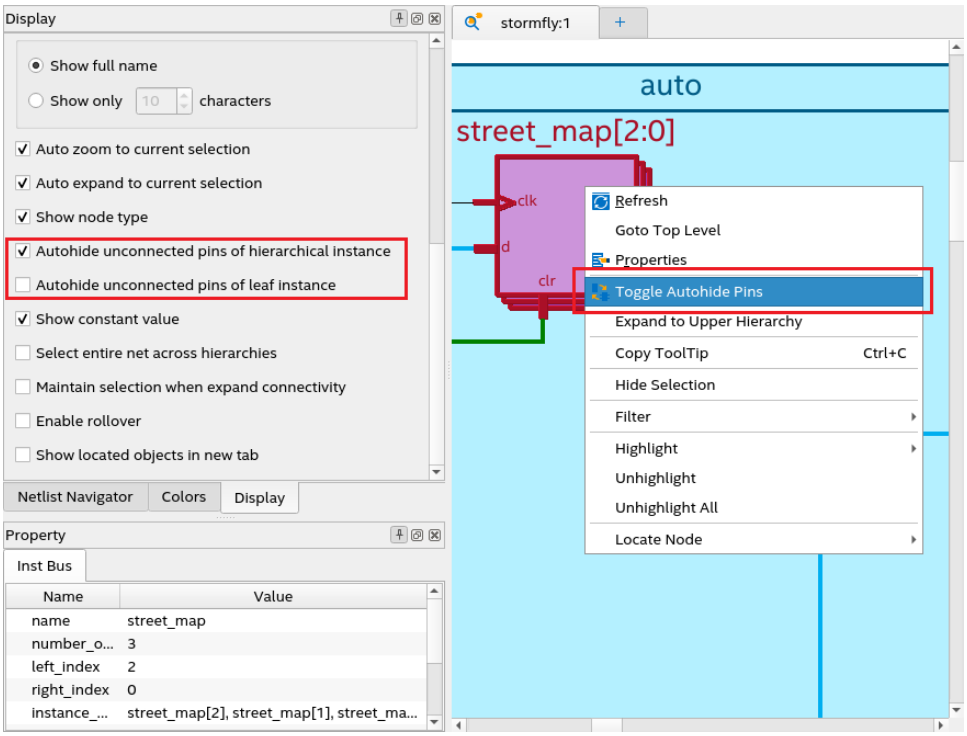
The **Autohide unconnected pins of hierarchical instance** option is enabled by default. However, **Autohide unconnected pins of leaf instance** is disabled by default, which means that the schematic viewer displays all instance ports even when the connected net is not displayed on the same schematic. This allows you to further expand the connectivity of the hanging instance port without requiring you to toggle autohide pins option first.

Note: Instance ports that are configured with default connection are hidden in the schematic viewer to improve schematic readability. For example, RAM block.

Auto-hiding unconnected pins makes the instance symbol less cluttered by hiding pins that are not connected to any net. You can disable both auto-hide unconnected pins options when you want to view all pins or obtain properties of unconnected pins.

You can disable auto-hide unconnected pins options either for all instances in the **Display** setting or for a particular instance using the context-sensitive menu, as shown in the following image:

Figure 38. Auto-hiding Unconnected Pins

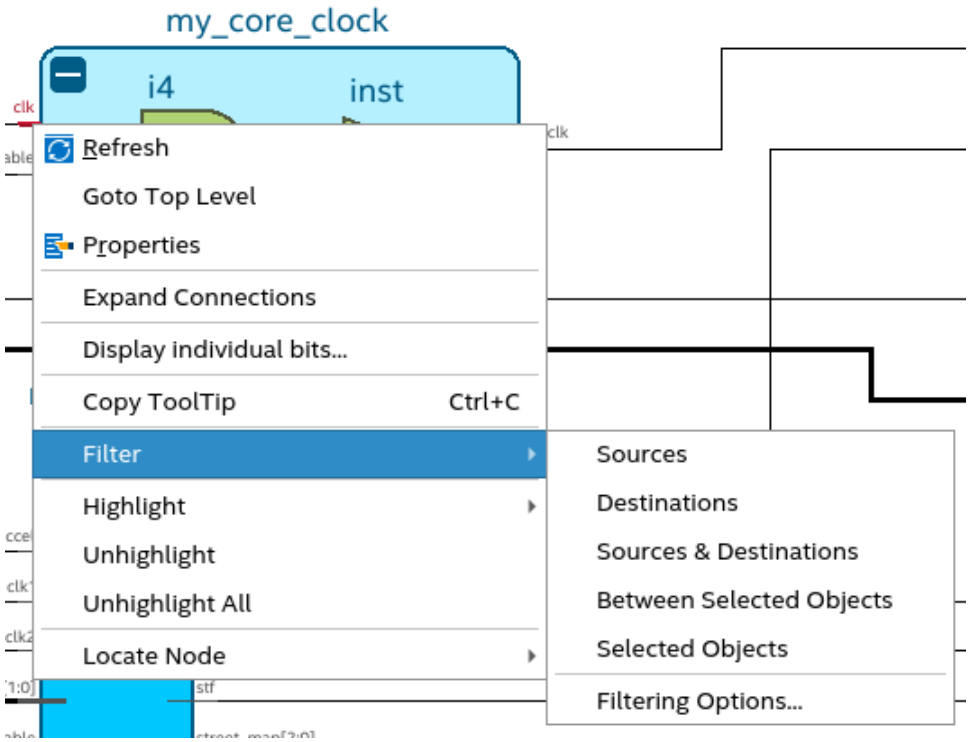


1.6.1.6. Filtering Ports and Nets

Filtering allows you to filter your netlist and view only a desired logic path. To filter the netlist, select ports, nodes, or node ports in the desired path.

You access the **Filter** feature either from the netlist hierarchy browser or the schematic viewer of the RTL Analyzer GUI. Select and right-click on the desired design object (port or node), point to **Filter**, and select the desired filter command in the context-sensitive menu, as shown in the following image. The RTL Analyzer generates a new page in the schematic view displaying the netlist that remains after filtering.

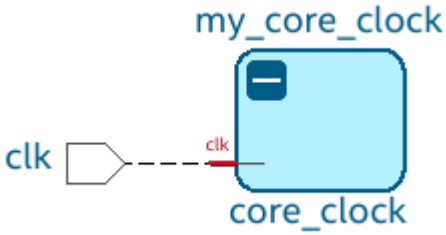
Figure 39. Filtering the Design Objects



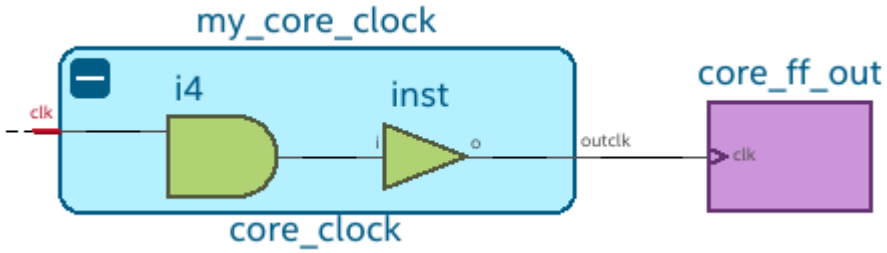
The following filtering commands are available:

Note: **Sources**, **Destinations**, **Sources & Destinations**, and **Between Selected Objects** are visible only for ports.

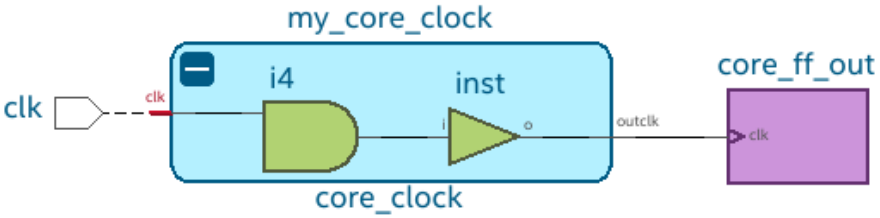
- **Sources**: Displays only the input source nodes that feed the selected port. In the following example image, when you use the **Sources** filter for the selected instance port `clk`, the source port `clk` displays:



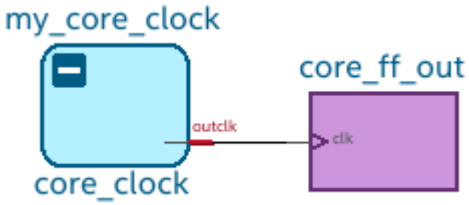
- **Destinations**: Displays only the fan-out destination nodes fed by the selected port. In the following example image, when you use the **Destinations** filter for the selected instance port `clk`, the destination instance port `outclk` displays:



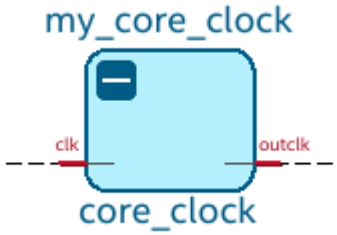
- **Sources & Destinations**: This option is a combination of source and destination filtering commands. The filtered view displays both sources and destinations of the selected port. In the following example image, when you use the **Sources & Destinations** filter for the selected instance port `clk`, you can view the source and destination nodes:



- **Between Selected Objects**: Displays nodes and connections in the path between the selected ports. In the following image, when you use the **Between Selected Objects** filter for the selected instance port `clk`, all objects between the input and output instance ports displays:



- **Selected Objects**: Displays only the selected objects. In the following image, when you use the **Selected Objects** filter, only the selected instance ports `clk` and `outclk` displays:



For all the filtering commands, the RTL Analyzer stops tracing through the netlist to obtain the filtered netlist when it reaches the port or instances. You can further control the filtering result by selecting **Filter ➤ Filtering Options** in the context-sensitive menu. The **Filtering Options** dialog box provides the following filtering options:

- **Stop filtering at register** (optional): The filtering stops when a register is encountered. This option is turned on by default.
- **Stop filtering at design partition** (optional): The filtering stops at the design partition. This option is turned on by default.
- **A specified number of logic levels**: The logic levels are counted from the selected port. The default value is 5 to ensure optimal processing time when performing filtering. However, you can specify a value between 1 and 15.

Note: A progress bar is shown while filtering is in progress. Click **Stop trace** to halt if the process takes too long to complete.

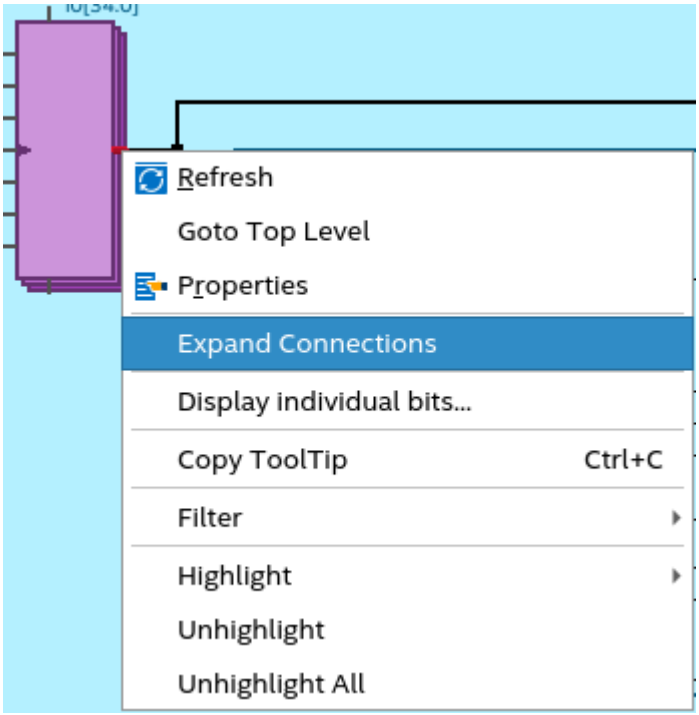
1.6.1.7. Expand Connections

Expand connections, which was previously supported in the Netlist Viewer, is also supported in the RTL Analyzer and useful when performing filtering. Expand Connections allow you to expand a selected pin and reveal the next connected node.

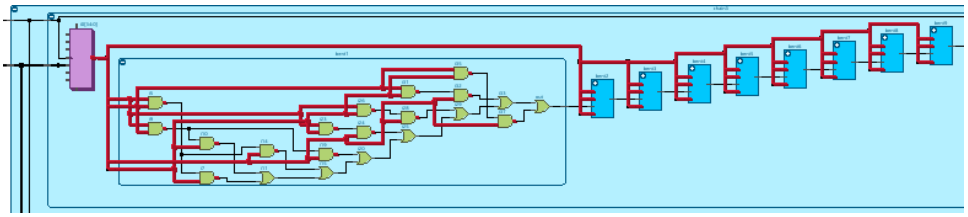
Note: Before using the **Expand Connections** feature, disable the Auto-hide Unconnected Pins in the Display settings.

To use this feature, select a pin, right-click and select **Expand Connections** in the context-sensitive menu, as shown in the following image:

Figure 40. Expand Connections



The logical path from the selected pin to the next connected node is highlighted, as shown in the following image:



1.6.2. Use Case Examples

In this section, you can explore the following examples:

Scripting Routine Tasks Using Tcl Commands on page 50

Traversing the Design Netlist Using Tcl Commands on page 52

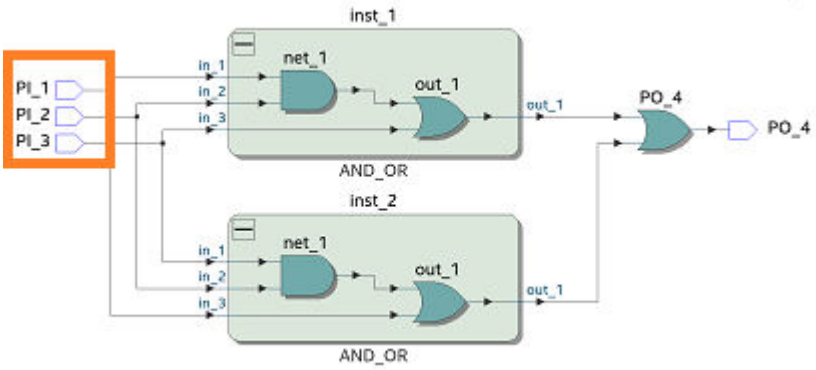
1.6.2.1. Scripting Routine Tasks Using Tcl Commands

The following examples enumerate how you can easily script a few routine tasks, such as attribute-based object filtering, name-based searching, or traversing object relationships, using different features of the `get` command:

Example 1. Retrieving Top-level Input Ports (highlighted in the schematic)

In some scenarios, you may need a list of top-level ports. For example, to ensure all input ports are constrained with `set_input_delay`, you must retrieve all ports in the design and filter based on `direction` using the `dni::get_ports` tcl command, as shown in the following:

```
dni::get_ports -filter direction==input
```



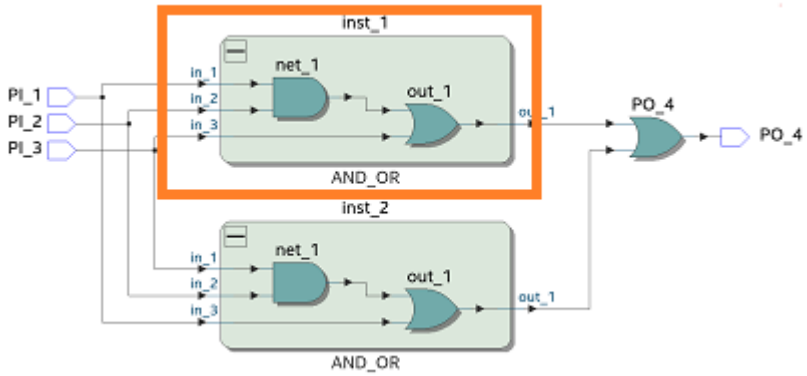
The tcl command returns a collection of input ports in the design, such as `PI_1`, `PI_2`, and `PI_3`. You can use the returned collection to pass or chain into other `get_object` commands. To access members of the collection, use collection iterators, such as `foreach_in_collection`. For example:

```
foreach_in_collection p [dni::get_ports -filter direction=input] { puts $p }
port::top::PI_1
port::top::PI_2
port::top::PI_3
```

Example 2. Retrieving the `inst_1` Instance

Within your design, you can search for a specific object (for example, instance) by name. To find an instance, use the `dni::get_cells` tcl command, as shown in the following:

```
dni::get_cells inst_1
```



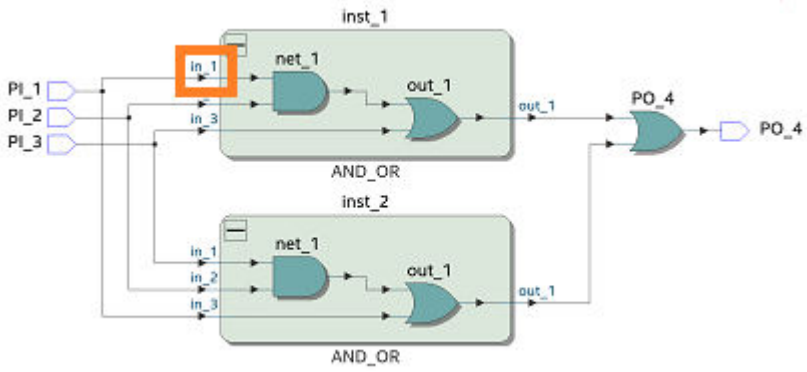
The tcl command returns the `inst_1` instance in the design. You can use the returned instance to pass or chain into other `get_object` commands. For example:

```
foreach_in_collection p [dni::get_cells inst_1] { puts [dni::get_property -name
name -object $p] }
inst_1
```

Example 3. Retrieving the `in_1` Instance Port of the `inst_1` Instance

During connectivity traversal, you may want to find a specific instance pin of an instance. In such scenarios, use the `-of_object` interface of the `dni::get_pins` command to traverse instance-to-instance pin relationship as shown in the following:

```
dni::get_pins -of_objects [dni::get_cells inst_1] -filter name==in_1
```



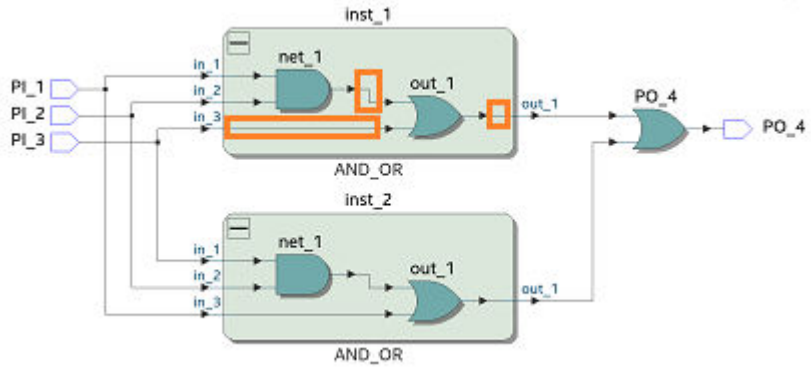
The tcl command returns `in_1` instance port of the `inst_1` instance in the design. You can use the returned instance port to pass or chain into other `get_object` commands. For example:

```
foreach_in_collection p [dni::get_pins -of_objects [dni::get_cells inst_1] -
filter name==in_1] { puts $p }
inst_port::top::inst_1|in_1
```

Example 4. Retrieving Nets of the `inst_1|out1` Instance

During connectivity trace, you may want to traverse all nets connected to an instance. In such a scenario, use the `-of_objects` interface of the `dni::get_nets` command to traverse nets connected to an instance. For example:

```
dni::get_nets -of_objects [dni::get_cells inst_1|out_1]
```



The tcl command returns nets of the `inst_1|out1` instance in the design. You can use the returned collection to pass or chain into other commands. For example:

```
foreach_in_collection p [dni::get_nets -of_objects [dni::get_cells inst_1|
out_1] ] { puts $p }
inst_port::top::inst_1|out_1|Net_7
inst_port::top::inst_1|out_1|Net_6
inst_port::top::inst_1|out_1|Net_8
```

Example 5. Listing the Properties Available on the `net` Object Type

The tcl interface allows you to access object schema via the `dni::list_properties` tcl command. You can use this command to list accessible properties of any object type as follows:

```
dni::list_properties -type net
name parent_name number_of_ports ports net_bus_name source_file source_line
is_user_declared
```

1.6.2.2. Traversing the Design Netlist Using Tcl Commands

In this topic, you can see examples of Tcl commands that you can use to extract basic netlist objects or selective objects of your design using filters, pattern, and other techniques.

Traversing Basic Design Objects

The following table lists Tcl command examples for extracting hierarchical instances, instance ports, nets, and ports from your design netlist:

Note: When using `dni::` Tcl commands in an SDC file, you can skip adding `dni::` as `read_sdc` adds the namespace correctly.

Table 6. Tcl Commands for Traversing Design Objects

Task	Tcl Command Example
Get all instances directly under the top module.	`foreach_in_collection obj [dni::get_cells] { puts $obj; }`
Get all design instances at all hierarchies.	`foreach_in_collection obj [dni::get_cells -hierarchical] { puts $obj; }`
Get all instances directly under a hierarchical instance using a pattern with the hierarchy separator (|).	`foreach_in_collection obj [dni::get_cells auto|*] { puts $obj; }`
Get all direct child instances of a specific hierarchical instance (for example, the instance `auto`).	# Use `dni::current_instance` to move the browsing scope to the `auto` hierarchical instance. `dni::current_instance auto` # Get all direct child instances of the hierarchical instance `auto`. `foreach_in_collection obj [dni::get_cells] { puts $obj; }` # Set the scope back to top design `dni::current_instance`
Get all nets directly under the top module.	`foreach_in_collection obj [dni::get_nets] { puts $obj; }`
Get all nets of the design at all hierarchies.	`foreach_in_collection obj [dni::get_nets -hierarchical] { puts $obj; }`
Get all nets directly under a hierarchical instance using a pattern with the hierarchy separator (|).	`foreach_in_collection obj [dni::get_nets tick|*] { puts $obj; }`
Get all nets directly under a specific hierarchical instance (for example, the instance `tick`).	# Use `dni::current_instance` to move the browsing scope to the `tick`hierarchical instance. `dni::current_instance tick` # Get all nets directly under `tick` `foreach_in_collection obj [dni::get_nets] { puts $obj; }` # Set the scope back to top design `dni::current_instance`
Get all ports of the design.	`foreach_in_collection obj [dni::get_ports] { puts $obj; }`
Get all instance ports of direct child instances under the design top module.	`foreach_in_collection obj [dni::get_pins] { puts $obj; }`
Get all instance ports of all child instances under a specific hierarchical instance (for example, `tick`).	# Use `dni::current_instance` to move the browsing scope to the `tick` hierarchical instance. `dni::current_instance tick` # Get all instance ports under `tick` `foreach_in_collection obj [dni::get_pins] { puts $obj; }` # Set the scope back to the top design `dni::current_instance`

continued...

Task	Tcl Command Example
Get all instance ports on all child instances using the object ID format to specify a hierarchical instance.	`set inst [get_cells tick]` `foreach_in_collection obj [dni::get_pins -of_objects $inst] { puts $obj; }`
	`foreach_in_collection obj [dni::get_pins -of_objects tick|ticket[1]] { puts $obj; }`
	`foreach_in_collection obj [dni::get_pins -of_objects instance::chiptrip::tick|ticket[1]] { puts $obj; }`

Getting Selective Design Objects

You can use the following techniques to refine your search criteria and extract select design objects from your netlist:

- **<patterns>**: It extracts objects by matching patterns in `dni::get_cells`, `dni::get_nets`, and `dni::get_pins` commands. Patterns can include wildcard characters * or ?. Wildcard characters do not match with the hierarchical separator.

  For objects in the design hierarchy:

  — The hierarchical path name of the object is matched against the pattern.

  — The hierarchical separator | splits the pattern into individual subpatterns, each of which is matched against one level of hierarchy. This means that a * does not match the hierarchy.

  — The name being matched is relative to the current search root (design top module). You can change the current search root using the `dni::current_instance` command.

Table 7. Getting Selective Design Objects Using <pattern>

Tasks Using <pattern>	Tcl Command Example
Get all first-level hierarchical and leaf instances.	`foreach_in_collection obj [dni::get_cells *] { puts $obj; }`
Get all second-level hierarchical and leaf instances.	`foreach_in_collection obj [dni::get_cells *|*] { puts $obj; }`
Get all instances under the hierarchical instance, for example, `auto`.	`foreach_in_collection obj [dni::get_cells auto|*] { puts $obj; }`
Get all nets under the top module.	`foreach_in_collection obj [dni::get_nets *] { puts $obj; }`
Get all nets under the first level hierarchical and leaf instances.	`foreach_in_collection obj [dni::get_nets *|*] { puts $obj; }`
Get all nets under the first-level hierarchical instance, for example, `auto`.	`foreach_in_collection obj [dni::get_nets auto|*] { puts $obj; }`
Get all instance ports on first-level hierarchical and leaf instances.	`foreach_in_collection obj [dni::get_pins *|*] { puts $obj; }`
Get all instance ports on the hierarchical instance, for example, `auto`.	`foreach_in_collection obj [dni::get_pins auto|*] { puts $obj; }`
Get all instance ports on all the child instances of the hierarchical instance, for example, `auto`.	`foreach_in_collection obj [dni::get_pins auto|*|*] { puts $obj; }`

- **<-of_objects>**: It extracts related design objects of the input object collection for `dni::get_cells`, `dni::get_nets`, `dni::get_pins`, and `dni::get_ports`. The following object relationships are considered:
  - `dni::get_cells -of_objects <inst_port|net>`
  - `dni::get_nets -of_objects <inst_port|port|instance>`
  - `dni::get_pins -of_objects <instance|net>`
  - `dni::get_ports -of_objects <net>`

Table 8. Getting Selective Design Objects Using <-of_objects>

Tasks Using <-of_objects>	Tcl Command Example
Get the hierarchical or leaf instance on which the input instance port is.	`foreach_in_collection obj [dni::get_cells -of_objects inst_port::chiptrip::auto|dir[0]] { puts $obj; }` `foreach_in_collection obj [dni::get_cells -of_objects inst_port::chiptrip::auto|street_map[2]|q] { puts $obj; }`
Get the hierarchical or leaf instances to which the input net is connected.	`foreach_in_collection obj [dni::get_cells -of_objects net::chiptrip::wire_get_ticket1] { puts $obj; }` `foreach_in_collection obj [dni::get_cells -of_objects net::chiptrip::auto|n5] { puts $obj; }` `foreach_in_collection obj [dni::get_cells -of_objects [dni::get_nets *]] { puts $obj; }` `foreach_in_collection obj [dni::get_cells -of_objects [dni::get_nets auto|*]] { puts $obj; }`
Get the net connecting a leaf instance port.	`foreach_in_collection obj [dni::get_nets -of_objects inst_port::chiptrip::auto|street_map[0]|q] { puts $obj; }`
Get the net connecting a hierarchical instance port.	`foreach_in_collection obj [dni::get_nets -of_objects inst_port::chiptrip::auto|get_ticket] { puts $obj; }`
Get the nets connecting to all the instance ports of a leaf instance.	`foreach_in_collection obj [dni::get_nets -of_objects instance::chiptrip::auto|street_map[0]] { puts $obj; }`
Get the nets connecting to all the instance ports of a hierarchical instance.	`foreach_in_collection obj [dni::get_nets -of_objects instance::chiptrip::auto] { puts $obj; }`
Get the net connecting to a primary port of the design.	`foreach_in_collection obj [dni::get_nets -of_objects port::chiptrip::gt1] { puts $obj; }`
Get instance ports on a leaf instance.	`foreach_in_collection obj [dni::get_pins -of_objects instance::chiptrip::tick|add_0] { puts $obj; }`
Get instance ports on a hierarchical instance.	`foreach_in_collection obj [dni::get_pins -of_objects instance::chiptrip::tick] { puts $obj; }`
Get the instance ports on the hierarchical instance, for example, `instance::chiptrip::auto`.	`foreach_in_collection obj [dni::get_pins -of_objects [dni::get_cells auto]] { puts $obj; }`
Get instance ports connected by a hierarchical net.	`foreach_in_collection obj [dni::get_pins -of_objects net::chiptrip::wire_get_ticket1] { puts $obj; }`
Get primary ports connected by a hierarchical net on the top level.	`foreach_in_collection obj [dni::get_ports -of_objects net::chiptrip::wire_get_ticket1] { puts $obj; }`

- **<-filter>**: It extracts objects by filtering objects returned from `dni::get_cells`, `dni::get_nets`, `dni::get_pins`, and `dni::get_ports` based on the filter expression. The filter expression verifies every object that satisfies the other search criteria. If the object does not satisfy the filter, it is not returned in the result.

  A filter expression comprises predicates combined using logical operators, such as `&&`, `and`/`AND`, `||`, `or`/`OR`, and `!`. Parentheses `( )` in the expression override precedence. Each predicate has the following form:

  ```
  prop_name relop value
  ```

  where,

  — *`prop_name`* is the name of the object property being evaluated.

  — *`value`* is a number or a string. You can provide simple strings without the surrounding double-quotes.

  — *`relop`* is one of the following comparison operators:

    - == (equal)
    - != (not equal)
    - =~ (matches, value is then a pattern)
    - !~ (does not match, value is then a pattern)
    - > (greater than)
    - < (less than)
    - >= (greater than or equal to)
    - <= (less than or equal to)

  For example: `-filter {name=~"out[*]" && ! (master_name==mod_1 || parent_name != block_1)}`

Table 9. Getting Selective Design Objects Using <-filter>

Tasks Using <-filter>	Tcl Command Example
Get all child instances under the top-level.	`foreach_in_collection obj [dni::get_cells *] { puts $obj; }`
Get properties of the instance object type.	`dni::list_properties -type instance`
Get objects that are not a leaf instance	`foreach_in_collection obj [dni::get_cells * -filter {is_leaf==1}] { puts $obj; }`
Get all objects that are hierarchical instances.	`foreach_in_collection obj [dni::get_cells * -filter {is_leaf==0}] { puts $obj; }`
Return hierarchical instances matching the name pattern (for example, `t*`) under the design top-level module.	`foreach_in_collection obj [dni::get_cells * -filter {is_leaf==0 && name=~"t*"}] { puts $obj; }`
Get all instances under the hierarchical instance, for example, `auto` that matches the module name in the filter.	`foreach_in_collection obj [dni::get_cells auto|*] { puts $obj; }` `dni::get_property -name module_name -object instance::chiptrip::auto|Mux_0`

continued...

Tasks Using <-filter>	Tcl Command Example
	foreach_in_collection obj [dni::get_cells auto|* -filter {module_name=="primitive_lib_WORKING_LIBRARY_INTERNAL/OPER_(MUX)"}] { puts $obj; }
Get properties that you can use to filter net objects. Then, get nets with three connected instance ports and primary ports under the top-level design.	dni::list_properties -type net foreach_in_collection obj [dni::get_nets -filter {number_of_ports==3}] { puts $obj; }
Get all nets in the full design hierarchies that are under the parent module, for example, auto_max.	foreach_in_collection obj [dni::get_nets -hierarchical -filter {parent_name=="auto_max"}] { puts $obj; }
Get all nets in the full design hierarchies that are associated with the net_bus, for example, street_map.	foreach_in_collection obj [dni::get_nets -hierarchical -filter {net_bus_name=="street_map"}] { puts $obj; }
Get properties that you can use to filter instance port objects and get the value of the direction property. Then, get all output inst_ports under the top-level design.	dni::list_properties -type inst_port dni::get_property -name direction -object inst_port::chiptrip::auto|get_ticket foreach_in_collection obj [dni::get_pins -filter {direction==output}] { puts $obj; }
Get all primary input ports in the design.	foreach_in_collection obj [dni::get_ports -filter {direction==input}] { puts $obj; }

- **Combined use of <pattern>, -of_objects <obj_col>, -hierarchical, and -filter <filter_expr>**:

Table 10. Combined Search Criteria

Tasks Using Combined Search Criteria	
Get all instance ports under the design top-module that match the combined search criteria of -of_objects and -filter.	foreach_in_collection obj [dni::get_pins -of_objects [dni::get_cells *] -filter {direction==input}] { puts $obj; }
Get all nets under the design top module that match the pattern and -hierarchical.	foreach_in_collection obj [dni::get_nets time* -hierarchical] { puts $obj; }

The following example shows the limitation of the combined search criteria:

```
tcl> foreach_in_collection obj [dni::get_pins clk -hierarchical]  { puts $obj; }
inst_port::chiptrip::auto|clk
inst_port::chiptrip::speed|clk
inst_port::chiptrip::tick|clk
inst_port::chiptrip::time_c|clk


tcl> foreach_in_collection obj [dni::get_nets time* -of_objects
instance::chiptrip::time_c] { puts $obj; }
Arguments '-of_objects' and 'patterns' are mutually exclusive


tcl> foreach_in_collection obj [dni::get_pins -hierarchical -of_objects
instance::chiptrip::time_c]  { puts $obj; }

Arguments '-of_objects' and '-hierarchical' are mutually exclusive
```

1.7. Design Synthesis

The Quartus Prime compiler's Analysis & Elaboration module analyzes your complete design source files, such as standards-compliant Verilog HDL (`.v`), VHDL (`.vhd`), SystemVerilog (`.sv`), and the Verilog Quartus Mapping (`.vqm`) generated from other EDA tools, and constraint files (`.sdc` or `.rtlsdc`) and provides an unmodified view of your design early in the compilation flow. The SDC-on-RTL constraints are applied to the elaborated netlist.

In the Synthesis stage, the compiler synthesizes and translates your design files into an atom netlist for mapping to device resources. During synthesis, timing constraints that were read during elaboration are propagated to the synthesized atom netlist.

Caution: Starting from the Quartus Prime Pro Edition software version 23.3, the compiler cannot synthesize schematic Block Design File (`.bdf`). You must convert it to an acceptable format, such as Verilog or VHDL using only the Quartus Prime Standard Edition (not possible with the Pro Edition) command `quartus_map` as shown in the following:

- To convert your `.bdf` file to Verilog Design File (`.v`):

```
quartus_map <project_name> --convert_bdf_to_verilog=<bdf_file_name>
```

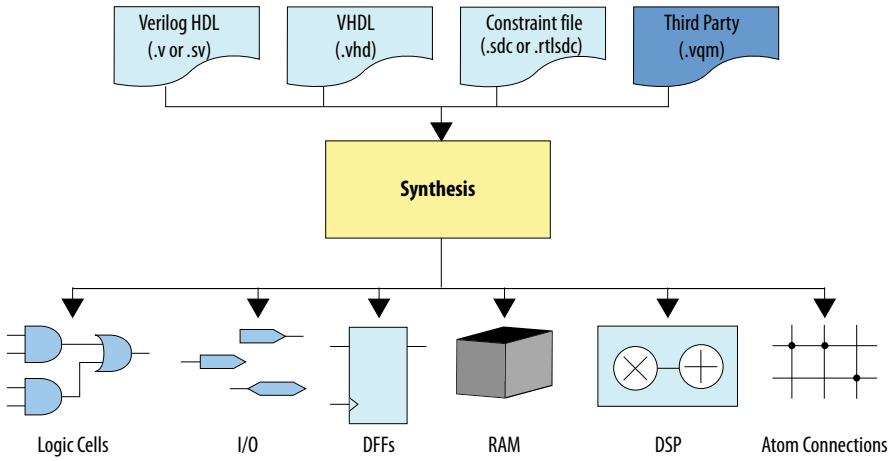
- To convert your `.bdf` file to VHDL Design File (`.vhd`):

```
quartus_map <project_name> --convert_bdf_to_vhdl=<bdf_file_name>
```

Synthesis examines the logical completeness and consistency of the design, and checks for boundary connectivity and syntax errors. Synthesis also minimizes and optimizes design logic. For example, synthesis infers D flip flops, latches, and state machines from "behavioral" languages, such as Verilog HDL, VHDL, and SystemVerilog. Synthesis may replace operators, such as + or –, with modules from the Quartus Prime IP library, when advantageous. During synthesis, the Compiler may change or remove user logic and design nodes. Quartus Prime synthesis minimizes gate count, removes redundant logic, and ensures efficient use of device resources.

At the end of synthesis, the Compiler generates an atom netlist. Atom refers to the most basic hardware resource in the FPGA device. Atoms include logic cells organized into look-up tables, D flip flops, I/O pins, block memory resources, DSP blocks, and the connections between the atoms. The atom netlist is a database of the atom elements that design synthesis requires to implement the design in silicon.

Figure 41. Design Synthesis



1.7.1. Preparing for Design Synthesis

Before running synthesis, apply any of the following settings and constraints that impact synthesis:

- To specify options for the synthesis of Verilog HDL input files, click **Assignments ➤ Settings ➤ Verilog HDL Input**.
- To specify options for the synthesis of VHDL input files, click **Assignments ➤ Settings ➤ VHDL Input**.
- To specify options that affect compilation processing time, click **Assignments ➤ Settings ➤ Compilation Process Settings**.
- To specify the Compiler's high-level optimization strategy and other options, click **Assignments ➤ Settings ➤ Compiler Settings**. Specify the optimization goal, according to Compiler Optimization Modes on page 149.
- On the **Compiler Settings** page enable or disable the **Enable Intermediate Fitter Snapshots** option to preserve snapshots for the Plan, Place, Route, and Retime stages any time you run full compilation. The Compiler does not generate intermediate snapshots by default.
- To specify advanced synthesis settings, click **Assignments ➤ Settings ➤ Compiler Settings**, and then click **Advanced Settings (Synthesis)**.
- Consider enabling fractal synthesis for arithmetic-intensive designs that exhaust all DSP resources, according to the guidelines in Fractal Synthesis Optimization on page 155.
- To register your SDC-on-RTL files and apply them to the elaboration netlist, refer to Registering the SDC-on-RTL SDC File on page 65 and Applying the SDC-on-RTL Constraints on page 66.

1.7.2. Running Synthesis

Run design synthesis as part of a full compilation, or as an independent process. Before running synthesis, specify settings that control synthesis processing. The Messages window dynamically displays processing information, warnings, or errors.

Following Analysis & Elaboration processing, the Synthesis report provides detailed information about the synthesis of your project. To run synthesis, perform the following steps.

1. Review Preparing for Design Synthesis on page 59.
2. Create or open an Quartus Prime project with valid design files for compilation.
3. Click **Synthesis** on the Compilation Dashboard.

Related Information

Synthesis Settings Reference on page 168

1.7.2.1. Preserving Registers During Synthesis

Quartus Prime synthesis minimizes gate count, merges redundant logic, and ensures efficient use of device resources. If you need to preserve specific registers through synthesis processing, you can specify any of the following entity-level assignments.

Assign the **Preserve Registers in Synthesis** or **Preserve Fan-Out Free Register Node** options to allow Fitter optimization of the preserved registers. **Preserve Registers** restricts Fitter optimization of the preserved registers. Specify synthesis preservation assignments by clicking **Assignments ➤ Assignment Editor**, by modifying the `.qsf` file, or by specifying synthesis attributes in your RTL.

Table 11. Synthesis Preserve Options

Assignment	Description	Allows Fitter Optimization?	Assignment Syntax
Preserve Registers in Synthesis	Prevents removal of registers during synthesis without restricting any optimization after synthesis, such as Hyper-Retiming or physical synthesis optimizations.	Yes	• `set_global_assignment -name PRESERVE_REGISTER_SYN_ONLY ON|OFF -entity <entity name>` `set_instance_assignment -name PRESERVE_REGISTER_SYN_ONLY ON|OFF -to <to> -entity <entity name>` • `preserve_syn_only` or `syn_preservesyn_only` (synthesis attributes)
Preserve Fan-Out Free Register Node	Prevents removal of assigned registers without fan-out during synthesis. The PRESERVE_FANOUT_FREE_NODE assignment cannot preserve a fanout-free register that has no fanout inside the Verilog HDL or VHDL module in which you define it. To preserve these fanout-free registers, implement the `noprune` pragma in the source file: `(*noprune*)reg r;` If there are multiple instances of this module, with only some instances requiring preservation of the fanout-free register, set a dummy pragma on the register in the HDL and also set the PRESERVE_FANOUT_FREE_NODE assignment. This dummy pragma	Yes	• `set_instance_assignment -name PRESERVE_FANOUT_FREE_NODE ON|OFF -to <to> -entity <entity name>` • `noprune on` (see Verilog or VHDL synthesis attribute for syntax)

continued...

Assignment	Description	Allows Fitter Optimization?	Assignment Syntax
	allows the register synthesis to implement the assignment. For example, set the following dummy pragma for a register r in Verilog HDL: `(*dummy*)reg r;`		
Preserve Registers	Prevents removal and sequential optimization of assigned registers during synthesis. Sequential netlist optimizations can eliminate redundant registers and registers with constant drivers.	No	• `set_global_assignment -name PRESERVE_REGISTER ON|OFF -entity <entity name>` `set_instance_assignment -name PRESERVE_REGISTER ON|OFF -to <to> -entity <entity name>` • `preserve`, `syn_preserve`, or `keep on` (synthesis attributes)

1.7.2.2. Preserving Signals for Monitoring and Debugging

The Compiler optimizes the RTL signals during synthesis and place-and-route. Unless preserved, the signal names in your RTL might not exist in the post-fit netlist after signal optimizations. For example, the compilation process can merge duplicate registers, or add tildes (~) to net names that fan-out from a node.

Preserving a signal so that it is available for debugging after synthesis and place-and-route involves the following steps:

- Marking the signal for preservation.

  You can mark the signal directly in your RTL code or through a QSF assignment command.
- Enabling signal preservation either for the entire project or for the instance that contains the marked signal.

  You can enable project-wide signal preservation through the Quartus Prime GUI or a QSF assignment command. To preserve signals in an instance, use a QSF assignment command.

By separating marking signals for preservation and enabling signal preservation, you can tag signals of interest as you write your code with no effect on optimization until you enable signal preservation.

When signal preservation is enabled, the nodes marked `preserve_for_debug` inherit the following attributes:

Table 12. Attributes Inherited by Nodes Marked `preserve_for_debug`

Attribute	Result
`preserve`	Prevents Quartus Prime from optimizing away or retiming a register
`keep`	Prevents Quartus Prime from minimizing or removing a signal net during combinational logic optimization
`noprune`	Prevents Quartus Prime from removing or optimizing a fanout free register
`dont_merge`	Prevents Quartus Prime from merging a register with a duplicate register
`dont_replicate`	Prevents Quartus Prime from duplicating a register

Important: Instance-level signal preservation settings always override project-wide level settings. Also, signal preservation settings in your RTL always override QSF settings.

After you synthesize your design, you can locate and review your preserved signals:

- The *Preserve for Debug Assignments* report shows the preservation status and name of all nodes that are marked for preservation.
- The *preserved for debug* filters in Node Finder helps you quickly find preserved nodes.

The overall flow to follow to preserve signals for monitoring and debugging is as follows:

1. Mark the signals that you want to preserve in one of the following ways:
   - Mark the signals directly in your RTL code with the `preserve_for_debug` attribute:

Table 13. Examples of Marking Signals with the `preserve_for_debug` Attribute in RTL

Verilog Example	VHDL Example
`(* preserve_for_debug *) wire [3:0] counter_wire;` `(* preserve_for_debug *) reg [3:0] counter_reg;` `(* preserve_for_debug *) wire [15:0] decode_out_top;` `(* preserve_for_debug *) reg [15:0] decode_out_reg_top;`	`attribute preserve_for_debug : boolean;` `attribute preserve_for_debug of counter_wire : signal is true;` `attribute preserve_for_debug of counter_reg : signal is true;` `attribute preserve_for_debug of decode_out : signal is true;` `attribute preserve_for_debug of decode_out_reg : signal is true;`

     In the Quartus Prime GUI, you can also use the **Insert Template** (**Edit ➤ Insert Template**) dialog box to add the `preserve_for_debug` attribute.
   - Mark the signals with the PRESERVE_FOR_DEBUG assignment in one of the following ways:
     - Specify the PRESERVE_FOR_DEBUG assignment for specific nodes from a command line or in your QSF file:

       ```
       set_instance_assignment -name PRESERVE_FOR_DEBUG ON -to <node hpath>
       ```

     - Specify the **Preserve signal for debug** assignment **To** any node of interest in the Assignment Editor. Select **On** as the assignment value.

     Important: In some rare cases, logic might be optimized away before the signal can be preserved when you use the PRESERVE_FOR_DEBUG assignment. If you encounter this issue, use the `preserve_for_debug` attribute in your HDL code instead.
   - Mark a system module, interface, or port for preservation in Platform Designer.

     For details, refer to "Preserving a System Module, Interface, or Port for Debugging" in *Quartus Prime Pro Edition User Guide: Platform Designer*.
2. Enable preserve for debug either project-wide or for specific instances.
   - To enable preservation and reporting project-wide, do one of the following steps:

- Enable project-wide signal preservation from the Signal Tap Logic Analyzer settings:

  Select **Assignments ➤ Settings ➤ Signal Tap Logic Analyzer ➤ Enable preserve for debug assignments**.

- Specify the global PRESERVE_FOR_DEBUG_ENABLE assignment from a command line or in your QSF file:

  ```
  set_global_assignment -name PRESERVE_FOR_DEBUG_ENABLE ON
  ```

- Specify a global **Enable preserve for debug assignments** assignment in the Assignment Editor. Select **On** as the assignment value.

Tip: When project-wide signal preservation and reporting is enabled, use the instance-level setting to exclude instances from project-wide signal preservation and reporting.

- To enable (or disable) preservation and reporting for only specific instances:
  - Specify the instance PRESERVE_FOR_DEBUG_ENABLE assignment as ON (or OFF) from a command line or in your QSF file:

    ```
    set_instance_assignment -name PRESERVE_FOR_DEBUG_ENABLE ON -to
    <instance hpath>
    ```

  - Specify the **Enable preserve for debug assignments** assignment **To** any instance of interest in the Assignment Editor. Select **On** (or **Off**) as the assignment value.
  - Specify the PRESERVE_FOR_DEBUG_ENABLE attribute with a value of ON (or OFF) on the instance in your RTL.

Table 14. Examples of Setting the PRESERVE_FOR_DEBUG_ENABLE Attribute in RTL

Verilog Example	VHDL Example
... (* altera_attribute = "-name PRESERVE_FOR_DEBUG_ENABLE ON" *) decoder decoder_inst(...	... attribute altera_attribute : string; attribute altera_attribute of my_decoder_inst : LABEL is "-name PRESERVE_FOR_DEBUG_ENABLE ON"; ... begin ... my_decoder_inst : decoder

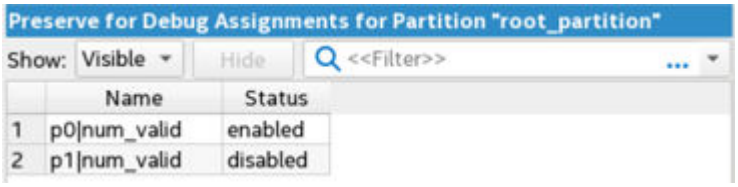
3. Synthesize your design.

   On the Compilation Dashboard, click **Analysis & Synthesis**. The Compilation Report generates when synthesis is complete. You can also now use *preserved for debug* filters in Node Finder to help you quickly find preserved nodes.

4. View the results of signal preservation.

   Open the Preserve for Debug Assignments report located in the **Synthesis ➤ Partition <*name*> ➤ Preserve for Debug** report folder.

Figure 42. Preserve for Debug Assignments Report



5. Run a full compilation to perform place and route of the design and Signal Tap instance by selecting **Processing ➤ Start Compilation**.

   The debug signals that you preserve persist through the Fitter into the finalized compilation database.

After running a full compilation, you can debug your design with any of the Quartus Prime debug tools.

Remember: After you have debugged your design, you might want to disable signal preservation (both globally and at the instance level) and run a full compilation to optimize signals that were preserved.

Table 15. Debug Signal Preservation Assignments

Assignment	Description	Example
`preserve_for_debug` (**Preserve signal for debug** in Assignments Editor)	Marks a signal to be preserved during compilation for post-synthesis or post-fit debugging when preserve-for-debug is enabled.	`set_instance_assignment -name PRESERVE_FOR_DEBUG ON -to <node hpath>`
`preserve_for_debug_enable` (**Enable preserve for debug assignments** in Assignments Editor)	Enables preserve-for-debug at either the project or instance level. When enabled, the Compiler reports the results of signal preservation in the Preserve for Debug Assignments report after compilation. The Compiler reports these nodes in the Preserve for Debug Assignments report following compilation.	`set_global_assignment -name PRESERVE_FOR_DEBUG_ENABLE ON` `set_instance_assignment -name PRESERVE_FOR_DEBUG_ENABLE ON -to <instance hpath>`

Related Information

- Intel Quartus Prime Pro Edition User Guide: Debug Tools
- Preserving Registers During Synthesis on page 60
- Using the ECO Compilation Flow chapter, User Guide: Design Optimization
- Preserving a System Module, Interface, or Port for Debugging chapter, User Guide: Platform Designer

1.7.3. Using Synopsys* Design Constraint (SDC) on RTL Files

SDC-on-RTL supports SDC files written using SDC 2.1-compliant SDC commands and can support general Tcl code that the Tcl console can parse. These SDC files target your design netlist, allowing you to target hierarchical ports.

Note:

- Only the Timing Analyzer Tcl console supports the `sdc_ext Tcl` package.
- Quartus Prime software GUI-based constraint authoring is currently disabled for files with `RTL_SDC_FILE` assignments. This means the timing constraint entry dialogs are not available. You must enter timing constraints only by typing them in.
- Issues, such as incorrect options or other syntax errors, found in the SDC-on-RTL SDC files are posted as warnings in the Quartus Prime software GUI and message console.

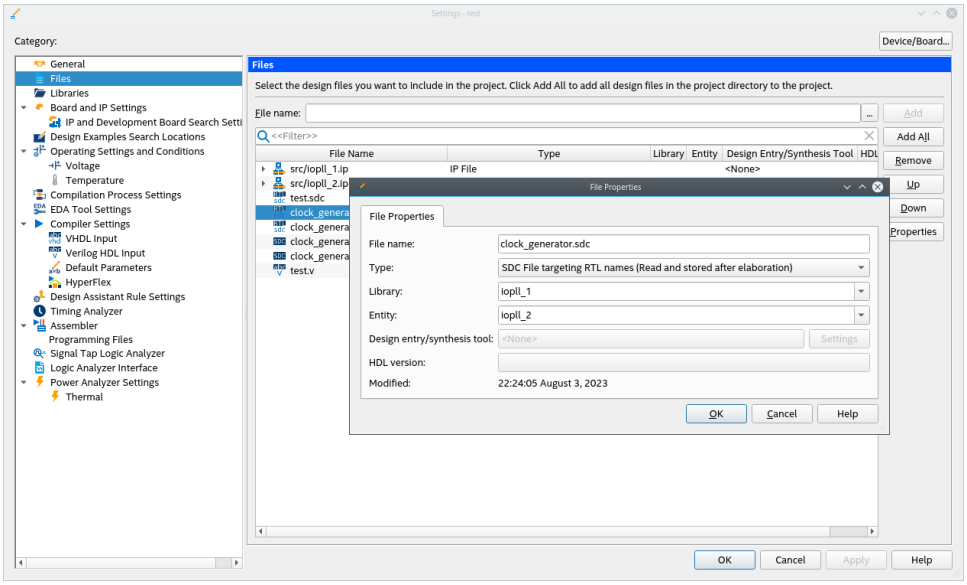
For more information about how to manage SDC-on-RTL SDC files, refer to the following topics:

Registering the SDC-on-RTL SDC File on page 65
Applying the SDC-on-RTL Constraints on page 66
Inspecting SDC-on-RTL Constraints on page 67
Creating Constraints in SDC-on-RTL SDC Files on page 72
Using Entity-Based SDC-on-RTL Constraints on page 75
Types of SDC Files Used in the Quartus Prime Software on page 78
Example: Using SDC-on-RTL Features on page 80

1.7.3.1. Registering the SDC-on-RTL SDC File

For the Post-Synthesis Static Timing Analysis (STA) on page 87, your design must include an associated SDC-on-RTL SDC file. You can use the GUI to register an SDC-on-RTL file with your current project from the **File Properties** dialog by specifying its type as **SDC File Targeting RTL names**, as shown in the following image:

Figure 43. Registering the SDC-on-RTL SDC File



Note: Alternatively, you can also associate an SDC-on-RTL file with your project from the Timing Analyzer's **Constraints ➤ Read SDC File** option.

SDC-on-RTL files are registered with an Quartus Prime software project through the following RTL_SDC_FILE assignment:

```
set_global_assignment -name RTL_SDC_FILE sdc_on_rtl_file.sdc
```

Note: Although you can use any file extension, Intel recommends using an intuitive file extension, for example, `rtlsdc`, to help distinguish SDC-on-RTL SDC files from the conventional Quartus Prime software SDC files if your design uses both.

1.7.3.2. Applying the SDC-on-RTL Constraints

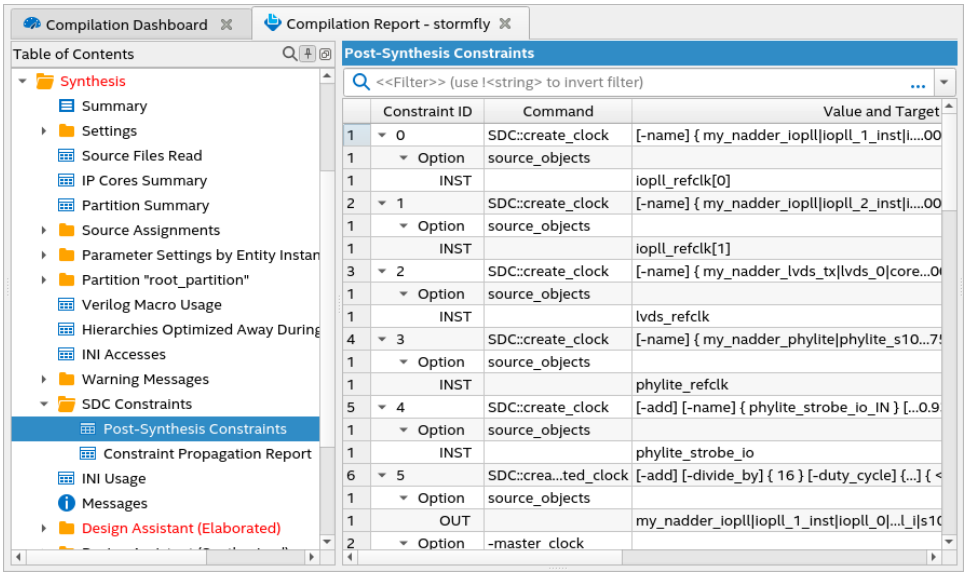
When you perform Analysis & Elaboration on your design, the SDC-on-RTL constraints are read and applied to your elaborated design. If you modify the constraints after Analysis and Elaboration, then you must rerun Analysis and Elaboration.

During the Analysis & Elaboration, `quartus_syn` reads all SDC-on-RTL SDC files and applies constraints to your design netlist. The order in which the files are listed in the QSF defines the reading order. Once this compilation stage completes, you can inspect the constraints in multiple ways. For more information, refer to Inspecting SDC-on-RTL Constraints on page 67 and Types of SDC Files Used in the Quartus Prime Software on page 78.

The constraints are stored in the internal Quartus Prime software netlist. As the compilation flow progresses, various compiler optimizations keep the constraint targets updated. This permits a write once, use anywhere methodology for the constraints.

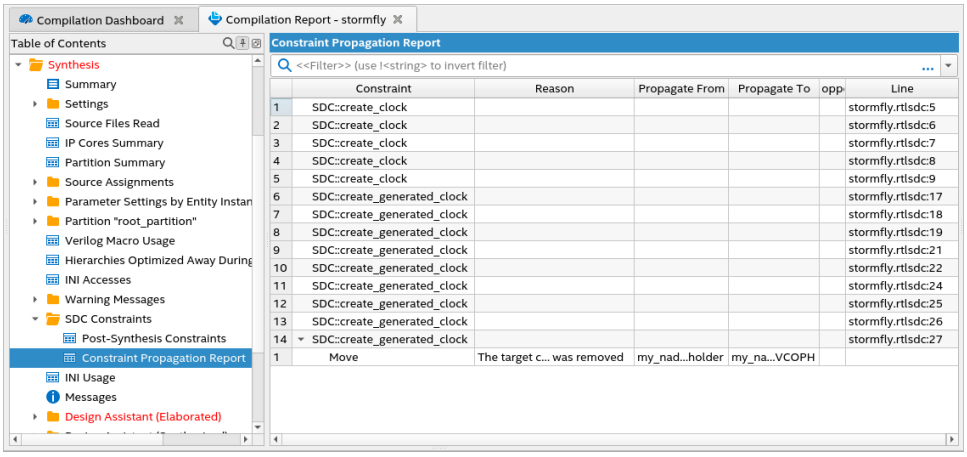
Once you are satisfied with the constraints, you can run Synthesis from the compilation dashboard. Synthesis converts the elaborated netlist into the node netlist for mapping to device resources. When Synthesis runs, the SDC constraints are processed and propagated by the Synthesis tool and you can review this in the **Post-Synthesis Constraints** report and **Constraint Propagation Report**.

Figure 44. Sample Post-Synthesis Constraints Compilation Report



The Constraint Propagation Report shows a chronological history of all changes made to each constraint during the compilation flow. You can observe an entry in the report whenever a constraint is duplicated, moved, or deleted. This report keeps getting updated with entries for modified constraints throughout the compilation flow and includes a reason when a constraint is moved. This report is beneficial for troubleshooting if the constraints changed in a way you did not expect.

Figure 45. Sample Constraint Propagation Report



1.7.3.3. Inspecting SDC-on-RTL Constraints

You can access SDC-on-RTL constraints in multiple ways in the order of the earliest opportunity in the flow, as described in the following sections:

Note: In all of the methods discussed in this section, the SDC-on-RTL constraints are read-only. You cannot modify the constraints in the compilation flow. You can only change the constraints by changing the source RTL SDC file and reloading it during Analysis and Elaboration.

Tcl Console

The Tcl console allows you to experiment conveniently with targeting constraints and the related syntax. The netlist is read-only when accessed from the Tcl console, and constraint commands are not saved into the design database.

Access the Tcl Console through the command-line interface using the following command:

```
quartus_syn -s
```

Once the Analysis & Elaboration compilation stage completes, you can load the project in the console (`project_open` *<project_name>*) and load the appropriate checkpoint using the `dni::load_design -checkpoint "constrained"` command. You can now perform tasks such as:

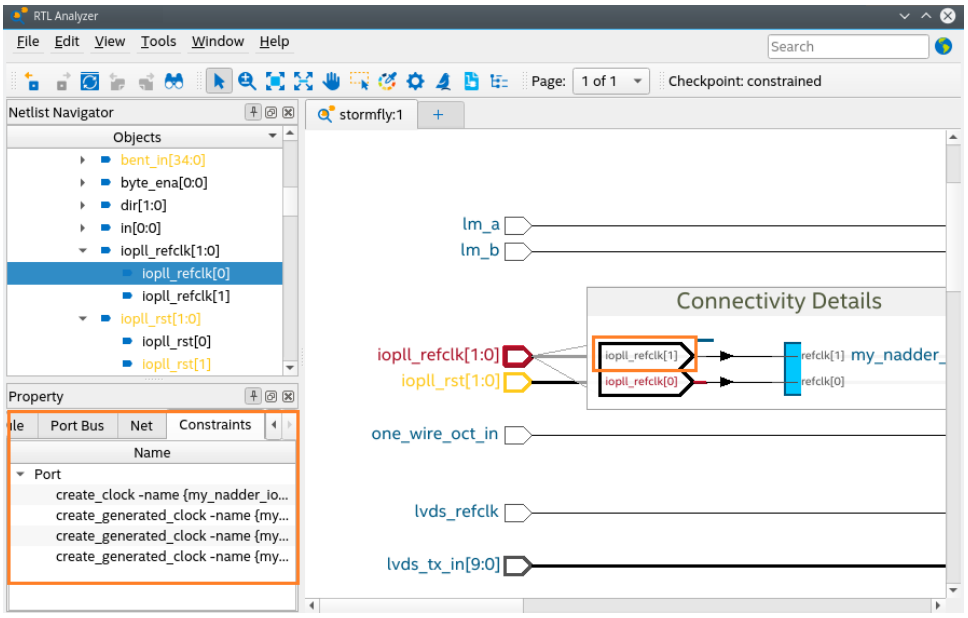
- Read a specific SDC file using the `dni::read_sdc`*<file_name>* command.
- Dump constraints using the `dni::write_sdc` command.
- Run constraint commands that are limited to the local session. For more information, refer to Creating Constraints in SDC-on-RTL SDC Files on page 72.

RTL Analyzer

To invoke the RTL Analyzer (Constrained mode) from the compilation dashboard, enable the **RTL Analysis Debug Mode** option under **Project ➤ Settings** as described in Analysis & Elaboration. The RTL Analyzer GUI allows you to view the constraints on the design netlist. When you select a netlist object in the schematic viewer or Netlist Navigator, you can view constraints targeting that object in the **Property** viewer. This helps ensure the constraints target the intended nodes in your RTL.

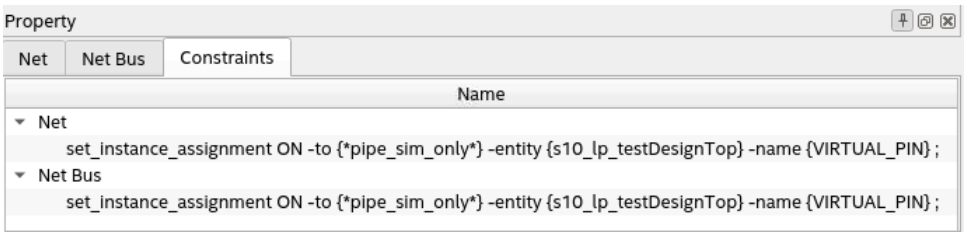
In the following example, when you select `iopll_refclk[1:0]` and right-click and select **Display Individual Bits** from its context-sensitive menu, the **Connectivity Details** pop-up window displays its ports. When you select one of the ports, constraints applied to the port are displayed in a separate **Constraints** tab, as shown in the following image:

Figure 46. Viewing Constraints in the RTL Analyzer



Besides the constraints of the selected object, the **Constraints** tab also includes constraints of the associated object. For example, a net bus' constraints are included along with the net's constraints, an instance bus' constraints with the instance's constraint, and so on.

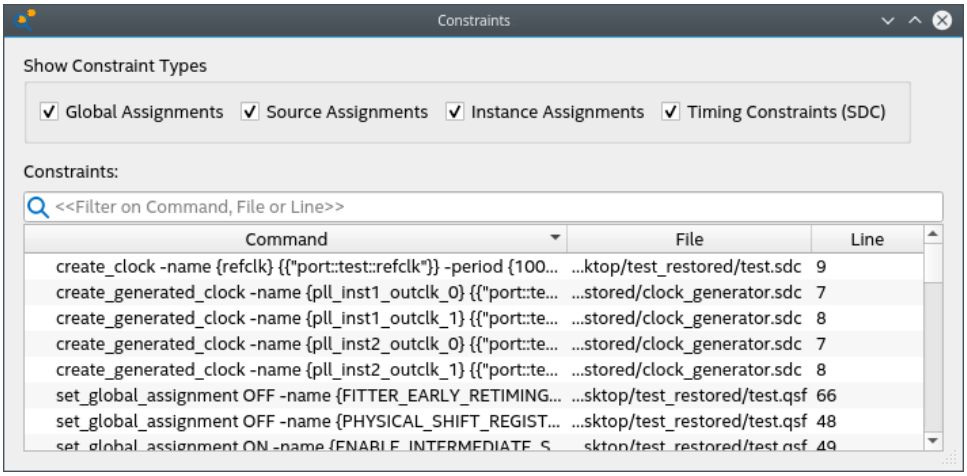
Figure 47. Viewing Constraints of Associated Objects



Note: You can cross-probe the SDC file by right-clicking on a constraint in the **Property** viewer and selecting the **View in Source** option.

You can also launch the **Constraints** dialog box (**Tools ➤ Object Constraints**) from the RTL Analyzer menu to view a list of all constraints (assignments and timing-related SDC). It allows you to select an assignment or a constraint and cross-probe to its source file by right-clicking and selecting **View in Source**. The source file that contains the assignment or constraint launches in the Quartus Prime GUI with the assignment line highlighted.

Figure 48. Object Constraints

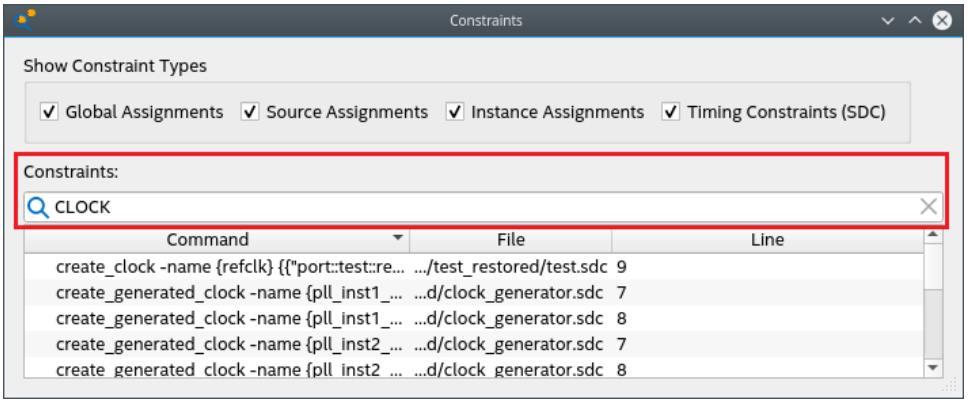


The Constraints GUI allows you to filter four types of constraints:

- **Global Assignments**: Assignments that are created with the `set_global_assignment` command.
- **Source Assignments**: Assignments embedded in source RTLs.
- **Instance Assignments**: Assignments created with the `set_instance_assignment` command.
- **Timing Constraints (SDC)**: Constraints created through SDC-on-RTL. They usually appear in the Constrained view of the Analysis & Synthesis stage. If no SDC is read in, the **Timing Constraints (SDC)** option is disabled. This can happen in views before SDC are read in, for example, in "Elaborated" and "Instrumented" views of the Analysis & Synthesis stage or when no SDC-on-RTL file is read in.

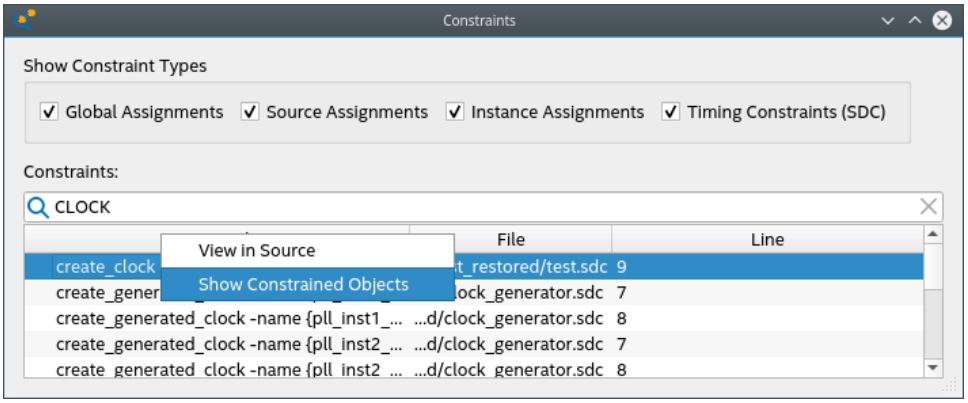
You can filter the constraints further using the filtering field to view constraints matching the string. It supports filtering through command name, file name, and line number. You can also sort the constraints using the **Command**, **File**, and **Line** column header.

Figure 49. Filtering Constraints



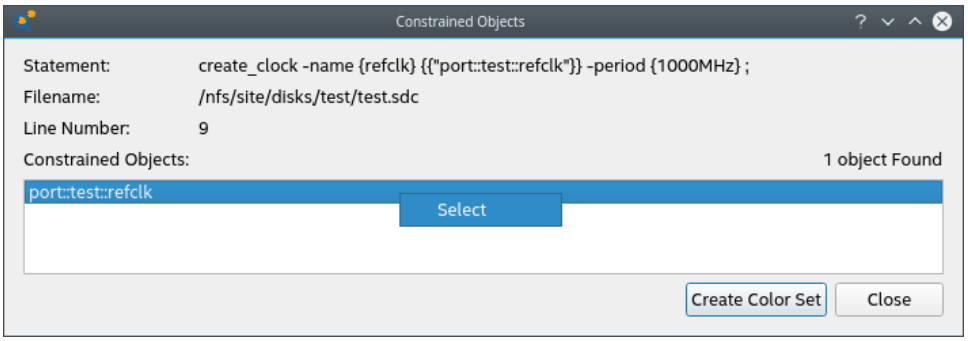
Through the context-sensitive menu of the Constraints GUI, you can view the selected constraints in its source file or the objects that it applies to.

Figure 50. Context-Sensitive Menu of the Constraints GUI



You can view objects that a constraint apply to through the **Show Constrained Objects** option, which displays the **Constrained Objects** dialog. In the **Constrained Objects**, you can select the constrained object through its context-sensitive menu. This allows you to view the object's property in the **Property Viewer**. You can also apply a color to each constrained object by clicking **Create Color Set**.

Figure 51. Constrained Objects



Quartus Prime Timing Analyzer

The Quartus Prime Timing Analyzer uses industry-standard constraint and analysis methodology to report on all data required times, data arrival times, and clock arrival times for all register-to-register, I/O, and asynchronous reset paths in your design. The Timing Analyzer verifies that the required timing relationships are met for your design to function correctly and confirms actual signal arrival times against the constraints you specify. For more information about the Timing Analyzer, refer to the *Quartus Prime Pro Edition User Guide: Timing Analyzer*.

Using the Timing Analyzer GUI or Tcl command console, you can load SDC-on-RTL constraints into the timing analysis session by running the `read_sdc` command. By default, the `read_sdc` command always loads SDC-on-RTL constraints, which happens before loading other conventional Quartus Prime software SDC files (`SDC_FILE`).

During static timing analysis, you can load only the SDC-on-RTL SDC constraints using the `import_sdc` command. This is helpful when debugging issues you suspect are caused by SDC-on-RTL constraints.

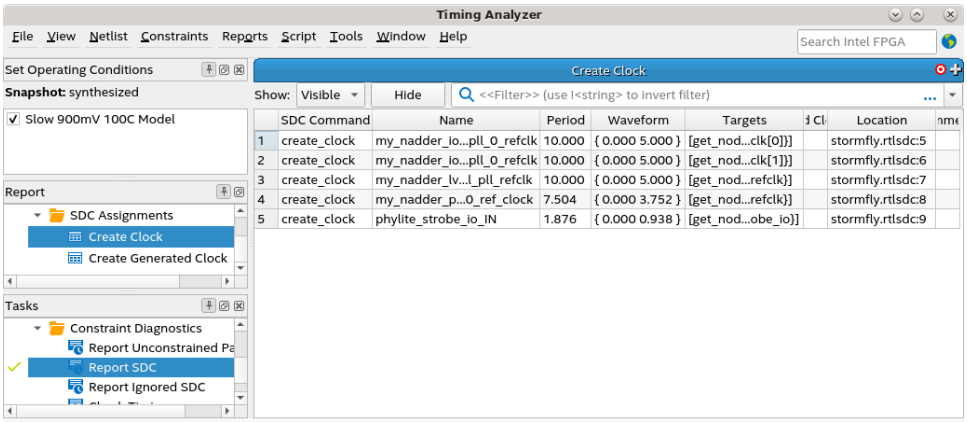
To disable the loading of SDC-on-RTL constraints during the calls to `read_sdc`, use the `read_sdc -no_import` option or set the QSF variable `ENABLE_IMPORT_SDC_DURING_READ_SDC` to `OFF`.

Note: If you change the actual SDC-on-RTL files and import them using `read_sdc` or `import_sdc` command without rerunning Analysis & Elaboration, the Timing Analyzer issues a warning message informing you that changes to the SDC-on-RTL files will not be observed until Analysis & Elaboration is rerun.

Once you import the constraints using the `read_sdc` or `import_sdc` command, they become standard constraints in the Timing Analyzer. Standard constraint diagnostic reports (`report_exceptions`, `report_sdc`, and so on) operate on these constraints, and you can update the constraints for the current Timing Analyzer session using the existing Quartus Prime timing analysis API commands.

Note: You can cross-probe the constraints in the `report_sdc` by right-clicking on the Location column of a constraint and selecting **Locate in Constraint File**.

Figure 52. SDC Report in the Timing Analyzer



1.7.3.4. Creating Constraints in SDC-on-RTL SDC Files

SDC 2.1 Compliance

SDC-on-RTL SDCs is an industry-standard to provide constraints targeting your RTL. Target nodes are from the elaborated netlist and aligned closely with your RTL. You can target hierarchical nodes and propagate constraints into the compilation flow through various compilation stages.

Note: The `read_sdc` command loads SDC-on-RTL SDC files into the Timing Analyzer. Depending on the snapshot you loaded, `read_sdc` attempts to read other SDCs suitable for that snapshot. For additional information about the types of SDC files used for different use cases, refer to Types of SDC Files Used in the Quartus Prime Software.

The Quartus Prime software preserves the order of constraints in SDC-on-RTL SDC files. The order in which you list constraints in the SDC file defines the order in which they are loaded in the Quartus Prime software.

SDC-on-RTL constraints are designed to support only SDC 2.1-compliant commands. Object accessors, such as `get_keepers`, `get_registers`, or `get_nodes`, are notably absent. The following is the list of supported SDC commands, and you can find more information about them in the TCL Commands and Packages Summary section of the *Quartus Prime Pro Edition User Guide: Scripting*:

- `create_clock`
- `create_generated_clock`
- `set_input_delay`
- `set_output_delay`
- `set_false_path`
- `set_max_delay`
- `set_min_delay`
- `set_multicycle_path`
- `set_clock_groups`
- `set_clock_latency`
- `set_clock_uncertainty`
- `set_data_delay`
- `set_net_delay`
- `set_max_skew`

Note: Not all of these commands support the full selection of SDC 2.1 arguments because the Timing Analyzer does not support all arguments. If you specify an unsupported argument, these commands issue a warning message and ignore the argument. If you prefer a Tcl error when you encounter a warning in the Timing Analyzer, use the following:

```
set_global_assignment -name ERROR_ON_WARNINGS_LOADING_SDC_ON_RTL_CONSTRAINTS ON
```

Identifying Timing Paths

For defining timing paths, use SDC standard accessors, such as `get_pins`, instead of the Quartus Prime software-specific commands, such as `get_keepers` or `get_registers`. Intel recommends using explicit `get_<pins|ports|nets>` commands when targeting objects. Using bare names can result in unintended targets, and the database issues a warning.

Quartus Prime Conventional SDC (targets Quartus Prime timing graph):

```
set_false_path -from [get_keepers { reg_A }] -to [get_keepers { reg_B }]
```

SDC-on-RTL SDC (targets your netlist nodes):

```
set_false_path -from [get_pins { reg_A|clk }] -to [get_pins { reg_B|d }]
```

Note: The path must start at the register `clk` pin to identify the timing path. If your design uses `reg_A|d` or `reg_A|q`, the timing path in the Timing Analyzer is invalid, as shown in the Report Exception report:

Status	Exception Command	From Flag	From	To Flag	To	Setup Slack
Complete	set_false_path	-from	[get_pins{tf_sync[0]|ff_src|clk}]	-to	[get_pins{tf_sync[0]|ff_dst|d}]	3.603
Invalid	set_false_path	-from	[get_pins{tf_sync[2]|ff_src|q}]	-to	[get_pins{tf_sync[2]|ff_dst|d}]	Invalid
Invalid	set_false_path	-from	[get_pins{tf_sync[3]|ff_src|d}]	-to	[get_pins{tf_sync[3]|ff_dst|d}]	Invalid

Note: Netlist targets are case-sensitive. Sometimes, the lower-level module ports are capitalized. In this case, using the `-nocase` option with the `get_<pins|ports|nets>` commands ignore the case sensitivity.

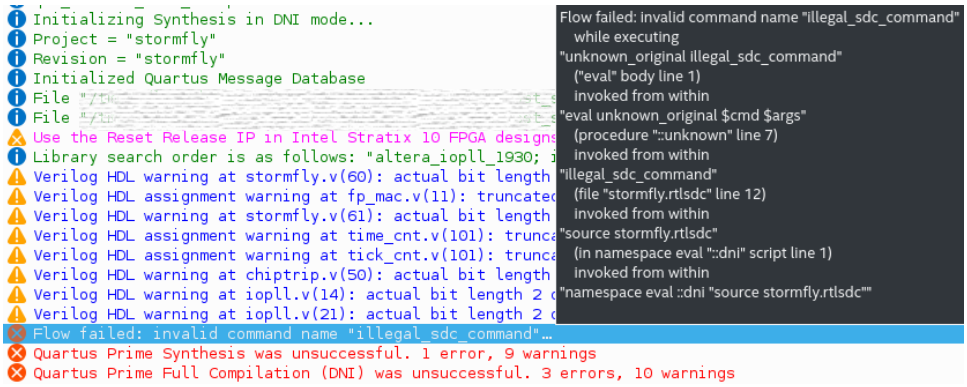
Debugging SDC-on-RTL Constraints

There are several ways to debug SDC-on-RTL constraints. For example, you can isolate the exact locations where constraints are used in your design and analyze them in the flow. You can also use compilation reports that report the constraints (see Elaboration Constraints Compilation Report). However, when debugging the constraints, consider the following:

Explicit Errors When Loading Constraints

Errors with reading SDC-on-RTL constraints issue an error message with a command stack showing the line number of the offending command. For example:

Figure 53. Error Message with a Command Stack



You can debug erroneous constraints directly through the Tcl console. You can iterate on issues with initial target resolution or command syntax at this early stage easily, as this is the stage where Quartus Prime software loads the SDC-on-RTL SDC files.

Note: In general, it is not possible to directly load SDC-on-RTL constraints in the Timing Analyzer because of the following reasons:

- The target netlist is different in the elaborated netlist and not the timing netlist. Although there may be some common nodes between the two netlists, both netlists are different.
- Compliant SDC 2.1 syntax used in SDC-on-RTL can be different from the conventional Quartus SDC commands.

Despite these caveats, if you still want to attempt to load any SDC file in the Timing Analyzer, use `read_sdc <sdc filename>`.

Constraint Behavior and Interpretation Issues

When handling the constraints, ensure the following:

- Targets and basic syntax are correct (node finding or target acquisition). You can use the RTL Analyzer to find the nodes on the elaborated netlist.
- Constraints make sense with respect to the timing graph (constraint behavior). You can review this in the Timing Analyzer.

  Examine the SDC-on-RTL constraints conveyed to the Timing Analyzer (`import_sdc` loads only the SDC-on-RTL commands) from the design netlist. The `import_sdc` command loads a version of the SDC-on-RTL constraints after propagating through various stages of the compile flow. After this step, the experience is the same as a typical Timing Analyzer session.

Note: Although Intel does not recommend loading raw SDC-on-RTL SDC files from the Timing Analyzer, you can issue Timing Analyzer-compatible SDC commands using the Timing Analyzer Tcl console to debug your design. By issuing SDC commands directly, you can override SDC-on-RTL constraints and explore the effect of changing the values. However, the changes are valid only for that timing analysis session. For permanent changes, you must update the source SDC-on-RTL SDC file and rerun Analysis and Elaboration.

1.7.3.5. Using Entity-Based SDC-on-RTL Constraints

A typical design includes a combination of third-party IPs and RTL, so the primary goal of entity-based SDC-on-RTL is to ensure seamless integration of IPs by empowering IP owners to encapsulate their IP's SDC constraints.

Given that timing constraints specified in an SDC file are generally applied globally throughout a design rather than to specific entities, proper encapsulation of IP SDCs becomes crucial. This encapsulation allows utilizing the IPs without encountering unexpected SDC leaks. To achieve this, the entity binding prepends the RTL path name of the IPs, effectively preventing any SDC leaks and averting the potential impact on design paths that might coincidentally match the name.

In addition, the introduction of entity based SDC-on-RTL constraints offer IP authors the ability to define specific constraints at the module boundaries. These constraints are optimized for post-synthesis timing analysis within the context IP instantiation in the design hierarchy. Even when IP authors lack precise knowledge about where their IPs are instantiated in the design hierarchy during its implementation, the constraints remain effective. This approach allows IP authors to implement their SDC constraints without requiring detailed information about the eventual placement of the IP within the hierarchy.

The flow supports reading entity-based SDC-on-RTL in designs and IP cores during the Analysis & Elaboration stage, where the entity-based SDC-on-RTL constraints are read and saved in a low-level entity database. These constraints are eventually processed in the SDC read-in order and applied to the hierarchical netlist objects during compilation.

The Quartus Prime runtime generates IP SDCs along with the IP instantiation and specifies a project assignment to associate IP SDCs with the IP design entity name.

QSF Assignment Syntax

```
set_instance_assignment -name RTL_SDC_FILE <sdc_file_name> -entity <entity_name>
[-no_sdc_promotion]
```

Where:

Argument	Description
`RTL_SDC_FILE`	Specifies the SDC-on-RTL file name.
`-entity`	Indicates that it is an entity-based assignment. The SDC file is applied to each instance of the design entity. The instance hierarchy path is implicitly applied to the `pattern` argument search for `dni::get_*` (`get_cells`, `get_pins`, `get_ports`, and `get_nets`) commands.
`[-no_sdc_promotion]`	An optional argument that works only with the `-entity` flag. For the entity-based constraints, the `[-no_sdc_promotion]` argument removes the default behavior of the instance hierarchy path being the default implicit with the netlist search commands, This

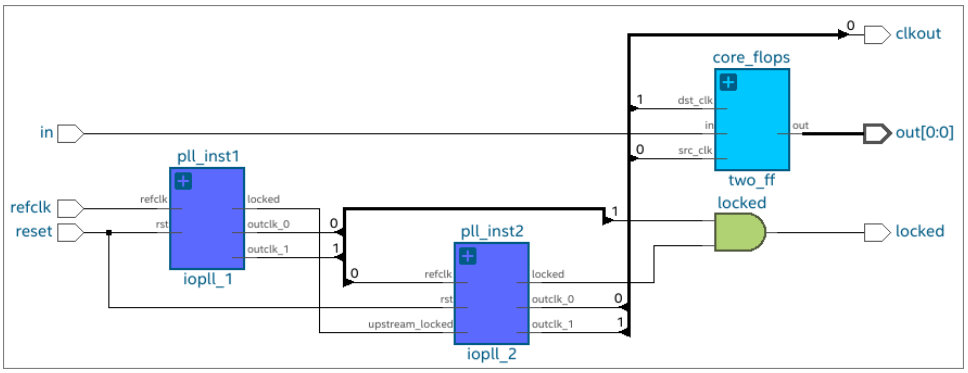
continued...

Argument	Description
	allows you to control which individual command to apply the instance hierarchy path in the netlist object search. Use the `get_entity_current_instance` Tcl command to obtain the current instance hierarchy path of the entity. For example: `set_false_path -from [get_pins [get_entity_current_instance]|ff_src|clk] \` `-to [get_pins [get_entity_current_instance]|ff_dst|d]]`

Example 6. Entity-based SDC-on-RTL Constraints Example

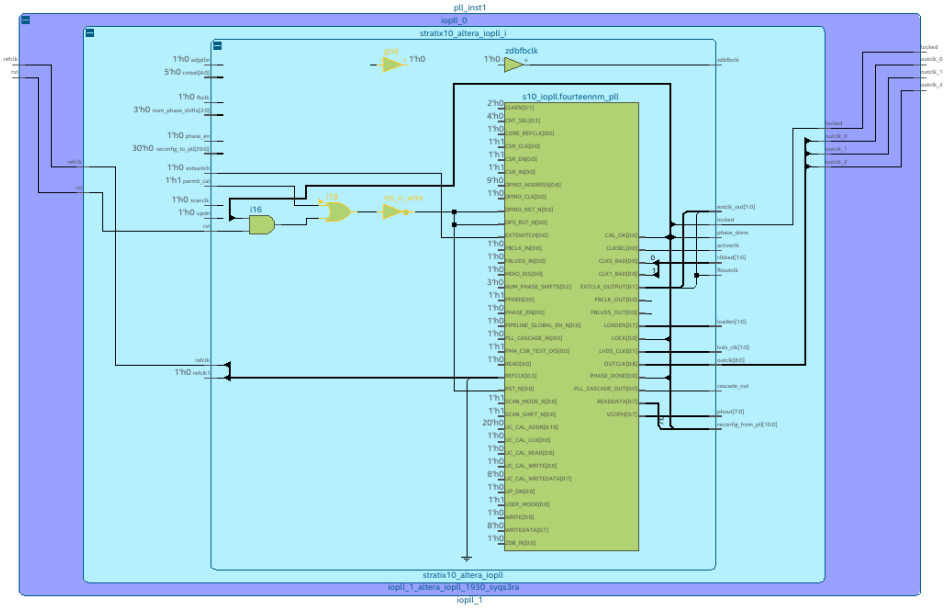
Furthermore, these entity-based SDC-on-RTL constraints provide flexibility. You can enhance or override them by introducing additional SDC constraints during the implementation stages. The following example shows how to establish logical constraints for post-synthesis timing analysis using entity-based SDC constraints on a design that includes two instances of IOPLLs:

Figure 54. Example Design with Two Instances of IOPLLs



In this simple design example, observe `iopll_1` and `iopll_2` entities. There are additional hierarchies inside these entities. The following image depicts the hierarchy for `iopll_1`:

Figure 55. Hierarchy Inside the `iopll_1` Block



1. Apply the initial SDC-on-RTL constraints file to the entire design using the RTL_SDC_FILE argument.

```
set_global_assignment -name RTL_SDC_FILE test.sdc
```

2. Define a specific SDC-on-RTL constraint file using the RTL_SDC_FILE complemented by the arguments `-entity` and `-library` to target each IOPLL entity.

```
set_global_assignment -name RTL_SDC_FILE clock_generator.sdc  -entity
"iopll_1" -library "iopll_1"
set_global_assignment -name RTL_SDC_FILE clock_generator.sdc  -entity
"iopll_2" -library "iopll_2"
```

3. Define the constraints that rule over the IOPLL module.

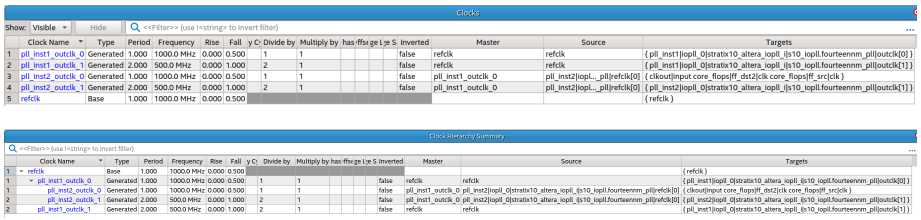
   Note: During the initial stages of designing when the RTL netlist is generated, the internal connections of the PLL might not be fully known. Consequently, apply these constraints at the module boundaries, specifically at the IOPLL ports using the `get_ports` (or `get_pins` *<port_name>*) Tcl command when addressing the module inputs and outputs. Along with other constraints, they facilitate the creation and propagation of output clocks generated from the incoming reference clock. The actual connections in the RTL netlist between the clock source and the target are not required to exist initially to apply early timing constraints. However, the Timing Analyzer will issue a warning about the missing connectivity and continue to operate.

```
# clock_generator.sdc
# get the name of the current instance to generate a name
set current_entity_instance [get_entity_current_instance]
```

```
# Note: Do not specify the -master_clock if you want STA to
#       use the source to determine the master clock name.
create_generated_clock -divide_by 1 -source [get_ports refclk] [get_ports
outclk_0] -name ${current_entity_instance}_outclk_0
create_generated_clock -divide_by 2 -source [get_ports refclk] [get_ports
outclk_1] -name ${current_entity_instance}_outclk_1
```

The clocks are shown here in the Timing Analyzer report, as shown in the following:

Figure 56. Post-Synthesis Clock Reports



It is advantageous to leverage SDC-on-RTL scoping capabilities to generate timing constraints that target RTL nodes, which can be applied during the early stages of timing analysis. Even so, the constraints defined through entity-based SDC-on-RTL files can be adjusted, replaced or supplemented, during the implementation stage, where physical constraints come into play, this allows you to target internal nodes as needed for accurate timing modeling.

Tcl Commands

The SDC commands set model closely relates to the industry-standard SDCs. In particular, the `dni::get_ports` command can query hierarchical ports on an unflattened design netlist. This behavior differs from the existing Timing Analyzer `get_ports` command that can query top-level ports in the post-synthesis design.

When used in the context of entity-based SDC-on-RTL file, the `dni::get_ports` command, for example, `[dni::get_ports refclk]` searches for a port in the context of the current instance of the instantiated RTL. To avoid the implicit prefix of the current instance in the `dni::get_ports` command, use the `-no_sdc_promotion` option along with the `dni::get_entity_current_instance` command in the SDC file where the entity's instance hierarchy path is required.

Related Information

Example: Using SDC-on-RTL Features on page 80

1.7.3.6. Types of SDC Files Used in the Quartus Prime Software

This section provides a high-level summary of the differences between various SDC file types available for use.

The Quartus Prime software uses conventional SDC files to target the timing netlist. These SDCs are not applied until the Fitter plan stage completes. However, for Post-Synthesis Static Timing Analysis (STA) on page 87, you can introduce SDC files to the Quartus Prime software in one of the following ways:

- SDC-on-RTL
- Synthesis SDC

The following table attempts to summarize the differences between the various SDC file types:

Table 16. Types of SDC Files Used in the Quartus Prime Software

	SDC-on-RTL	Synthesis SDC	SDC (Conventional)
Stage where constraints are read	Analysis & Elaboration	Synthesis	Fitter, Signoff
Stage where constraints are processed	Synthesis through Fitter	Synthesis only	Fitter, Signoff
QSF assignment	`RTL_SDC_FILE` (supports entities)	`SDC_FILE SDC_ENTITY_FILE-read_during_post_syn_and_post_fit_timing_analysis` `SDC_FILE SDC_ENTITY_FILE-read_during_post_syn_and_not_post_fit_timing_analysis`	`SDC_FILE`
Syntax supported	Tcl with SDC 2.1 commands	Tcl with Quartus Prime SDC commands	Tcl with Quartus Prime SDC commands
SDC 2.1-compliant	Yes	No	No
Target type	RTL	Quartus Prime timing graph	Quartus Prime timing graph
Hierarchical targets	Yes	No	No
Buried timing nodes (used by IP)	No	Core fabric only. *Note:* Such nodes do not exist for the periphery in post-synthesis STA.	Yes
STA command to load constraints	Executes the `read_sdc` or `import_sdc` command in any snapshot.	Executes the `read_sdc` command only during static timing analysis on the synthesized snapshot.	Executes the `read_sdc` command during static timing analysis on any fitter snapshot (plan, place, route, retime). *Note:* Not loaded during synthesis.

As shown in the table, the assignments allow different SDCs to be used during post-synthesis STA and post-fit STA. It is also possible for an SDC to check the current snapshot through `is_post_syn_sta` to determine the appropriate commands to use, as shown in the following example:

```
if {[is_post_syn_sta]} {
    puts "In post syn sta!"
}
```

1.7.3.7. Example: Using SDC-on-RTL Features

This topic provides some examples of how to use SDC-on-RTL features and set up timing constraints in your design.

You can work with both new and pre-existing designs that incorporate components from diverse sources, including Verilog, VHDL, IPs, and Platform Designer. Suppose your designs contain nodes that require constraint definitions, such as clocks, generated clocks, or asynchronous paths, across various hierarchy levels. In that case, you can efficiently target them using SDC-on-RTL constraints. It is imperative that your chosen design successfully passes the Analysis & Elaboration stages within the compilation flow for seamless and effective testing process.

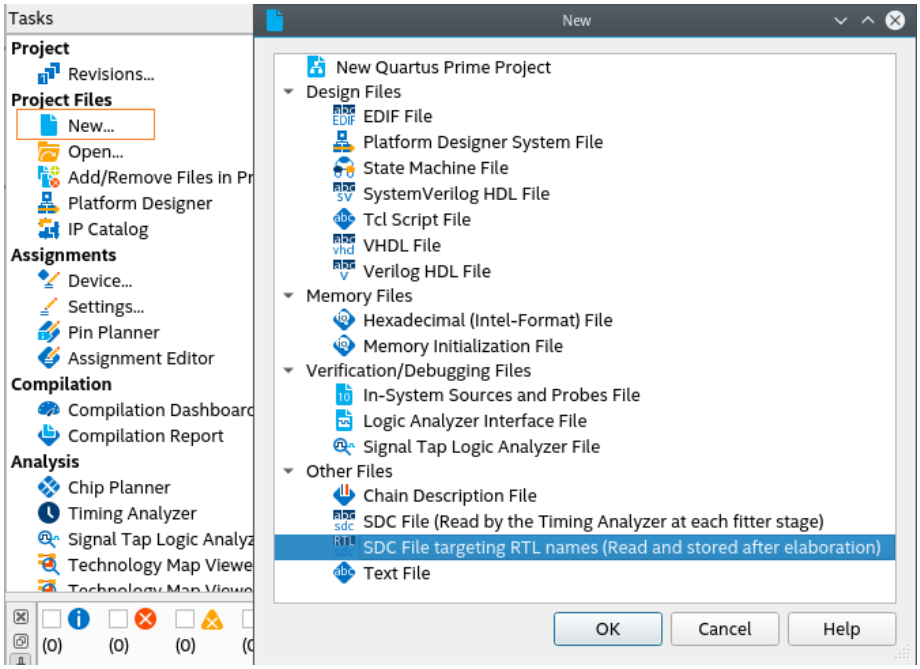
You can control the number of stages generated during the Analysis & Elaboration using the **RTL Analysis Debug Mode** option under **Project ➤ Settings**. This mode is off by default, which means only Elaborated and Swept checkpoints are available, and Instrumented and Constrained checkpoints are unavailable. When you enable this mode, all four checkpoints become available.

Note: Currently, Quartus Prime IPs use conventional SDCs only, which means if you want to create SDC-on-RTL constraints that interact with IP clocks, define them initially with SDC-on-RTL constraints. These clocks can eventually be overwritten by the IP SDCs. Alternatively, if you do not need the clocks defined at Elaboration, use `derive_clocks` during Post-Synthesis Static Timing Analysis (STA) on page 87 to automatically generate clocks temporarily for the Timing Analyzer session.

The following steps demonstrate the process of setting up timing constraints for your designs by targeting RTL node names:

1. Click **New** in the left-hand **Tasks** pane. The New dialog appears.
2. Click **SDC File targeting RTL Names (Read and stored after elaboration)**.

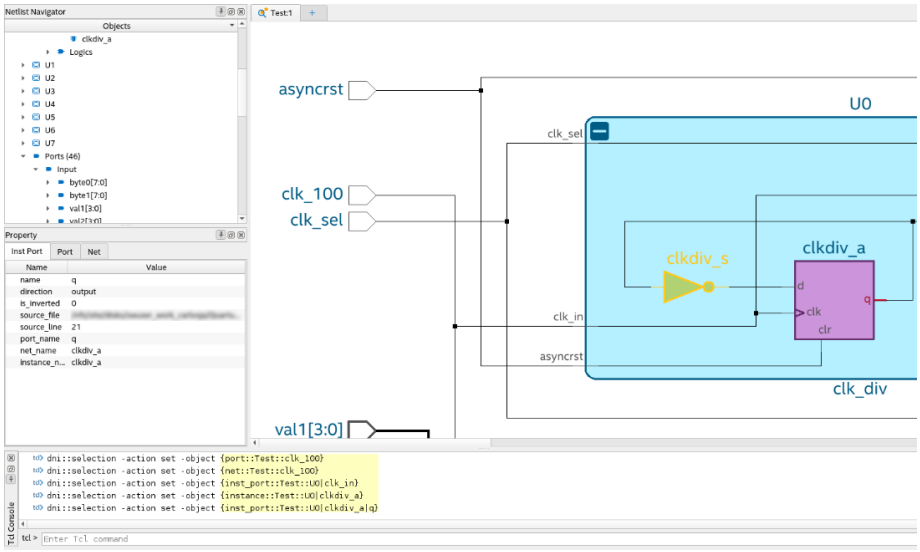
Figure 57. New Dialog Box



3. Click **OK**.
4. Within the new SDC-on-RTL file, formulate a comprehensive set of constraints targeting nodes by their RTL names. For information about how to obtain a list of the supported commands, refer to Creating Constraints in SDC-on-RTL SDC Files on page 72.
5. Ensure the constraint targets are correct and the selected design passes the Analysis & Elaboration compilation stage as the constraints are read during this stage and applied to the elaborated netlist. This step is pivotal as it empowers the software to generate a database of your design. Additionally, it grants access to various checkpoints from where you can access the RTL Analyzer.

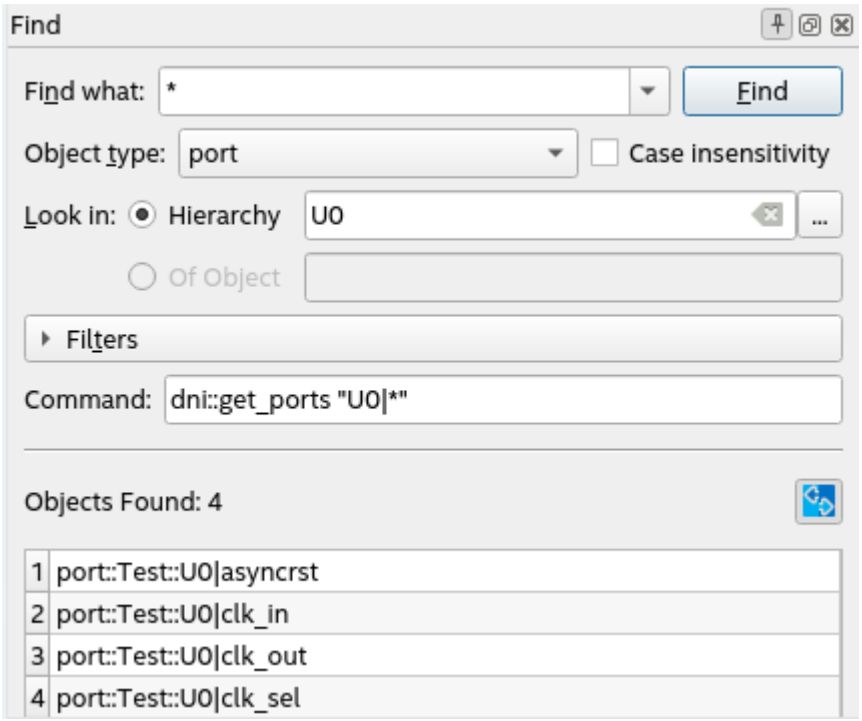
   Note: The RTL Analyzer assists you in locating specific netlist nodes within your design that require constraint application. It allows you to navigate your design and select your desired target nodes. Subsequently, you can use corresponding Tcl commands generated within the Tcl console to extract hierarchical node names from the Tcl commands. This process streamlines the task of implementing constraints on the netlist nodes. For more information about the RTL Analyzer, refer to Exploring the RTL Analyzer on page 23.

Figure 58. Locating Specific Netlist Nodes in the RTL Analyzer



Alternatively, you can locate nodes within your target netlist using the **Find** tool, which is available from the **Edit** menu in the RTL Analyzer. The **Find** tool allows you to search objects within the updated database from the checkpoint accessed.

Figure 59. Find Dialog in the RTL Analyzer



In addition, you can create your collections by focusing on the inputs and outputs of your design through the use of the `get_ports` command. Within your design, you can locate specific nets using the `get_nets` command or target pins using the `get_pins` command.

After establishing your timing constraints in your designs, use the following examples as a guide to set up your constraints, specifically focusing on RTL node names, thereby ensuring precision and efficiency throughout your design process.

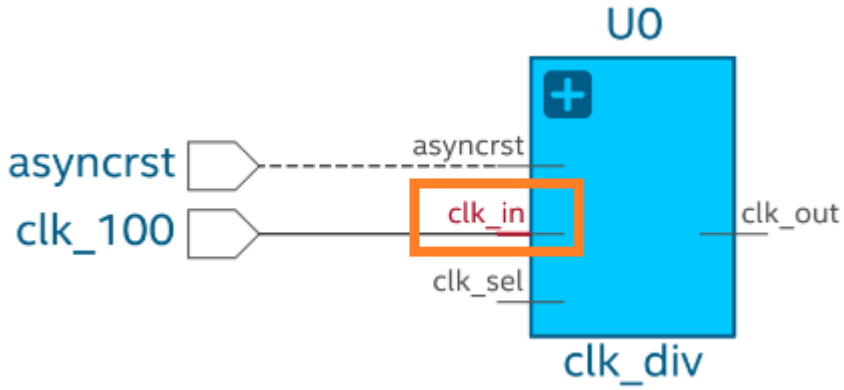
Example 7. Targeting Pins of Top-level Instances

In certain scenarios, it is necessary to constrain pins of top-level instances. For example, when defining clocks and reset inputs, you can utilize the `get_pins` Tcl command followed by the hierarchical pin name to filter each pin as shown in the following:

```
get_pins U0|clk_in
```

This Tcl command returns the input pin `clk_in` in the instance `U0`.

Figure 60. Targeting Pins of Top-level Instances



Similarly, you can target other pins of the same instance and apply necessary constraints as shown in the following:

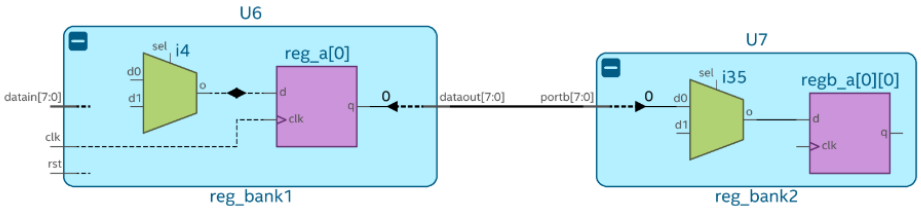
```
create_clock -name clk_input -period 10 [get_pins U0|clk_in]
create_generated_clock -name clk_output -source [get_pins U0|clk_in] -divide_by
2 [get_pins U0|clk_out]
```

Example 8. Targeting Pins in Cells

When constraining your design, you might need to target cell pins to apply constraints. In such scenarios, you can use the `get_pins` Tcl command and RTL names to locate specific cell pins, such as `clk` or `d` inputs and `q` outputs. For example:

```
get_pins U6|reg_a[0]|clk
get_pins U6|reg_a[0]|d
get_pins U6|reg_a[0]|q
```

Figure 61. Targeting Pins in Cells Example



These Tcl commands return specific pins of the `reg_a[0]` register in the U6 instance. You can use the returned collection to constrain paths. For example, from `U6|reg_a[0]` to `U7|regb_a[0][7]` as follows:

```
set_false_path -from [get_pins U6|reg_a[0]|clk] -to [get_pins U7|regb_a[0][0]|d]
```

Example 9. Targeting Pins Using Wildcards

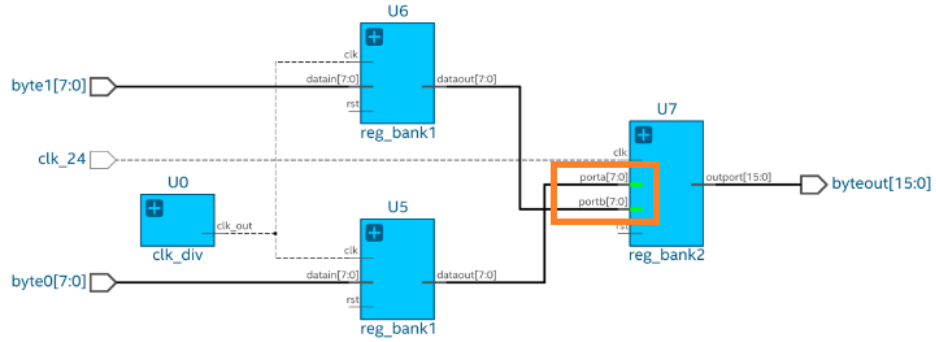
Within your design, you might need to constrain several pins with similar properties and names. In this case, use wildcards to filter the results.

Note: Use the wildcards judiciously and ensure the scope does not include more targets than necessary. Targeting extraneous nodes can limit optimizations and increase compilation runtime and memory.

The `get_pins` command can target buses `porta` or `portb` of `U7`, as shown in the following:

```
get_pins U7|porta[*]
get_pins U7|portb[*]
```

Figure 62. Targeting Pins Using Wildcards Example



The Tcl command returns a collection of pins from `porta[0]` to `porta[7]` and from `portb[0]` to `portb[7]`. You can use the returned collection to constrain all objects simultaneously, as shown in the following:

```
set_false_path -from [get_pins U5|rega*|clk] -to [get_pins U7|porta[*]]
set_false_path -from [get_pins U6|rega*|clk] -to [get_pins U7|portb[*]]
```

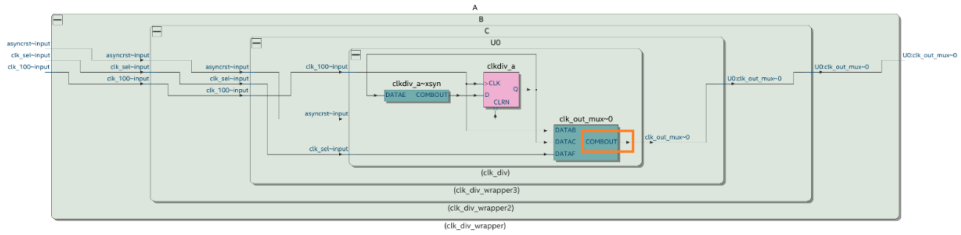
Example 10. Example 4. Applying Constraints at Deeper Hierarchies

Within your design, you can apply constraints at different levels of hierarchy using the pipe character (|) to separate hierarchy levels of instances. Constraints are applied when the hierarchy levels match and the string values, including wildcards, match the pin names. For example:

```
get_pins U3|U0_lv1|U0_lv2|U0_lv3|flag
```

The fundamental timing analysis flow requires executing the fitter to elaborate the timing netlist before applying any constraints. In the following example, suppose you intend to constraint a generated clock buried deep within module A:

Figure 63. Applying Constraints at Deeper Hierarchies Example

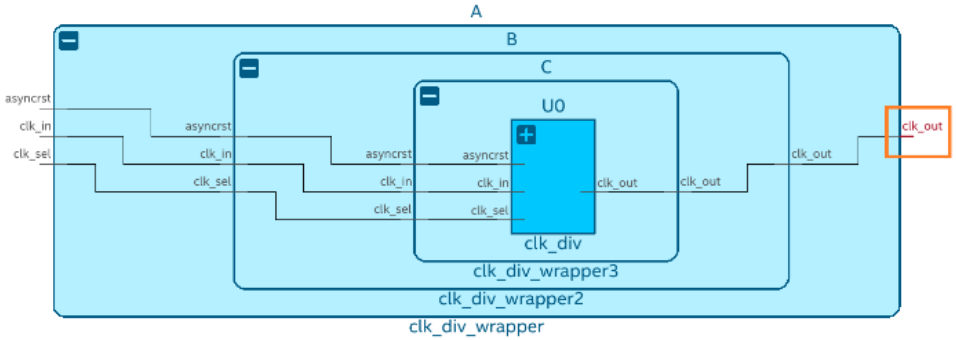


To achieve this, locate the cell `clk_out_mux` where the constraint must be applied and identify the pin name `COMBOUT`. This process often results in a complex path name that can be challenging to decipher, especially when module names are intricate (unlike straightforward names, such as `A|B|C`). Additionally, suppose the hierarchy evolves in the future. In that case, you must rerun the fitter and delve into the design again to derive the updated path, which can lead to inconsistencies between the design and the constraint targets. You can target the `COMBOUT` pin as follows:

```
get_pins A|B|C|U0|clk_out_mux~0|combout
```

SDC-on-RTL also offers an efficient means of constraint propagation, enabling you to apply constraints at module boundaries. It ensures that these constraints seamlessly extend to the corresponding leaf instances during the synthesis stage. For instance, revisit the previous example, in which the clock constraint embedded within module A can be established using SDC-on-RTL constraints at the module A's boundaries, specifically focusing on the `clk_out` pin.

Figure 64. Deeper Design Hierarchies



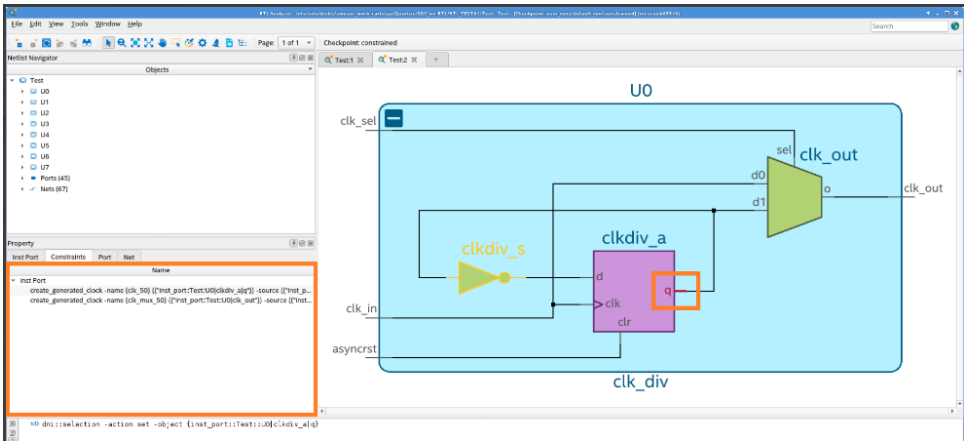
To target the `clk_out` pin of module A, use the following filter:

```
get_pins A|clk_out
```

By directing your attention towards pins located at the boundaries of abstract blocks, you gain the flexibility to modify the internal instance hierarchy as needed. Constraints remain effective even if you decide to rename an internal instance within your module, for instance, changing it from `A|B|C|U0` to `A|X|Y|U0`. Importantly, this can be accomplished without requiring alterations to your existing constraints. This demonstrates the robust capabilities of SDC-on-RTL, allowing you to concentrate on boundary pins rather than navigating complex hierarchies. This approach ensures constraint accuracy and simplifies constraint management.

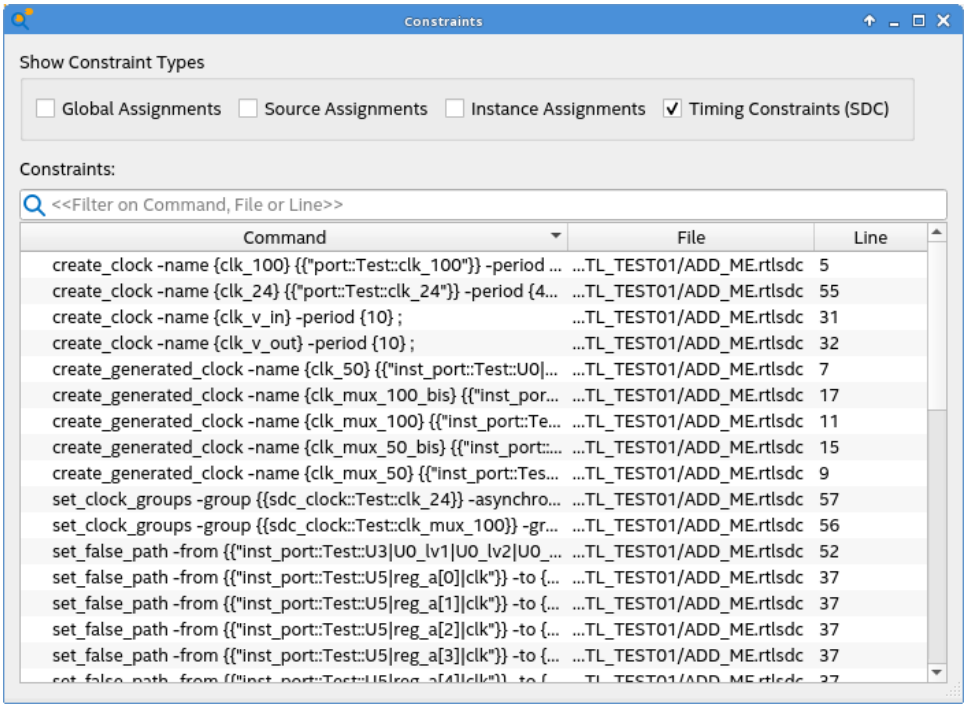
After creating the constraints, rerun the Analysis & Elaboration on the compilation dashboard so that constraints are read and applied to your design. Open the constrained checkpoint of the RTL Analyzer and carefully select the nodes where the constraints were applied. Utilize the Property Viewer to verify the correct application of constraints to these nodes. For additional information, refer to Inspecting SDC-on-RTL Constraints on page 67.

Figure 65. Property Viewer in the RTL Analyzer



Utilize the Quartus Prime software's additional tools to explore and confirm that all SDC constraints were read and successfully applied. Tools like the Constraints viewer launched from the RTL Analyzer can assist you in verifying and cross-probing the constraints with the Schematic Viewer.

Figure 66. Constraints Viewer



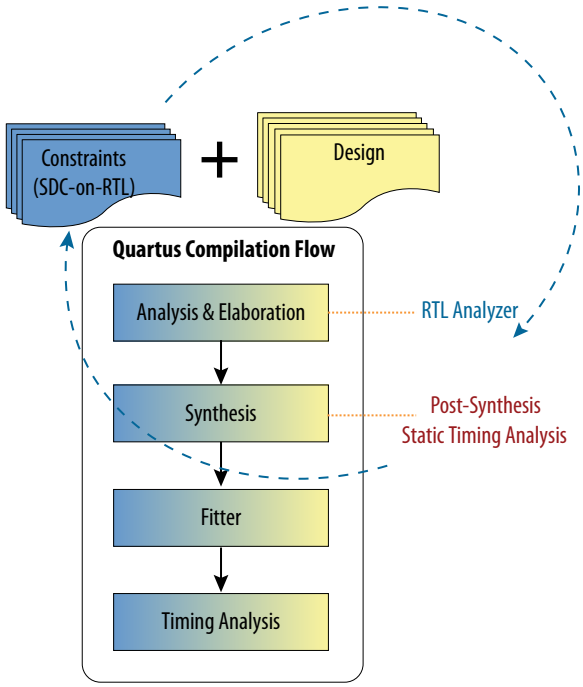
Additionally, the reports under **Compilation Report ➤ SDC Constraints** folder provide a detailed view of the constraints and their locations. Use the Constraint Propagation Report to view how constraints are propagated as the netlist is transformed and inspect where the constraints end up post optimizations.

Important: These tools offer valuable means to verify the accurate application of your constraints. However, if you intend to iterate on the process of defining constraints using the SDC-on-RTL approach, you must rerun the Analysis & Elaboration stage each time. This iterative approach ensures that the constraints are meticulously analyzed during netlist generation, resulting in enhanced performance and seamless integration with your design.

1.7.4. Post-Synthesis Static Timing Analysis (STA)

Post-synthesis static timing analysis (STA) allows you to run the Timing Analyzer directly after synthesis. This flow involves running Analysis & Elaboration and Synthesis stages and iterating on your design's static timing analysis results early in the Quartus Prime software compilation flow without running the Fitter.

Figure 67. Early Timing Analysis Flow



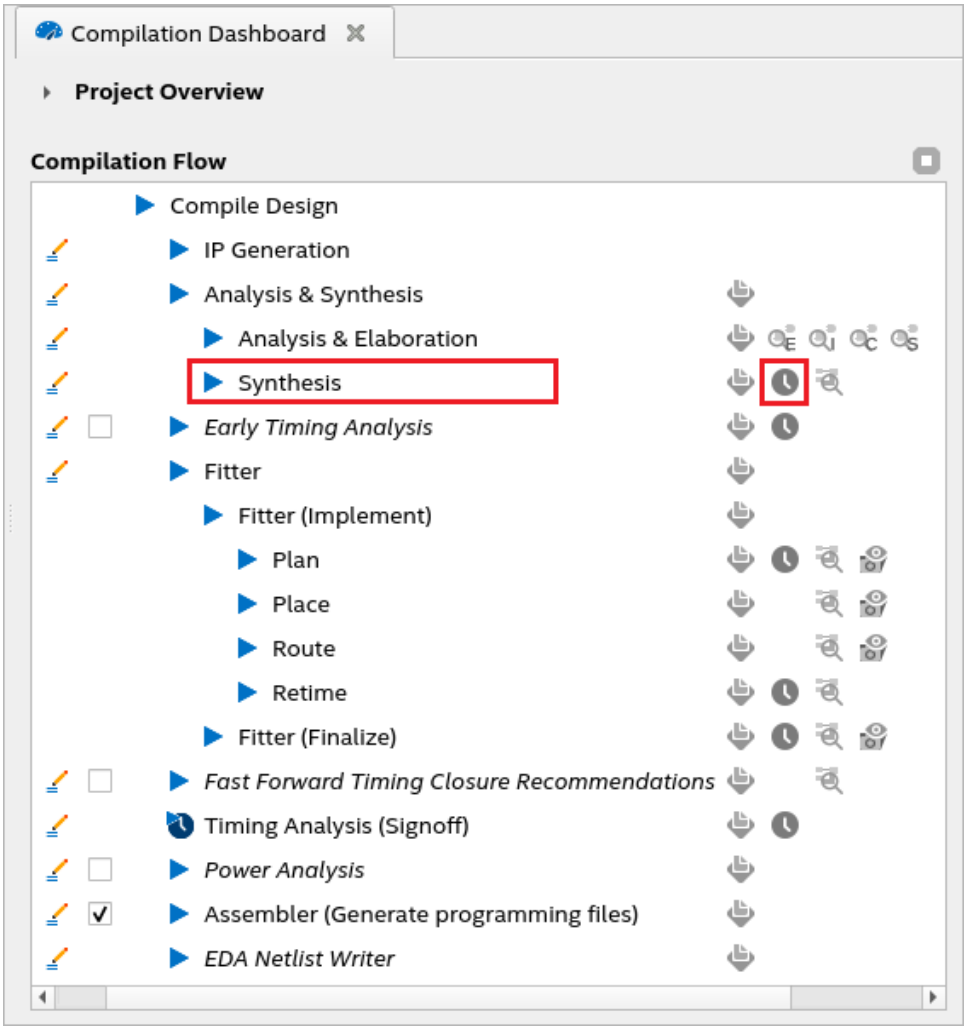
The Synopsys* Design Constraint (SDC) on RTL supports the underlying technology to read the constraints early in the compilation flow and use them in the later stages of the Quartus Prime compilation. However, you can run the flow even without RTL SDCs where you can view the synthesized timing netlist.

Post-synthesis STA defaults to a simple average value delay model based on the types of blocks a net connects. "Average Value" interconnect (IC) delay model to control STA after synthesis. Using the STA_POST_SYN_DELAY_MODEL QSF, you can switch to the "Zero Value" IC delay model to exclude interconnect delays from the timing model.

Note: If you want zero delays, you can also use `create_timing_netlist -zero_ic_delay` argument.

You can now access the Timing Analyzer right after design synthesis in the compilation dashboard, as shown in the following image:

Figure 68. Early Timing Analysis Stage



Post-synthesis static timing analysis (STA) uses a timing netlist representing core blocks and their contents. It also includes periphery blocks (but nothing inside them is modeled) and cell delays of the core blocks. Routing delays between core blocks is represented by IC delays that use the average interconnect model mentioned above.

Post-synthesis STA timing netlist provides you with an early view of your design's core timing. You can run timing analysis reports and constraint diagnostic commands, allowing you to examine SDC-on-RTL constraints.

Perform the following steps to run post-synthesis STA:

1. Create an Quartus Prime software project using your design RTL and associated SDC-on-RTL SDC file.
2. Run **Analysis and Elaboration** compilation stage on your design as follows:

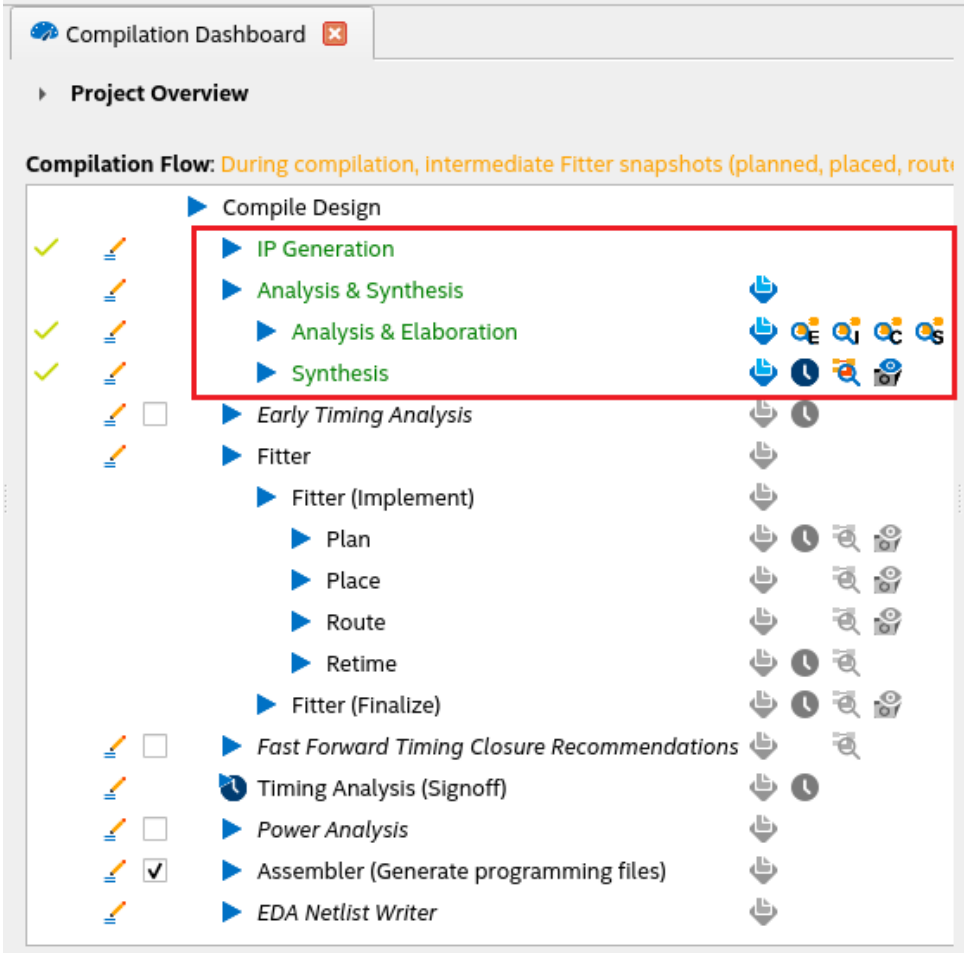
```
quartus_syn --analysis_and_elaboration <design>
```

3. Perform Synthesis on your design as follows:

```
quartus_syn --synthesis <design>
```

You can also perform the above steps using the Quartus Prime software GUI, as shown in the following image:

Figure 69. Performing Post-synthesis STA in the Quartus Prime Software GUI



After running post-synthesis STA on your design, you can use the Timing Analyzer conventionally. However, a fundamental difference in the netlist topography is that the post-synthesis STA timing netlist has no connectivity inside any periphery block.

Note: The post-synthesis STA delay model defaults to a simple average value delay model. Cell delays are computed assuming default configurations.

For post-synthesis constraints, Intel recommends using an SDC-on-RTL file. In cases where this is not possible, post-synthesis STA introduces the `SDC_FILE -read_during_post_syn_and_not_post_fit_timing_analysis` QSF argument, which is used to inform the Timing Analyzer to include a conventional SDC in the list of SDCs to be read during a post-synthesis STA session. This QSF is beneficial for blocks that do not have SDC-on-RTL constraints available. Since the post-synthesis STA netlist differs from the post-plan STA netlist, conventional SDCs written for the post-plan netlist might not function during post-synthesis STA. By creating a new category of SDC files, you can identify scripts you want to load during post-synthesis STA.

1.7.5. Viewing Synthesis Reports

The Compilation Report window opens automatically during compilation processing. The Report window displays detailed synthesis results for each partition in the current project revision.

Figure 70. Synthesis Reports

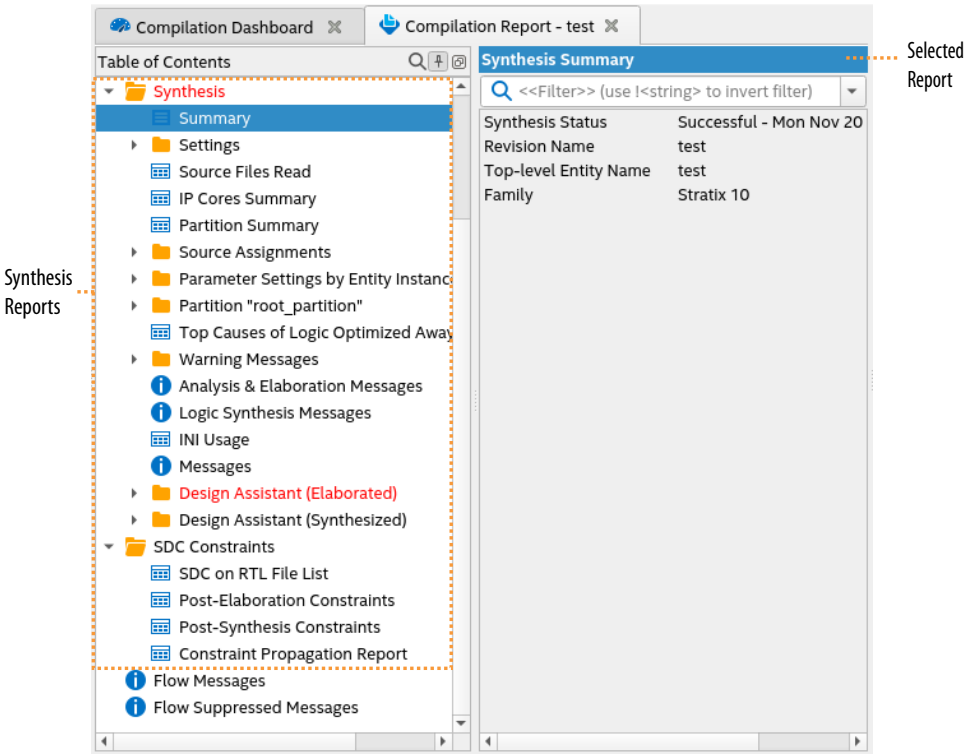


Table 17. Synthesis Reports (Design Dependent)

Generated Report	Description
Summary	Shows summary information about synthesis, such as the status, date, software version, entity name, device family, timing model status, and various types of logic utilization.
Settings	Lists the values of all synthesis settings during design processing.
Source Files Read	Lists details about all source files in design synthesis. Details include file path, file type, and any library information.

continued...

Generated Report	Description
IP Cores Summary	Lists details about each IP core instance in design synthesis. Details include IP core name, vendor, version, license type, entity instance, and IP include file.
Partition Summary	Shows a summary of partitions in the design. Details include partitions names, hierarchy path, partition type, and partition properties.
Source Assignments	A series of reports that list details about source assignments. Details include assignment, value, and source location.
Parameter Settings by Entity Instance	A series of reports that list parameter settings for entities in your design. Details in the reports include parameter name, parameter value, and parameter data type. *Note:* You can view the parameter settings for a module directly from the Project Navigator by locating the module and selecting **View Parameter Settings** in its context-sensitive menu. Compilation Report appears, displaying the parameter settings for the entity.
Partition reports	Each design partition has a series of reports: • Resource Utilization By Entity: Lists the quantity of all types of logic usage for each entity in design synthesis. • Optimization Results: The reports in this folder provide statistics for the following items: — Registers, including registers protected by synthesis and registers removed by synthesis — Multiplexers, including restructuring that synthesis performs and multiplexers implemented. • Post-Synthesis Netlist Statistics • Resource Usage Summary: Lists the quantity of all types of logic usage for the design partition in design synthesis. • RAM Summary: Lists RAM usage details for the design partition in design synthesis. Details include the name, type, mode, and density.
Messages	Lists all information, warning, and error messages that report conditions observed during the Analysis & Synthesis process.
Warning Messages	A series of reports that summarize the warning messages generated during synthesis by providing one entry per message ID, its severity, the count of all its occurrences, and one sample warning message. A separate report is generated for warnings from each source file. General warning messages that are not associated with a source file are put in a separate report.
Design Assistant (Elaborated)	Lists Design Assistant rules that failed during the Analysis & Elaboration stage.
Design Assistant (Synthesized)	Lists Design Assistant rules that failed during the Synthesis stage.
SDC Constraints	Lists all constraints-related reports. Post-Elaboration Constraints, Post-Synthesis Constraints, and Constraint Propagation Reports are available at the end of synthesis. For more information about these reports, refer to Applying the SDC-on-RTL Constraints on page 66.

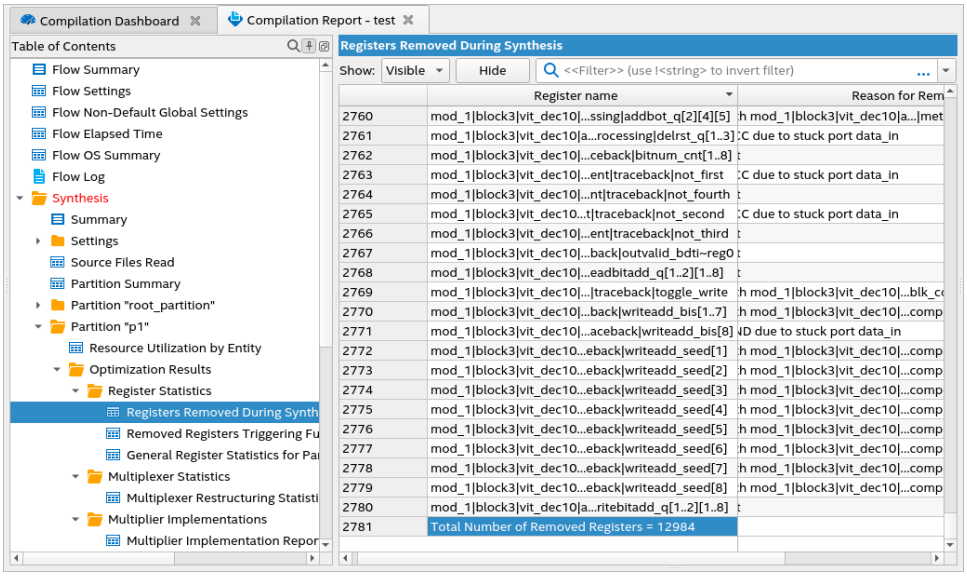
1.7.6. Viewing Synthesis Dynamic Report

The dynamic report feature allows you to interact with a full report that displays only a few items in the GUI. This feature is off by default, and you can enable it through the `synth_rpt_enable_dynamic_report` QSF. Currently, you can view dynamic reports only for the **Registers Removed During Synthesis** report.

In the following report, you can observe that the compiler removed 12984 registers during synthesis. However, after 2780 registers, the remaining registers are truncated from the report due to the GUI threshold. With the dynamic report feature, you can retrieve details of these remaining registers without recompiling your design using the updated GUI threshold.

Note: You can modify the number of removed registers reported through the **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis) ➤ Number of Removed Registers Reported in Synthesis Report** option.

Figure 71. Sample Registers Removed During Synthesis Report



Within one compilation, the compiler stores all items from the report in a dynamic SQLite database under `<project_directory>/dynamic_report/registers_removed.sqlite`. After design synthesis, you can access the SQLite database through Tcl commands to extract a summary of the SQLite database (for example, data column and total size of the items), query from the database using SQL commands (for example, SELECT and WHERE), and retrieve the desired number of truncated lines from the report.

Note:

- If you recompile after generating the SQLite database, contents are rewritten to the same database. If you turn off the QSF during recompilation, the SQLite database retains the information from the previous compile.
- If your design does not contain registers to be removed during synthesis, the compiler does not generate the **Registers Removed During Synthesis** report. Hence, the `dynamic_report` directory and the SQLite database are also not generated.

Executing Tcl Commands

You can execute the `dynamic_report` Tcl commands in one of the following ways:

- Include Tcl commands in a Tcl script and run it using `quartus_syn -t <name>.tcl`.
- Execute a single command directly from your terminal as `quartus_syn --tcl_eval <tcl command>`.
- Execute a single command from the GUI by launching your project in the Quartus Prime software and navigating to **View ➤ Tcl Console** on the menu. In the Tcl Console window, type your Tcl command.

The result is the same irrespective of the method you use to execute your `dynamic_report` Tcl command.

Note: When you execute a `dynamic_report` Tcl command, the Tcl command attempts to open the corresponding SQLite database immediately for further action. The path to open the database is absolute. So, you must ensure that your current directory is where your design is located and not in the `dynamic_report` directory. Otherwise, an error displays indicating the path to the SQLite database is incorrect or an issue with opening the SQLite database.

The following table lists some example `dynamic_report` Tcl commands with various options:

Table 18. `dynamic_report` Tcl Command Examples

`dynamic_report` Command Option	Description	Tcl Command Example
`-help`	Provides detailed help for the `dynamic_report` Tcl command, including currently supported report values, options, and error values.	`dynamic_report -help`
`-long_help`	Provides long help for the `dynamic_report` Tcl command with examples and possible return values.	`dynamic_report -long_help`
`-report <report name>`	Targets the report that `<report name>` specifies. For the **Registers Removed During Synthesis** report, use `"registers removed"`. *Note:* The `-report` option is mandatory as the Tcl command takes further action based on the report name. Absence of the `-report` option or using an unsupported value for `<report name>` results in an error.	`dynamic_report -report "registers removed"`
`-summary`	Generates a summary of the specified report, such as the table name, total size, and column names, which help query from the SQLite database.	`dynamic_report -report "registers removed" -summary`
`-query <sqlite query>`	Accepts an entire SQLite query as an option to retrieve the desired result from the database. For example, `-query "SELECT * FROM <table> WHERE <column> = <...>"`	`dynamic_report -report "registers removed" -query "SELECT * FROM 'Registers Removed During Synthesis' WHERE reason = 'Stuck at VCC due to stuck port data_in'"`

continued...

dynamic_report Command Option	Description	Tcl Command Example
	If the query command is incorrect or incomplete, an error message displays with details. The result of this query is sent to an output file under the <project directory>/dynamic_report directory in ASCII format by default.	
-dump_all	Dumps all items in the report. The result of this query is sent to an output file under the <project directory>/dynamic_report directory in ASCII format by default.	dynamic_report -report "registers removed" -dump_all
-dump_lines <*unsigned integer*>	Dumps first <*unsigned integer*> of lines in the report. An error message displays if the value specified for <*unsigned integer*> is not an unsigned integer or not in the range of 1 and 999999999. The result of this query is sent to an output file under the <project directory>/dynamic_report directory in ASCII format by default.	dynamic_report -report "registers removed" -dump_lines 250
-filename <*file_name*>	Specifies an output file name for the results dumped by -query, -dump_all, and -dump_lines options. -filename is optional. If you do not specify a file name, the compiler names the output file as <*report name*>.ascii. An error message displays if the file name is empty or contains special characters or spaces. *Note:* With the -filename option, you can use the -html or -xml option to generate the output file in HTML or XML format, respectively.	dynamic_report -report "registers removed" -dump_all -filename "my_result"

Attention: The options -summary, -query, -dump_all, and -dump_lines are mandatory but exclusive, which means that the dynamic_report Tcl command requires only one option. If you include multiple options within a single dynamic_report Tcl command, an error message displays requesting you to specify a single option.

Related Information

dynamic_report (::quartus::interactive_synthesis)

1.8. Design Place and Route

The Compiler's Fitter module (quartus_fit) performs design placement and routing. During place and route, the Fitter determines the best placement and routing of logic in the target FPGA device, while respecting any Fitter settings or constraints that you specify.

By default, the Fitter selects appropriate resources, interconnection paths, and pin locations. If you assign logic to specific device resources, the Fitter attempts to match those requirements, and then fits and optimizes any remaining unconstrained design logic. If the Fitter cannot fit the design in the current target device, the Fitter terminates compilation and issues an error message.

The Quartus Prime Pro Edition Fitter introduces a hybrid placement technique that combines analytical and annealing placement techniques. Analytical placement determines an initial mathematical starting placement. The annealing technique then fine-tunes logic block placement in high resource utilization scenarios.

Related Information

- Running the Fitter on page 96
- Viewing Fitter Reports on page 98

1.8.1. Running the Fitter

The Compiler's Fitter module performs all stages of design place and route, including the Plan, Early Place, Place, Route, and Retime stages.

The Quartus Prime Pro Edition Compiler allows control and optimization of each individual Fitter stage, including the Plan, Place, and Route stages. Run all stages of the Fitter as part of a full design compilation, or run any Fitter stage independently after design synthesis. Before running the Fitter, you specify settings that impact Fitter processing.

After running a Fitter stage, view detailed report data and analyze the timing of that stage. The Compiler preserves Fitter results of the final snapshot by default.

1. Specify initial Fitter constraints:
   a. To assign device I/O pins, click **Assignments ➤ Pin Planner**.
   b. To assign device periphery, clocks, and I/O interfaces, click **Tools ➤ Interface Planner**.
   c. To constrain logic placement regions, click **Tools ➤ Chip Planner**.
   d. To specify Fitter optimization goals, click **Assignments ➤ Settings ➤ Compiler Settings**. Compiler Optimization Modes on page 149 describes these options in detail
   e. To fine-tune place and route with advanced Fitter options, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**
2. To run one or more stages of the Fitter, click any of the following commands on the Compilation Dashboard:
   - To run all Fitter stages in sequence, click **Fitter**.
   - To run only device periphery placement and routing, click **Plan**.
   - To run only logic placement, click **Place**.
   - To run only logic routing, click **Route**.
   - To run only retiming of ALM registers into Hyper-Registers, click **Retime**.(2)
   - To run the Implement flow (runs Plan, Place, Route, and Retime stages), click **Fitter (Implement)**.
   - To run the Finalize flow (runs Plan, Place, Route, Retime, and Finalize stages), click **Fitter (Finalize)**.

(2) Retime available for Hyperflex® architecture devices only.

Related Information

- Fitter Settings Reference on page 174
- Step 2: Review Retiming Results on page 117

1.8.1.1. Fitter Commands

Launch Fitter processes from the Processing menu or Compilation Dashboard with Fitter commands.

Table 19. Start Fitter Commands

Command	Description
Start Fitter (Plan)	Loads synthesized periphery placement data and constraints, and assigns periphery elements to device I/O resources. This command creates the planned snapshot.
Start Fitter (Place)	Places all core elements in a legal location. This command creates the placed snapshot.
Start Fitter (Route)	Creates all routing between the elements in the design. This command creates the routed snapshot.
Start Fitter (Retime)	Performs register retiming and moves existing registers into Hyper-Registers to increase performance by removing retiming restrictions and eliminating critical paths. The Compiler may report hold violations for short paths following the Retime stage. This command creates the retimed snapshot.
Start Fitter (Finalize)	Performs post-routing optimization on the design. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths. After correcting the hold violation, the Fitter performs the following physical synthesis optimizations for further setup timing improvement: retiming, LUT and ALM rotation, re-synthesize logic, wire LUT removal, inverter optimization, and skew-optimization for Agilex 7 devices. This stage converts unneeded tiles from High Speed to Low Power. This command creates the final snapshot. For Stratix 10 and Agilex 7 designs, the Fitter also runs post-route fix-up to correct any short path hold violations remaining from retiming.

Note: The Compiler reports violations under the **Compilation Report ➤ Fitter** section. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths.

1.8.1.2. Disabling or Enabling Physical Synthesis Optimization

Physical synthesis optimization improves circuit performance by performing combinational and sequential optimization and register duplication.

The Compiler performs physical synthesis optimization by default during place and route. You can disable or enable physical synthesis optimization and related options by following these steps:

To disable or enable physical synthesis optimization:

1. Click **Assignments ➤ Settings ➤ Compiler Settings**.
2. To enable retiming, combinational optimization, and register duplication, click **Advanced Settings (Fitter)**.
3. Enable **Advanced Physical Synthesis**.
4. View physical synthesis results in the **Netlist Optimizations** report under **Compilation Report ➤ Fitter** section.

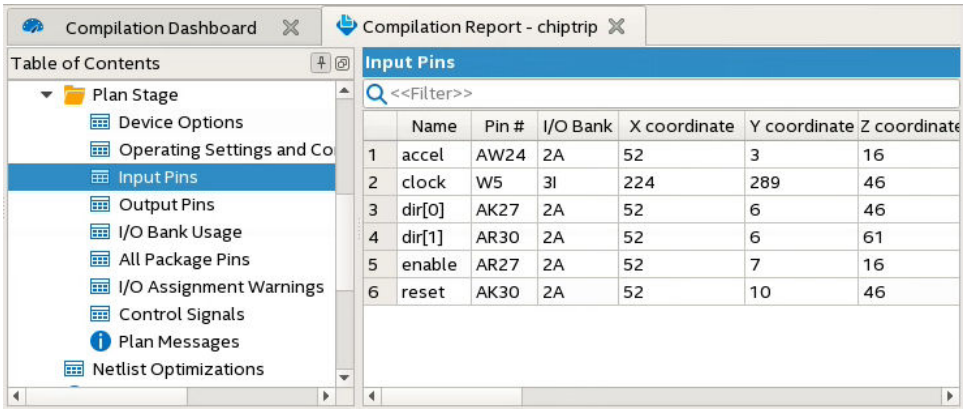
1.8.2. Viewing Fitter Reports

The Fitter generates detailed reports and messages for each stage of place and route. The Fitter Summary reports basic information about the Fitter run, such as date, software version, device family, timing model, and logic utilization.

1.8.2.1. Plan Stage Reports

The Plan stage reports describe the I/O, interface, and control signals discovered during the periphery planning stage of the Fitter.

Figure 72. Plan Stage Reports (Arria 10 and Cyclone 10 GX Designs)



For Arria 10 and Cyclone 10 GX designs, the Plan stage includes the **Global & Other Fast Signals Summary** report that allows you to verify which clocks the Compiler promotes to global clocks. Clock planning occurs after the Plan stage for Stratix 10 and Agilex 7 designs.

NoC Connectivity Report

For Agilex 7 M-Series FPGAs only, the NoC Connectivity Report provides information on connections between NoC initiators and targets in the implemented design, and their associated base addresses. Use this report to verify that the implementation of connection and attribute assignments are correct. The table in this report contains a row for each initiator to target connection. Additional rows may report any unconnected NoC elements. The following columns report data:

- **Group**—displays which NoC group the connection is assigned to.
- **Status**—displays whether the row is for connected or unconnected elements.
- **Initiator**—lists the NoC initiator elements.
- **Target**—lists the NoC target elements.
- **Address**—displays the hexadecimal base address for each connection

NoC Performance Report

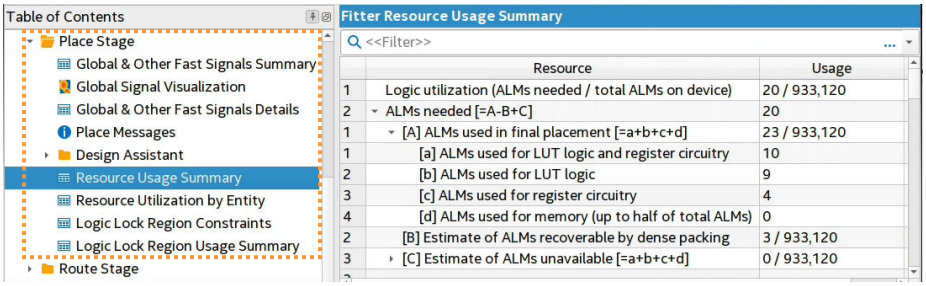
For Agilex 7 M-Series FPGAs only, reports the user-requested read and write bandwidths, as well as the minimum latency of NoC request and response transactions.

The latencies in this report are based on the minimum structural latency with respect to the initiator and target placement. These latencies are for the NoC portion of the path only. These latencies do not include any latency of, for instance, external memory access. Nor do these latencies account for potential delay due to congestion on the NoC. You can achieve lower minimum structural latency by placing the NoC initiators and targets closer together.

1.8.2.2. Place Stage Reports

The Place stage reports describe all device resources the Fitter allocates during logic placement, as well as use of Logic Lock regions and global and other fast signals.

Figure 73. Place Stage Reports



1.8.2.2.1. Global Signal Visualization Report

For Stratix 10 and Agilex 7 designs, you can access the Global Signal Visualization report to view global signal routing and clock sector utilization in an interactive heat-map. This report allows you to track the routing and placement of each individual clock. You can use this data to analyze global signal routing congestion issues, and to debug global signal placement and routing failures.

View global clock tree implementation details and assess capacity to add more global signals to the design. In cases of clock tree synthesis errors, the report can also show targeted regions for failing signals, and competing signals that are contributing to routing congestion.

The interactive heatmap color gradients show clock sector congestion of the clock signals terminating inside the sector. Hover the cursor over a clock signal in the table to highlight the clocks sectors and routing elements. Select a clock signal in the table to dim all irrelevant sectors and routing elements, while highlighting only the clock's sectors and routing elements. The global clock signal routing on different layers displays in the report's stacked layer view.

Figure 74. Global Routing Wire Utilization (Single Layer)

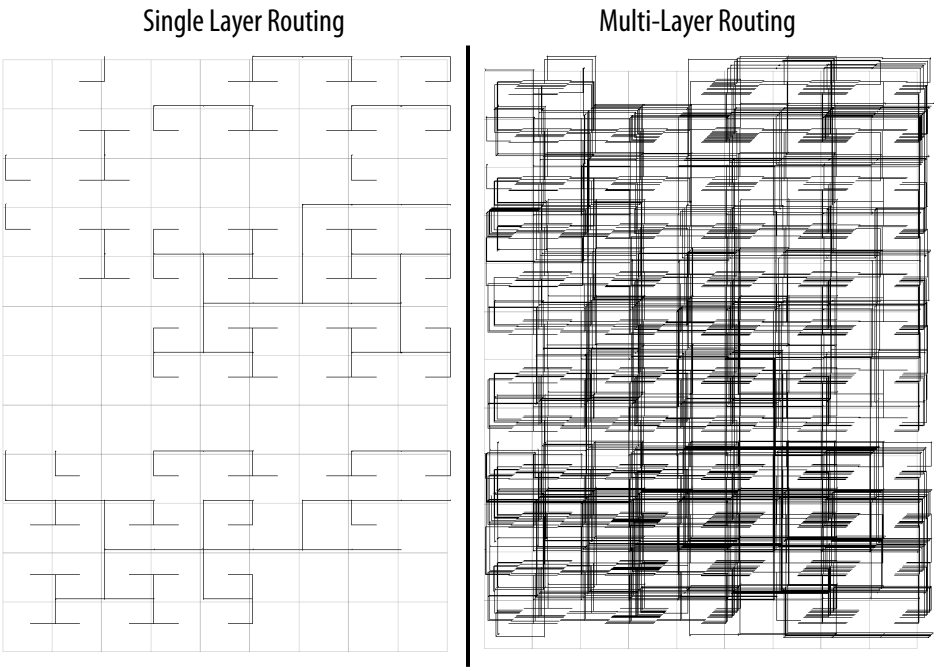
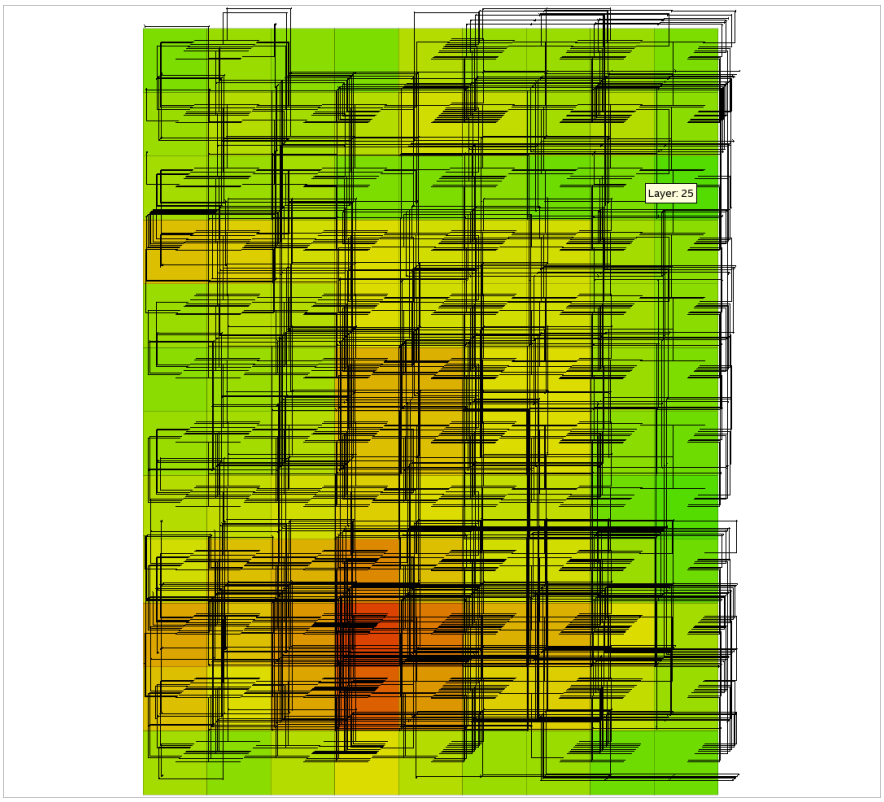
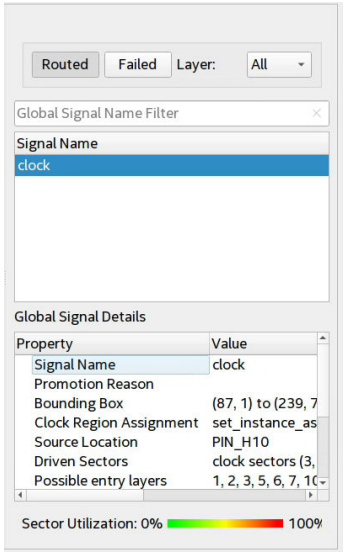


Figure 75. Heat Map Sector and Routing Wire Utilization (All Layers)



Filter the display to **Show Routing Utilization** and **Show Sector Utilization**. The content of the table changes based on the selections you make in the heatmap. You can search for **Signal Names**, and then select the signal names to display its properties in the lower pane. Select any signal to **Locate** in other tools.

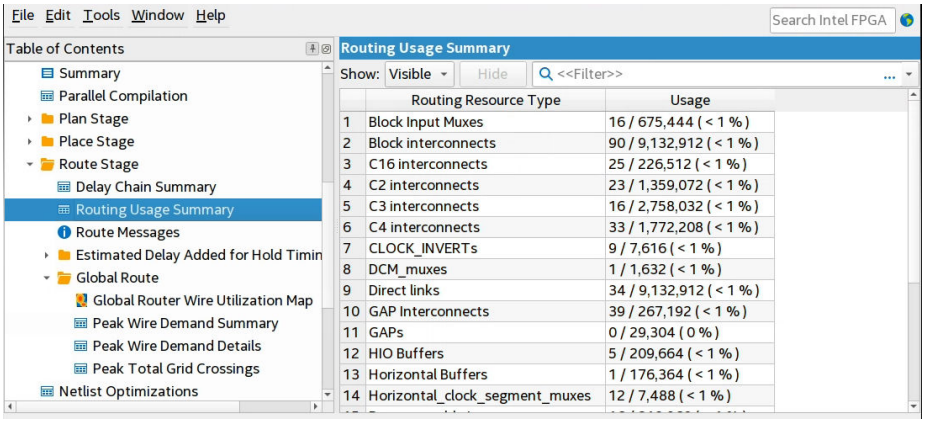
Figure 76. Signals Names and Property Details



1.8.2.3. Route Stage Reports

The Route stage reports describe all device resources that the Fitter allocates during routing. Details include the type, number, and overall percentage of each resource type. The Route stage also reports delay chain summary information.

Figure 77. Route Stage Reports



1.8.2.3.1. Global Router Congestion Hotspot Summary Report

The Global Router Congestion Hotspot Summary report shows congested net hotspots in your design by hierarchical node name.

A global routing congested region is an area of the FPGA where short wire usage in a particular direction exceeds the capacity of that region. A congested net is a net that passes through a congested region.

If the congested area is small, detailed routing can recover by detouring around these overused regions. However, if the congested area is large, you most likely encounter a no route. The exact threshold where that happens varies greatly depending on the design.

Use the Global Router Congestion Hotspot Summary report to identify parts of your RTL code that are associated with routing congestion.

Figure 78. Example of the Global Router Congestion Hotspot Summary Report

Global Router Congestion Hierarchical Summary

<<filter>>

	Hierarchical Node Name	Number of Congested Nets	Full Hierarchy Name
1	▾ |	7583 / 271506 (2.8%)	|
1	|prg[64]~input	1 / 1 (100.0%)	prg[64]~input
2	▾ |u_respe|	7581 / 250835 (3.0%)	u_respe|
1	▾ |g_impl_on.resp_0|	7570 / 250445 (3.0%)	u_respe|g_impl_on.resp_0|
1	|b_or_s_iqsp_out_r_re_16_RNO_0_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_0_~ERTM
2	|b_or_s_iqsp_out_r_re_16_RNO_11_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_11_~ERTM
3	|b_or_s_iqsp_out_r_re_16_RNO_12_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_12_~ERTM
4	|b_or_s_iqsp_out_r_re_16_RNO_14_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_14_~ERTM
5	|b_or_s_iqsp_out_r_re_16_RNO_15_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_15_~ERTM
6	|b_or_s_iqsp_out_r_re_16_RNO_16_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_16_~ERTM
7	|b_or_s_iqsp_out_r_re_16_RNO_17_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_17_~ERTM
8	|b_or_s_iqsp_out_r_re_16_RNO_1_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_1_~ERTM
9	|b_or_s_iqsp_out_r_re_16_RNO_3_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_3_~ERTM
10	|b_or_s_iqsp_out_r_re_16_RNO_5_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_5_~ERTM
11	|b_or_s_iqsp_out_r_re_16_RNO_6_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_6_~ERTM
12	|b_or_s_iqsp_out_r_re_16_RNO_7_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_7_~ERTM
13	|b_or_s_iqsp_out_r_re_16_RNO_8_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_8_~ERTM
14	|b_or_s_iqsp_out_r_re_16_RNO_9_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_9_~ERTM
15	▾ |g_tmc1.u_tmc1|	28 / 4685 (0.6%)	u_respe|g_impl_on.resp_0|g_tmc1.u_tmc1|
1	|p_inreg_s_dt_tin32_r_15_21_	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|g_tmc1.u_tmc1|p_inreg_s_dt_tin32_r_15_21_
2	|p_inreg_s_dt_tin32_r_28_5_~sr_lutram_dout101	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|g_tmc1.u_tmc1|p_inreg_s_dt_tin32_r_28_5_~sr_lutram_dout101
3	|p_inreg_s_dt_tin32_r_28_5_~sr_lutram_dout103	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|g_tmc1.u_tmc1|p_inreg_s_dt_tin32_r_28_5_~sr_lutram_dout103

Note: In the **Number of Congested Nets** column, the numerator is the number of nets under the current hierarchical level that overlap with the largest congestion island, and the denominator is the total number of nets under the current hierarchical level (including nets that are not in the congestion island). The largest congestion island is the largest group of adjacent grids whose short wire usage exceeds capacity. You can see a visualization of this report in the Global Router Wire Utilization Map.

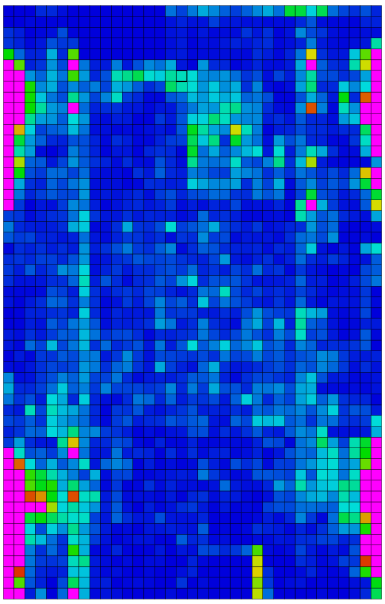
The Global Router Wire Utilization Map provides a visualization of the Global Router Congestion Hotspot Summary Report.

If the size of adjacent congested regions is below a threshold, the report shows the threshold and size of the largest congested region instead of a hierarchical report of congested nets.

1.8.2.3.2. Global Router Wire Utilization Map Report

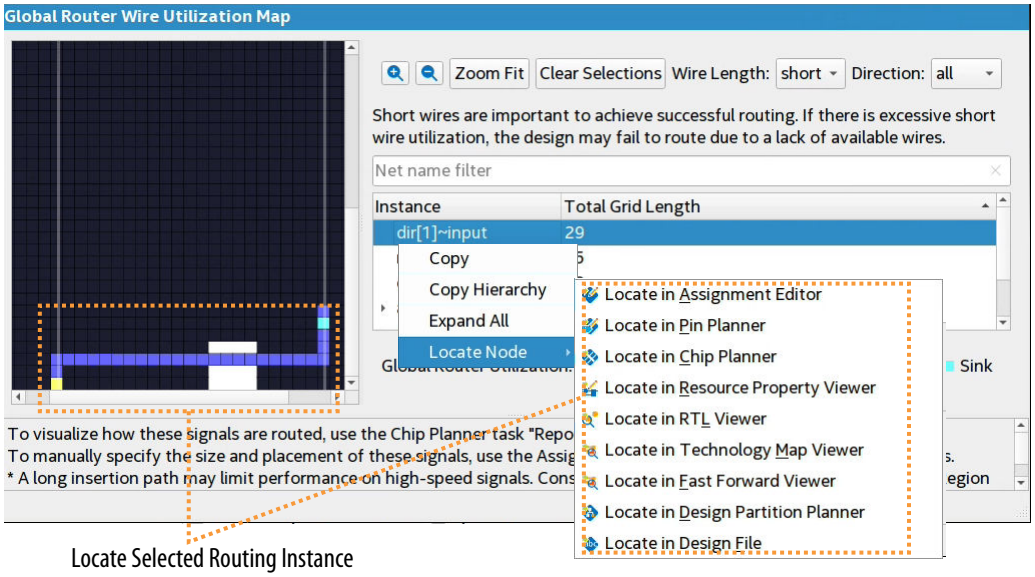
The Global Router Wire Utilization Map report displays global signal routing in an interactive heat-map. This report shows routing utilization rate of long and short routing wires. You can also use this report to obtain a detailed view of all the nets in the design. The report's heatmap grid shows the available device LABs. The color of the grid ranges from blue to red as the utilization rate changes from 0% to 100%. The color becomes pink if the utilization rate is greater than 100%.

Figure 79. Global Router Wire Utilization Heat Map (Multiple Layers)



Filter the Global Router Wire table to show **short** or **long Wirelengths** in all **Directions**. The content of the table changes based on the selections you make in the heatmap. You can search for **Signal Names**, and then single- or multi-select the signal names to display properties in the lower pane. Select one or more nodes in the table to **Locate** in various editors.

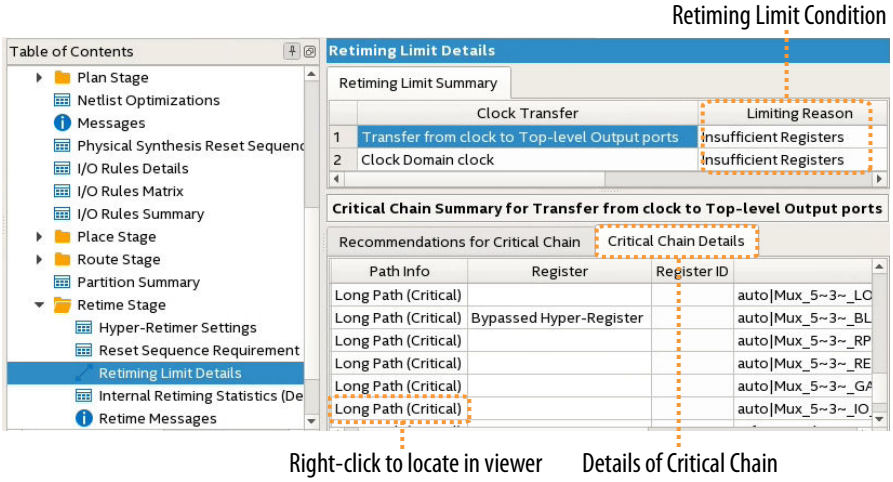
Figure 80. Global Router Wire Utilization Details



1.8.2.4. Retime Stage Reports

For Stratix 10 and Agilex 7 designs, the Fitter generates detailed reports showing the results of the Retime stage, including the Retiming Limit Details report. This report lists the limiting reason, along with the critical chain and recommendations for the critical chain for each clock transfer.

Figure 81. Retiming Limit Details



Related Information

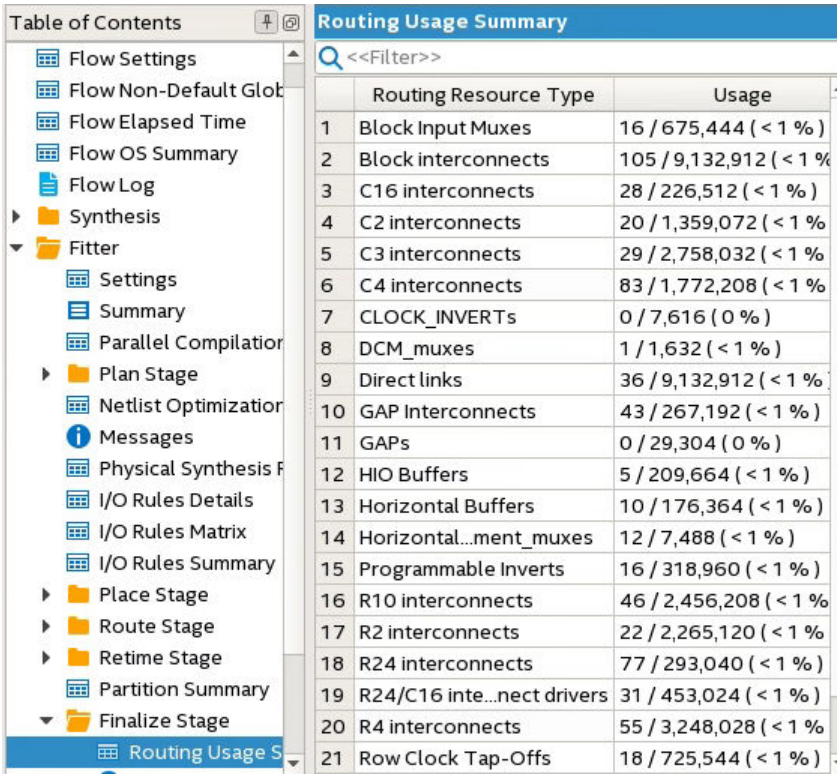
- Viewing Critical Chains
- Report Retiming Restrictions
- Interpreting Critical Chain Reports

1.8.2.5. Finalize Stage Reports

The Finalize stage reports describe final placement and routing operations, including:

- HSLP Summary. For Arria 10 and Cyclone 10 GX designs, the Compiler converts unnecessary tiles to High-Speed or Low-Power (HSLP) tiles.
- Post-route hold fix-up data. For Stratix 10 and Agilex 7 designs, the Compiler reports hold violations for short paths following the Retime stage. The Fitter identifies and corrects the short paths with hold violations during the Fitter (Finalize) stage by adding routing wire along the paths.

Figure 82. Finalize Stage Reports (Stratix 10 Design)



Related Information

Step 2: Review Retiming Results on page 117
For information on Retiming and Fast Forward compilation reports

1.9. Incremental Optimization Flow

The Quartus Prime Pro Edition software supports incremental optimization at each stage of design compilation. In incremental optimization, you run and optimize each compilation stage independently before running the next compilation module in sequence. The Compiler preserves the results of each stage as a snapshot for analysis. When you make changes to your design or constraints, the Compiler only runs stages impacted by the change. Following synthesis or any Fitter stage, you can view results and perform timing analysis; modify design RTL or Compiler settings as needed; and then re-run synthesis or the Fitter and evaluate the results of these changes. Repeat this process until the module performance meets requirements. This flow maximizes the results at each stage without waiting for full compilation results.

Figure 83. Incremental Optimization Flow

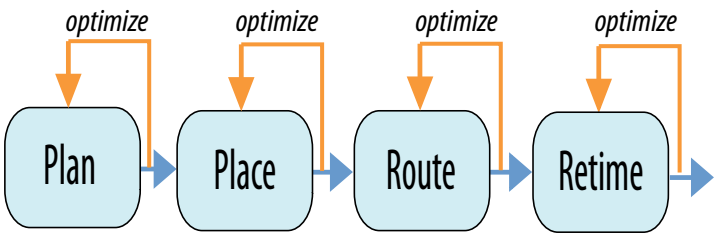


Table 20. Incremental Optimization at Fitter Stages

Fitter Stage	Incremental Optimization
Plan	After this stage, you can run post-Plan timing analysis to verify timing constraints, and validate cross-clock timing windows. View the placement and properties of the periphery (I/O).
Place	After this stage, validate resource and logic utilization in the Compilation Reports, and review placement of design elements in the Chip Planner.
Route	After this stage, perform detailed setup and hold timing closure in the Timing Analyzer, and view routing congestion in the Chip Planner.
Retime	After this stage, review the Retiming results in the Fitter report and correct any restrictions limiting further retiming optimization.

Note: The Compiler saves the planned, placed, routed, and retimed snapshots during full compilation only if you turn on **Enable Intermediate Fitter Snapshots** (**Assignments ➤ Settings ➤ Compiler Settings**). You can also run any intermediate Fitter stage independently to generate the snapshot for that stage.

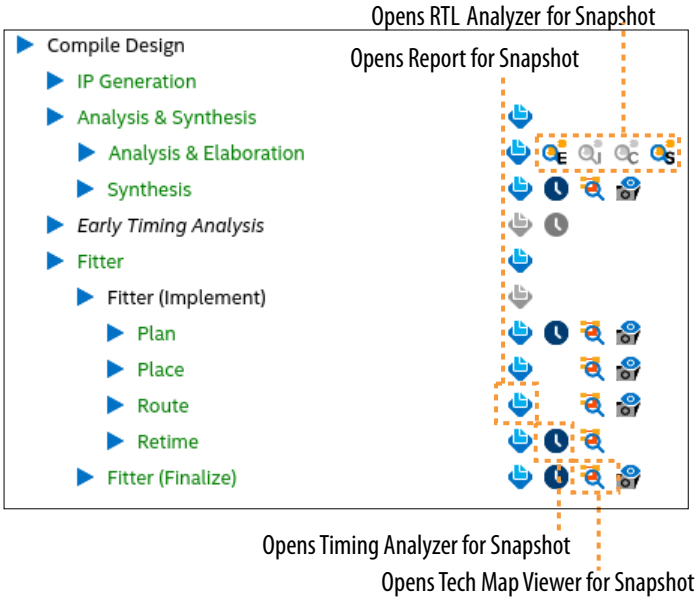
1.9.1. Concurrent Analysis During Synthesis or Fitting

If you run Analysis & Synthesis, or the Fitter, you can access results while downstream Fitter stages are still running. Once the **Concurrent Analysis** icons become active in the dashboard, you can view the analysis without interrupting compilation.

During Analysis & Synthesis, you can click the **Concurrent Analysis** icons on the Dashboard to view reports, the RTL Viewer, or the Technology Map Viewer. While the Fitter is processing, you can analyze timing during the stages displaying the **Timing Analyzer** icon, and view Technology Map Viewer snapshots during Fitter stages.

Caution: Do not attempt to change the SDC-on-RTL constraints during concurrent analysis. SDC-on-RTL constraints are read and loaded in the post-elaboration stage. If you modify the constraints at the Fitter stages, you must restart the compilation from the beginning.

Figure 84. Concurrent Analysis Options



1.9.2. Analyzing Compiler Snapshots

You can analyze the results of a compilation snapshot to evaluate your design before running the next stage, or before running a full compilation. Analyze Compiler snapshots to isolate potential problems and reduce the overall time you spend running design compilation.

1.9.2.1. Running Snapshot Viewer

You can run the Snapshot Viewer to assist with timing closure and design analysis after running the Plan, Place, Route, or Finalize stages of the Fitter. The Snapshot Viewer allows you to run various analysis tasks from the **Flow Navigator** to achieve faster timing closure and maximize design performance.

Figure 85. Snapshot Viewer Flow Navigator

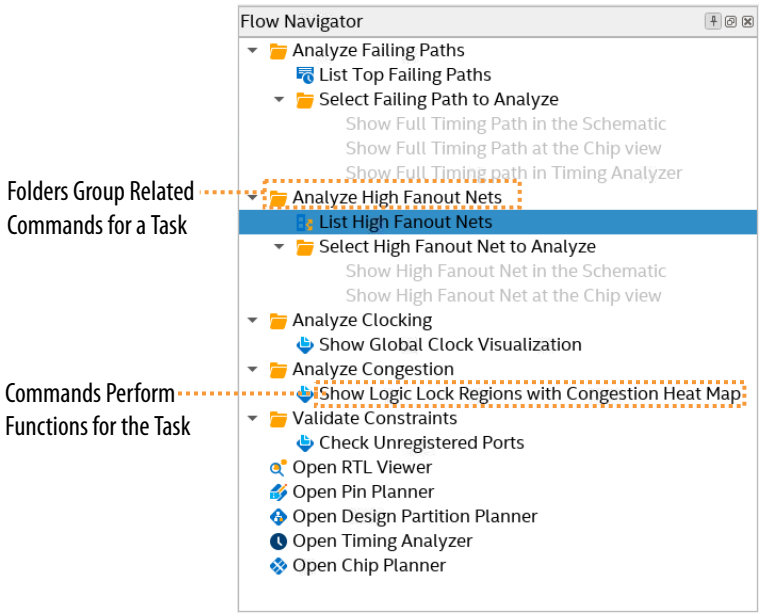


Table 21. Snapshot Viewer Tasks and Commands

Design Task	Available at Snapshot	Snapshot Viewer Commands
Timing Closure—**Analyze Failing Paths**	Planned, Placed, Routed, Finalized	• **List Top Failing Paths**—lists all top failing paths in **Snapshot Selections**. Select a path to locate in RTL Viewer or Chip Planner. • **Show Full Timing Path in the Schematic**—highlights the path in RTL Viewer for further analysis. • **Show Full Timing Path in Timing Analyzer**—The path loads in the Timing Analyzer for further analysis.
	Placed, Routed, Finalized	• **Show Full Timing Path in the Chip View**—highlights the path in Chip Planner for further analysis.
Timing Closure—**Analyze Clocking** This task is available only for Stratix 10 devices.	Placed, Finalized	**Show Global Clock Visualization**—loads the Global Signal Visualization report for the snapshot that allows you to visualize clock sector utilization.
Timing Closure—**Analyze High Fanout Nets**	Placed, Routed, Finalized	• **List High Fanout Nets**—lists high fan-out nets in **Snapshot Selections**. Select a path to locate in RTL Viewer or Chip Planner. • **Show High Fanout Nets in the Schematic**—highlights the paths in RTL Viewer for further analysis. • **Show High Fanout Nets in the Chip View**—highlights the paths in Chip Planner for further analysis.
Timing Closure—**Validate Constraints**	Planned	**Timing Exceptions**—displays the Timing Exceptions Results report that identifies timing paths with hold or removal slack exceeding threshold.
	Planned, Placed, Finalized	**Check Unregistered Ports**—displays the Check Unregistered Ports Results report that identifies unregistered partition inputs and paths.
Timing Closure—**Analyze Congestion**	Placed, Routed, Finalized	**Show Logic Lock Regions with Congestion Heat Map**—the Chip Planner displays the Logic Lock regions in a congestion heat map for further analysis.

The following sections describe each analysis task in detail.

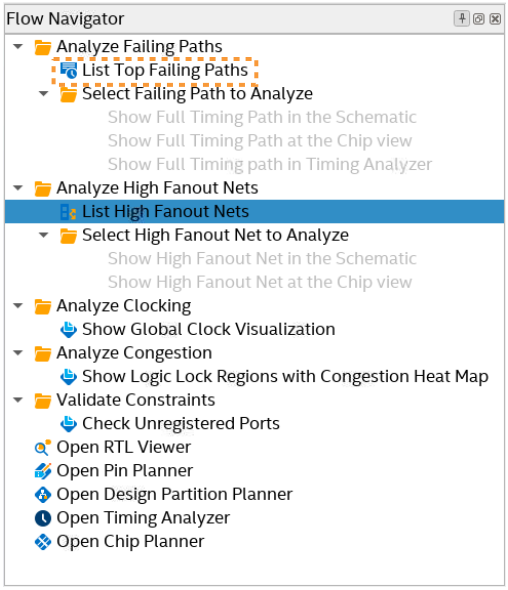
1.9.2.1.1. Analyzing Failing Paths with Snapshot Viewer

1. To run the Plan, Place, or Route stage of the Fitter, double-click the stage in the Compilation Dashboard.
2. After the stage completes, click the **Snapshot Viewer** icon for that stage in the Compilation Dashboard. The Snapshot Viewer opens.

Figure 86. Snapshot Viewer Icon

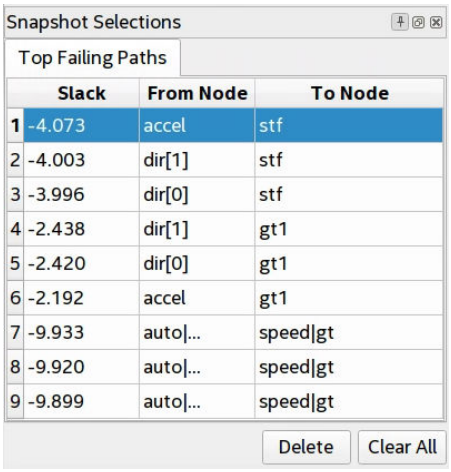
3. Under **Analyze Failing Paths**, click **List Top Failing Paths**.

Figure 87. List Top Failing Paths



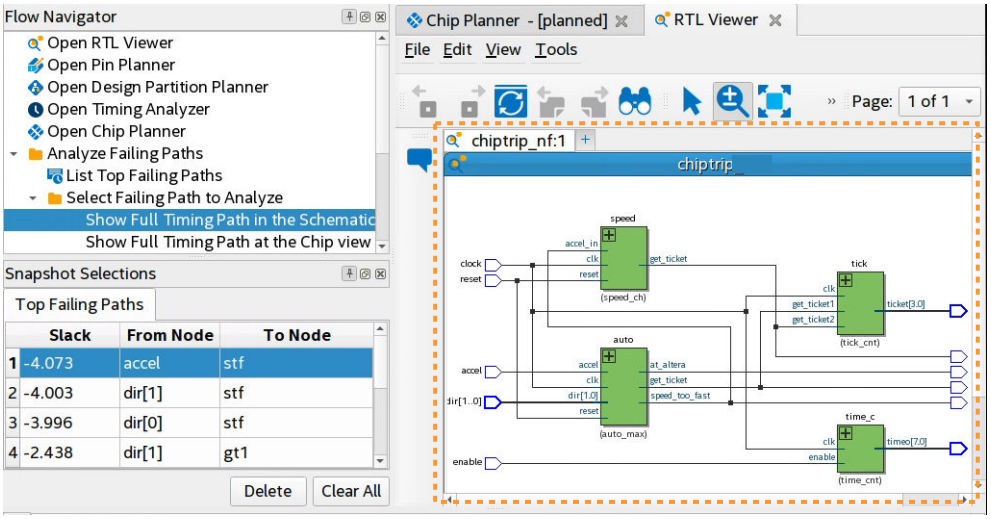
4. In **Snapshot Selections**, select the failing path for analysis.

Figure 88. Snapshot Selections



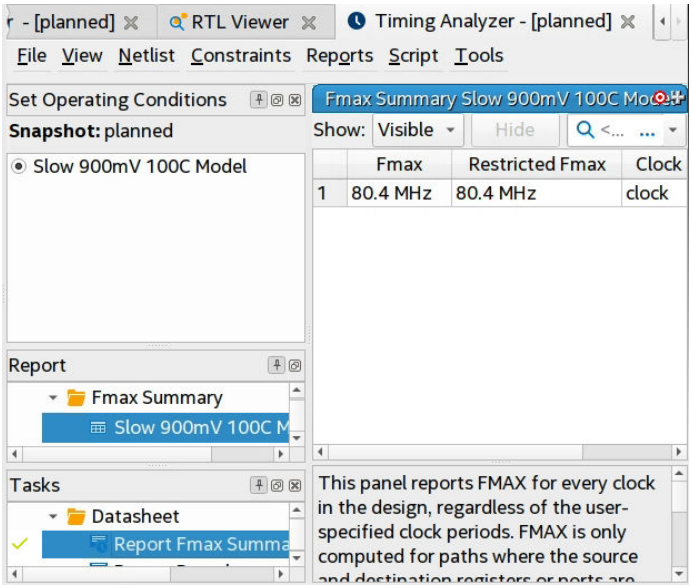
5. Under **Select Failing Path to Analyze**, click **Show Full Timing Path in the Chip View**. The path displays and highlights in the Chip Planner for further analysis.
6. Under **Select Failing Path to Analyze**, click **Show Full Timing Path in Schematic**. The path displays and highlights in RTL Viewer for further analysis.

Figure 89. Show Full Timing Path in Schematic



7. Under **Select Failing Path to Analyze**, click **View Path Characteristics**. The path loads in the Timing Analyzer for further analysis.

Figure 90. View Path Characteristics in Timing Analyzer

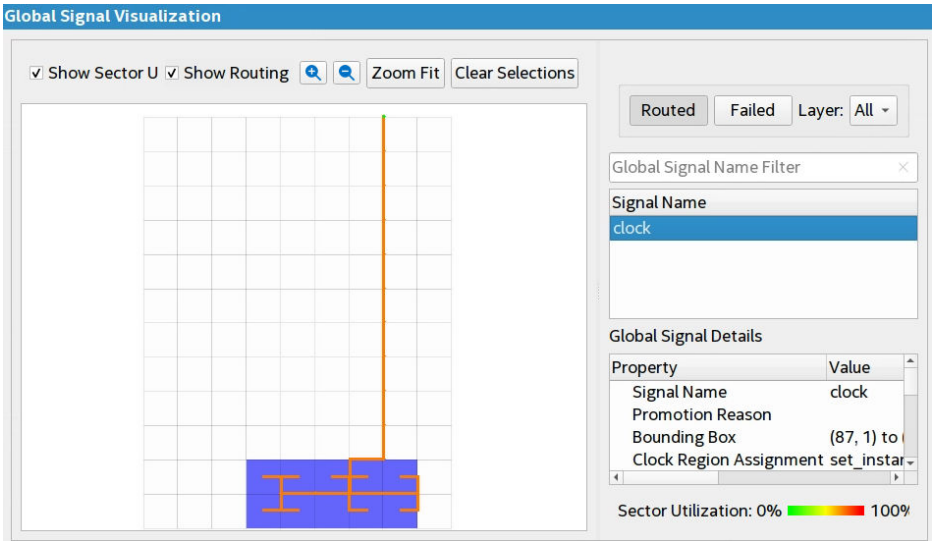


1.9.2.1.2. Analyzing Clocking with Snapshot Viewer

This task is available only for Stratix 10 devices.

1. Run the Place stage, and then click the **Snapshot Viewer** icon for the stage in the Compilation Dashboard. The Snapshot Viewer opens.
2. Under **Analyze Clocking**, double-click **Show Global Clock Visualization**. The Global Signal Visualization report displays in Snapshot Viewer for analysis of clock sector and routing utilization.

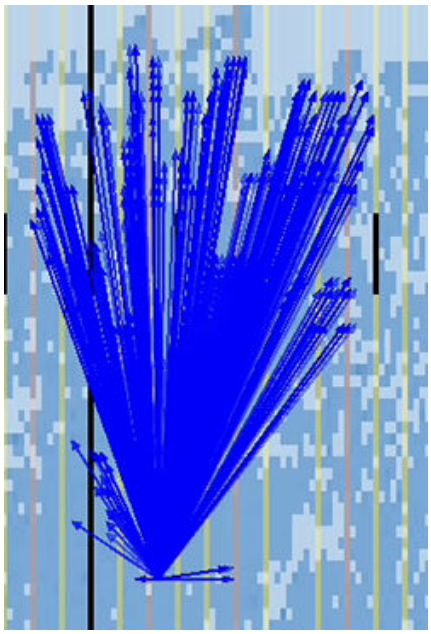
Figure 91. Global Clock Visualization Report



1.9.2.1.3. Analyzing High Fan-out Nets with Snapshot Viewer

1. To run the Place or Route stage of the Fitter, double-click the stage in the Compilation Dashboard.
2. After the stage completes, click the **Snapshot Viewer** icon for that stage in the Compilation Dashboard. The Snapshot Viewer opens.
3. Under **Analyze High Fanout Nets**, click **Show High Fanout Nets in the Schematic**. The path displays and highlights in Tech Map Viewer for further analysis.
4. Under **Analyze High Fanout Nets**, click **Show High Fanout Nets in the Chip View**. The path displays and highlights in the Chip Planner for further analysis.

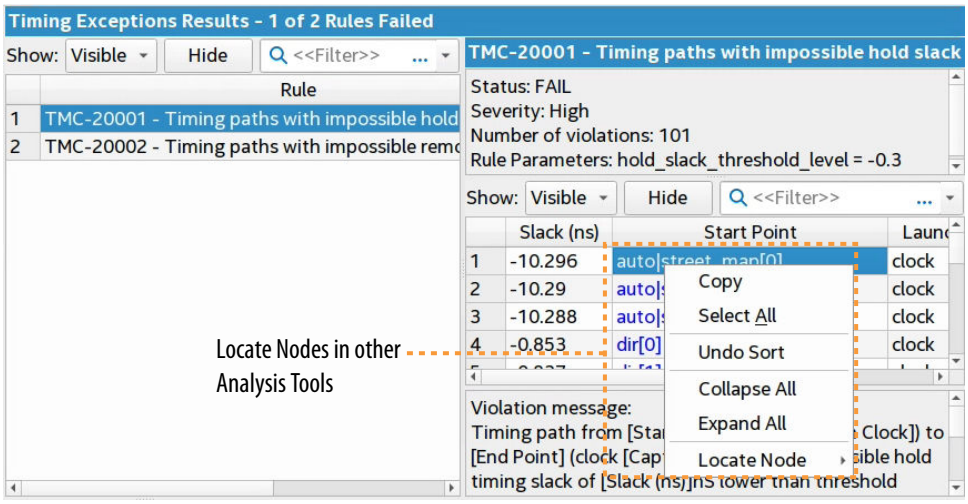
Figure 92. Non-Global High Fan-Out Signal in Chip Planner



1.9.2.1.4. Validating Timing Constraints with Snapshot Viewer

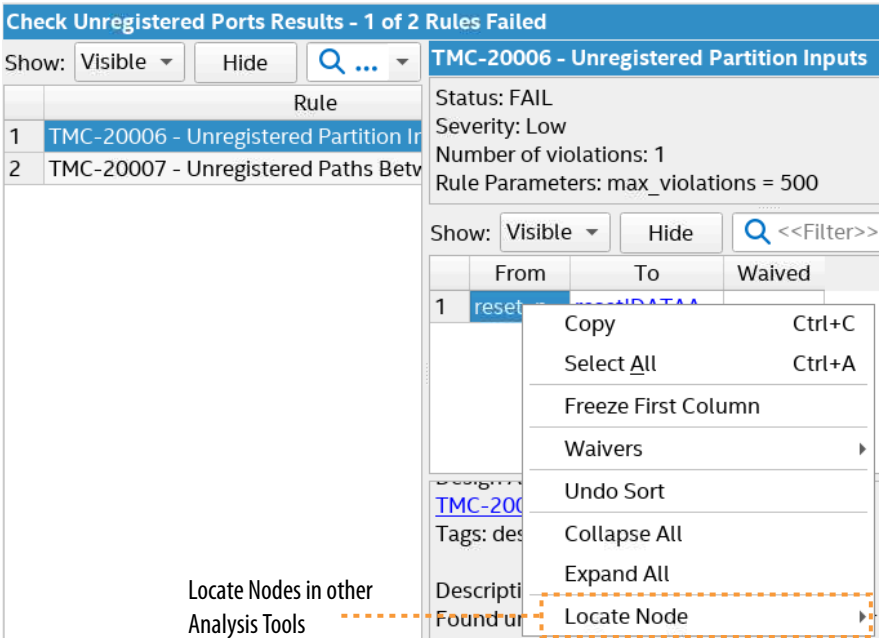
1. To run the Plan or Place stage of the Fitter, double-click the stage in the Compilation Dashboard.
2. After the stage completes, click the Snapshot Viewer icon for that stage in the Compilation Dashboard. The Snapshot Viewer opens.
3. Under **Validate Constraints**, double-click **Timing Exceptions**. The Timing Exception Results report opens, allowing additional analysis and locating to other tools.

Figure 93. Validate Constraints—Timing Exceptions Report



4. Under **Validate Constraints**, double-click **Check Unregistered Ports**.

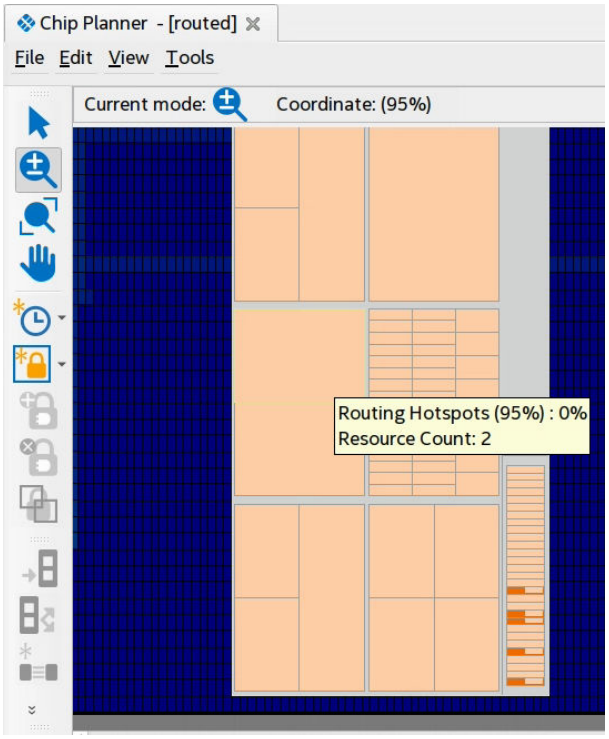
Figure 94. Validate Constraints—Check Unregistered Ports Report



1.9.2.1.5. Analyzing Congestion with Snapshot Viewer

1. To run the Place or Route stage of the Fitter, double-click the stage in the Compilation Dashboard.
2. After the stage completes, click the **Snapshot Viewer** icon for that stage in the Compilation Dashboard. The Snapshot Viewer opens.
3. Under **Analyze Congestion**, double-click **Show Logic Lock Regions with Congestion Heat Map**. The Chip Planner displays the Logic Lock regions in a congestion heat map for further analysis.

Figure 95. Show Logic Lock Regions with Congestion Heat Map



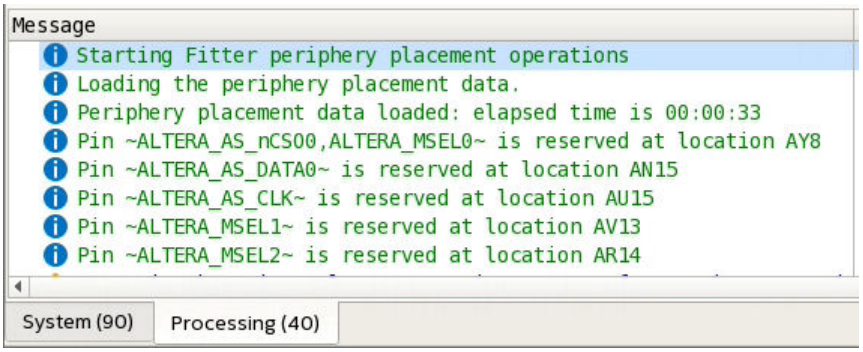
Related Information

- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
- Intel Quartus Prime Pro Edition User Guide: Design Optimization

1.9.3. Validating Periphery (I/O) after the Plan Stage

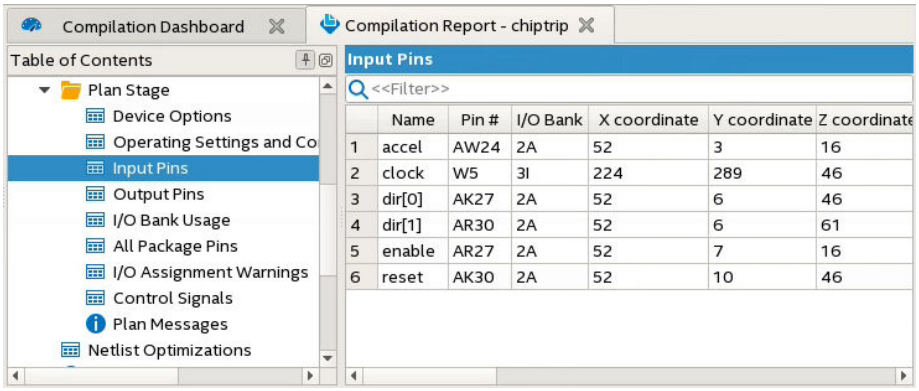
The Compiler begins periphery placement during the Plan stage, and reports data about periphery elements, such as I/O pins and PLLs. After the Plan stage, view the Compilation Report to evaluate the placement of periphery elements before proceeding to the next compilation stage.

Figure 96. Plan Stage Periphery Placement Message



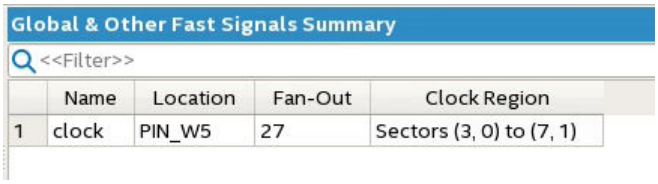
1. In the Compilation Dashboard, click the **Plan** stage.
2. In the Compilation Report, under the **Plan Stage** folder, click the **Input Pins**, **Output Pins**, **I/O Bank Usage**, **PLL Usage Summary**, or other reports. Verify attributes of the I/O pins, such as the physical pin location, I/O standards, and PLL placement.

Figure 97. Input Pins Report



3. For Arria 10 and Cyclone 10 GX designs, click **Global & Other Fast Signals Summary** report to verify which clocks the Compiler promotes to global clocks. Clock planning occurs after the Plan stage for Stratix 10 and Agilex 7 designs.

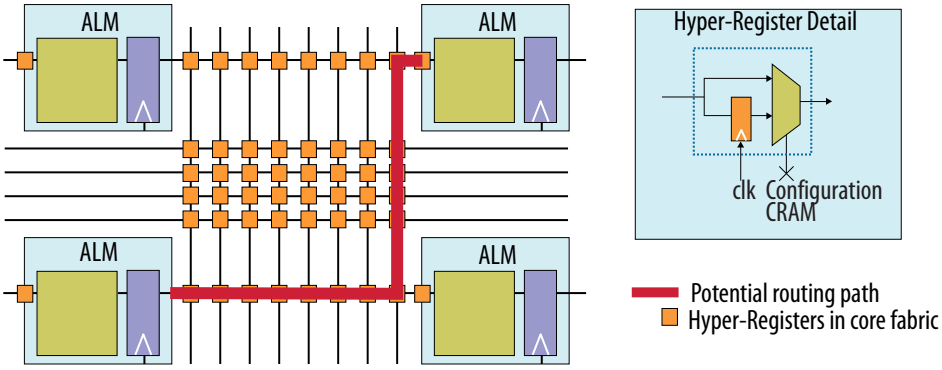
Figure 98. Global & Other Fast Signals Report Shows Clock Promotion (Intel Arria 10 and Intel Cyclone 10 GX FPGAs)



1.10. Fast Forward Compilation Flow

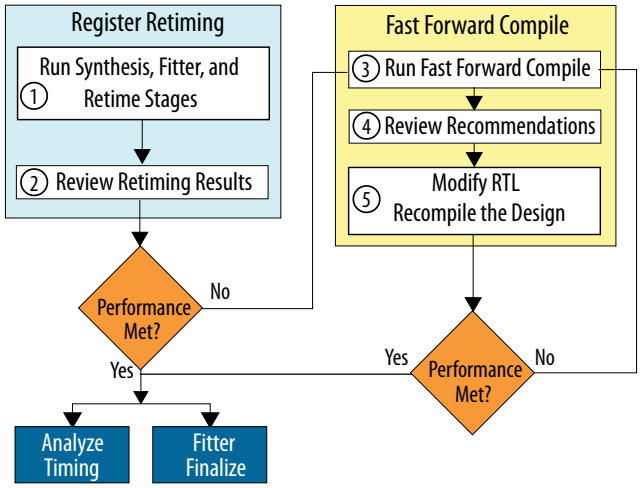
The Hyperflex architecture includes multiple Hyper-Registers in every routing segment and block input. Maximizing the use of Hyper-Registers improves the balance of time delays between registers, and mitigates critical path delays. Fast Forward compilation generates design recommendations to help you to break performance bottlenecks and maximize use of Hyper-Registers to drive the highest performance in Stratix 10 and Agilex 7 designs.

Figure 99. Hyper-Registers in Hyperflex Architecture



The Fast Forward compilation reports show precisely where to make the most impact with RTL changes, and the performance benefits you can expect from each change after removing retiming restrictions. The Fast Forward compilation flow includes the following high-level steps:

Figure 100. Fast Forward Compile Flow



1.10.1. Step 1: Run Register Retiming

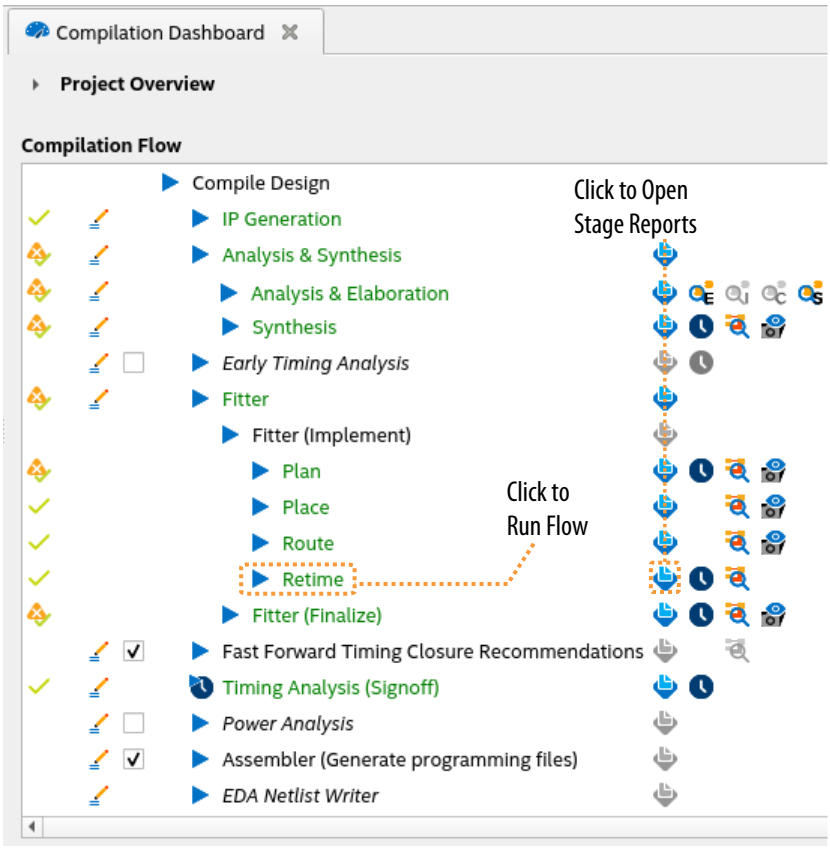
Register retiming improves design performance by moving registers out of ALMs and retimes them into Hyper-Registers in the Stratix 10 and Agilex 7 device interconnect.

The Fitter runs the **Retime** stage automatically following place and route when you target an Stratix 10 or Agilex 7 device. Alternatively, start or stop the individual **Retime** stage in the Compilation Dashboard. After running register retiming, view the Fitter reports to optimize remaining critical paths.

To run register retiming:

1. Click **Retime** on the Compilation Dashboard. The Compiler runs prerequisite stages automatically before running Retime stage.

Figure 101. Retiming Stage in Compilation Dashboard



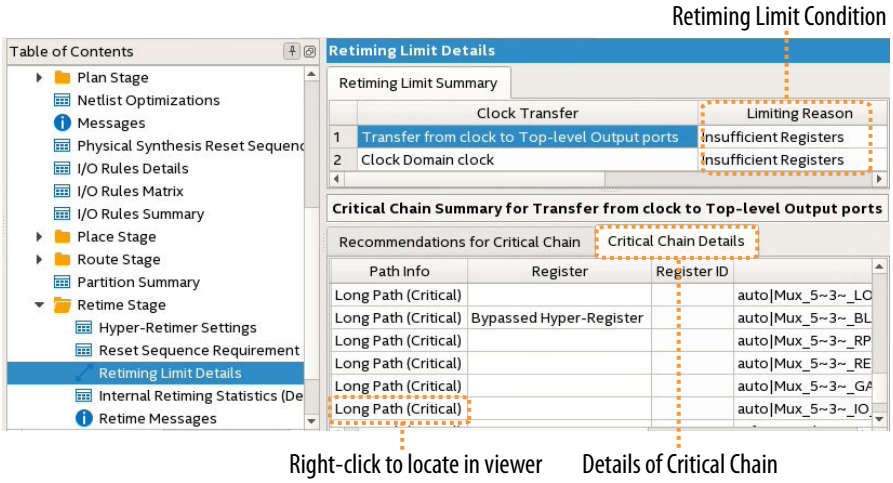
2. Review the results of the register retiming stage, as Step 2: Review Retiming Results on page 117 describes.

1.10.2. Step 2: Review Retiming Results

Follow these steps to review the results of register retiming. Use the results to determine if additional performance improvements are necessary and possible by removing retiming limits.

1. To open the **Retiming Limit Details** report, click the **Report** icon for the **Retime** stage in the Compilation Dashboard. The **Retiming Limit Details** lists the number of registers moved, their paths, and the limiting reason preventing further retiming.

Figure 102. Retiming Limit Details



2. To further optimize, resolve any **Limiting Reason** in your design, and then rerun the **Retime** stage, as necessary.

Table 22. Retiming Limit Details Report Data

Report Data	Description
Clock Transfer	Lists each clock domain in your design. Click the domain to display data about each entry.
Limiting Reason	Specifies any design condition that prevent further register retiming improvement, such as any of the following conditions: • **Insufficient Registers**—indicates insufficient quantity of registers at either end of the chain for retiming. Adding more registers can improve performance. • **Short Path/Long Path**—indicates that the critical chain has dependent paths with conflicting characteristics. For example, one path improves performance with more registers, and another path has no place for additional hyper-registers. • **Path Limit**—indicates that there are no further Hyper-Register locations available on the critical path, or the design reached a performance limit of the current place and route. • **Loops**—indicates a feedback path in a circuit. When the critical chain includes a feedback loop, retiming cannot change the number of registers in the loop without changing functionality. The Compiler can retime around the loop without changing functionality. However, the Compiler cannot place additional registers in the loop.
Critical Chain Details	Lists register timing path associated with the retiming limitations. Right-click any path to **Locate Critical Chain in Technology Map Viewer**.

3. If register retiming achieves all performance goals for your design, proceed to Fitter (Finalize) and Timing Analysis stages of compilation.
4. If your design requires further optimization, run **Fast Forward Timing Closure Recommendations** as Step 3: Run Fast Forward Compile on page 120 describes.

1.10.2.1. Locate Critical Chains

The **Retiming Limit Details** report shows the design paths that limit further register retiming. Right-click any path to locate the path in the Technology Map Viewer - Post-fitting view. This viewer displays a schematic representation of the complete design after place, route, and register retiming. To view the retimed netlist in the Technology Map Viewer, follow these steps:

1. To open the **Retiming Limit Details** report, click the **Report** icon next to the **Retime** stage in the Compilation Dashboard.
2. Right-click any path in the **Retiming Limit Details** report and click **Locate Critical Chain in Technology Map Viewer**. The netlist displays as a schematic in the Technology Map Viewer.

Figure 103. Technology Map Viewer

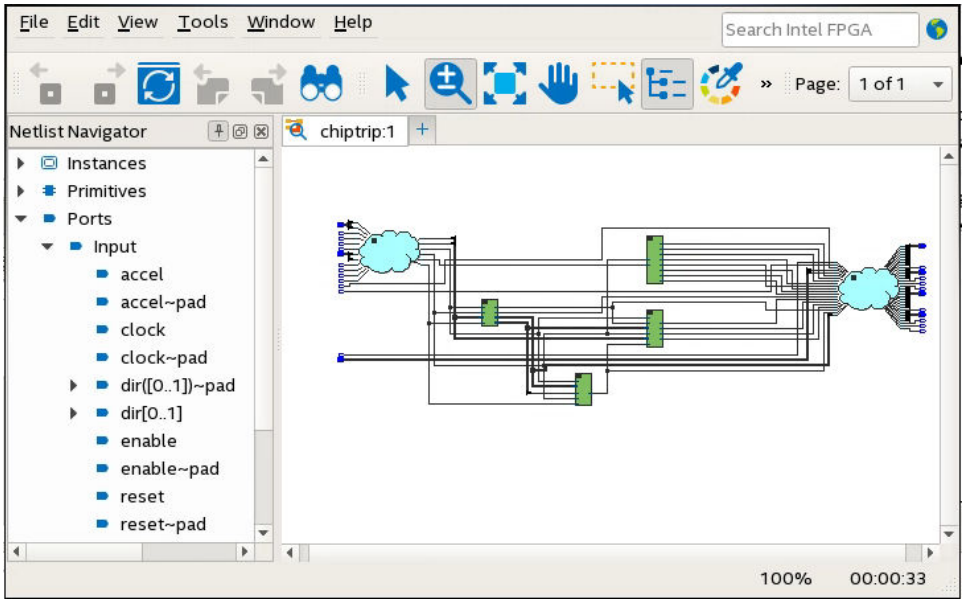
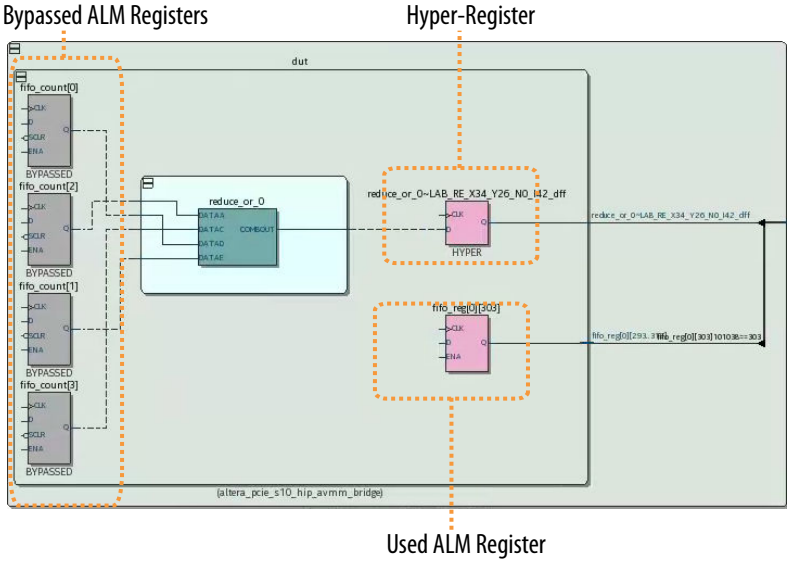


Figure 104. Post-Fit Viewer After Retiming

In the post-fit viewer, bypassed ALM registers are gray. Hyper-Registers are pink with the word "HYPER" below them. Used ALMs are pink without the word "HYPER" below them



1.10.3. Step 3: Run Fast Forward Compile

Fast Forward compilation generates design-specific timing closure recommendations, and predicts the maximum performance after the removal of all timing restrictions.

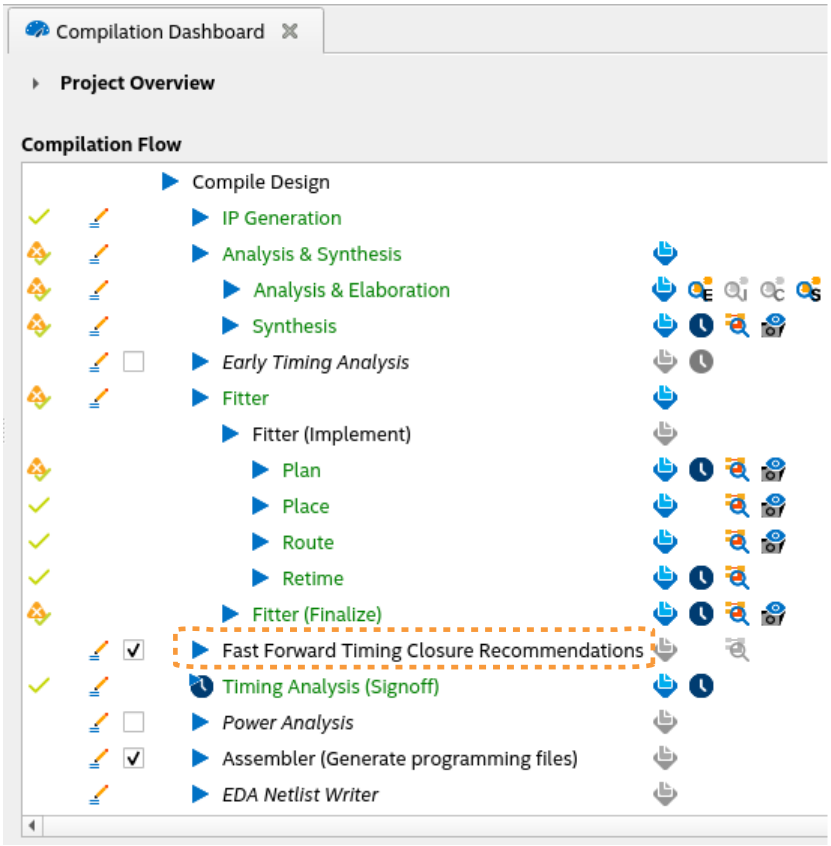
You can review the Fast Forward recommendations and implement the changes in your RTL that remove timing restrictions and enable mobility within the netlist for register Hyper-Retiming.

You can run Fast Forward compilation for the entire design hierarchy, or for only specific instances in the hierarchy, as Fast Forward Compile By Hierarchy on page 121 describes.

To generate Fast Forward timing closure recommendations, follow these steps:

1. Optionally, specify any of the following options if you want to automate or refine Fast Forward analysis:
   - If you want to run Fast Forward compilation during each full compilation, click **Assignments ➤ Settings ➤ Compiler Settings ➤ HyperFlex** and enable **Run Fast Forward Timing Closure Recommendations during compilation.**
   - If you want to modify how Fast Forward compilation interprets specific I/O and block types, click **Assignments ➤ Settings ➤ Compiler Settings ➤ HyperFlex ➤ Advanced Settings**.
2. On the Compilation Dashboard, click **Fast Forward Timing Closure Recommendations**. The Compiler runs prerequisite synthesis or Fitter stages automatically, as needed, and generates timing closure recommendations in the Compilation Report.

Figure 105. Running Fast Forward Compilation



3. View timing closure recommendations in the Compilation Report to evaluate design performance and implement key RTL performance improvements, as Step 4: Review Fast Forward Results on page 124 describes.

1.10.3.1. Fast Forward Compile By Hierarchy

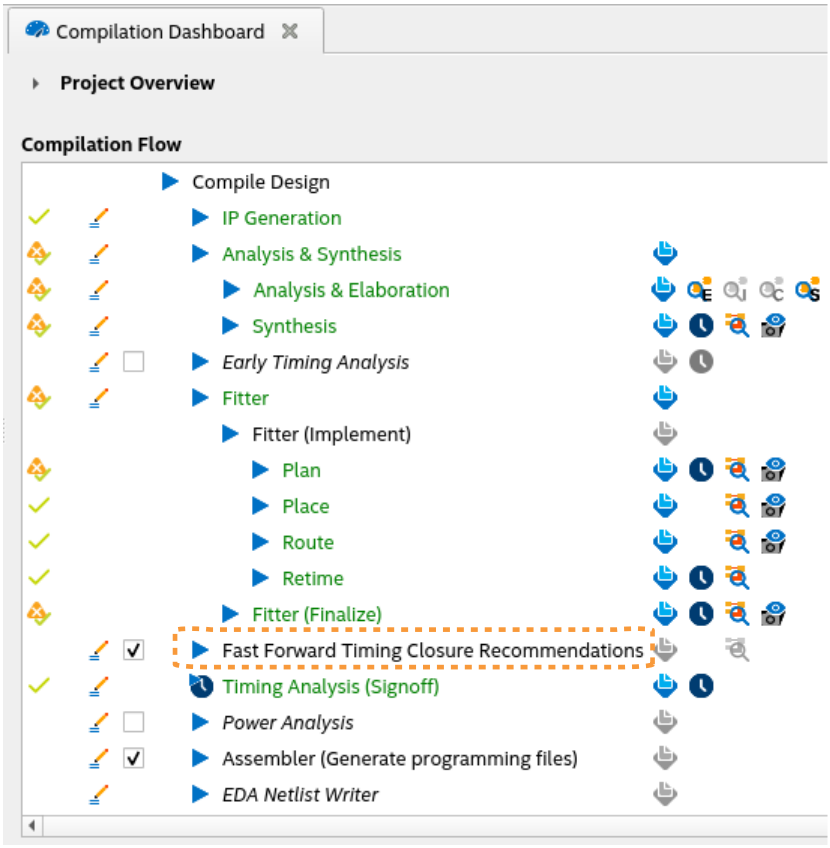
When enabled, Fast Forward compile runs on the entire design hierarchy by default. Optionally, you can specify the **Enable Hyper-Retimer Fast Forward Hierarchy analysis during compilation** assignment to include or exclude specific design subhierarchies and instances during Fast Forward compile. This technique allows you to focus Fast Forward reporting and optimization efforts on only specific areas of the design. Fast Forward compilation by hierarchy generates the same reports as Fast Forward compilation of the entire hierarchy.

Follow these steps to include or exclude specific design subhierarchies and instances during Fast Forward compilation:

1. To enable the optional Fast Forward Compilation stage during full compilation, turn on **Fast Forward Timing Closure Recommendations** on the Compilation Dashboard, or add the following assignment to the project `.qsf`:

```
set_global_assignment -name FLOW_ENABLE_HYPER_RETIMER_FAST_FORWARD ON
```

Figure 106. Enable Fast Forward Timing Closure Recommendations



2. To exclude a specific hierarchy or entity from Fast Forward Compilation, set the **Enable Hyper-Retimer Fast Forward Hierarchy analysis during compilation** assignment to **Off** in the Assignment Editor, or add the following assignment to the project `.qsf` for each hierarchy or entity that you want to exclude:

```
set_global_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF \
    <INSTANCE MODULE NAME>
```

3. To include a specific hierarchy or entity from Fast Forward Compilation, set the **Enable Hyper-Retimer Fast Forward Hierarchy analysis during compilation** assignment to **On** in the Assignment Editor, or add the following assignment to the project `.qsf` for each hierarchy or entity that you want to include:

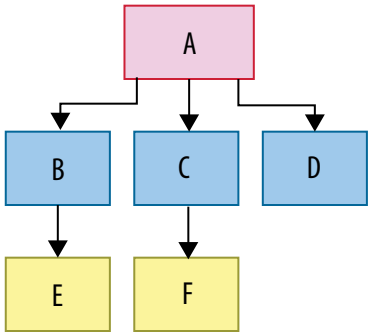
```
set_global_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY ON \
     <INSTANCE MODULE NAME>
```

4. Click the **Fast Forward Timing Closure Recommendations** stage on the Compilation Dashboard, or click **Processing ➤ Start Compilation** to run a full compilation that includes Fast Forward Compile.

You can mix ON and OFF assignments for the same instance within a single `.qsf`. If you assign mixed ON and OFF assignments to the same instance, the last assignment that appears in the `.qsf` takes precedence.

If you want to perform Fast Forward analysis for a subset of the hierarchies in your design, turn off Fast Forward analysis for all hierarchies that you want to omit from analysis. Otherwise, turn off Fast Forward analysis at the root hierarchy, and turn on Fast Forward analysis for the hierarchies that you want to analyze. The following examples show some of these assignment combinations, with respect to the *Example Design Hierarchy*.

Figure 107. Example Design Hierarchy




```
# This runs Fast Forward Compile on the entire hierarchy: A,B,C,D,E,F
# This produces the same result as if FAST_FORWARD_HIERARCHY was not set in the
QSF

set_global_assignment -name FLOW_ENABLE_HYPER_RETIMER_FAST_FORWARD ON
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY ON -to |
```

```
# Runs Fast Forward Compile on B and E only, ignores A,C,D,F

set_global_assignment -name FLOW_ENABLE_HYPER_RETIMER_FAST_FORWARD ON
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF -to |
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY ON -to B
```

```
# Runs  Fast Forward Compile on C only, ignores A,B,D,E,F

set_global_assignment -name FLOW_ENABLE_HYPER_RETIMER_FAST_FORWARD ON
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF -to |
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY ON -to C
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF -to F
```

```
# ON instance HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY takes precedence
# Fast Forward Compile runs on only C and F
# If the assignments were reversed then FFC would not run

set_global_assignment -name FLOW_ENABLE_HYPER_RETIMER_FAST_FORWARD ON
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF -to |
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF -to C
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY ON -to C
```

```
# This runs on A,B,C,F

set_global_assignment -name FLOW_ENABLE_HYPER_RETIMER_FAST_FORWARD ON
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF -to D
set_instance_assignment -name HYPER_RETIMER_FAST_FORWARD_ON_HIERARCHY OFF -to E
```

Note: Instance assignments apply to the post-fit netlist. Therefore, you may need to define design partitions. Otherwise, instance names can change during synthesis, leading to unexpected assignment results. Refer to Creating a Design Partition on page 140.

1.10.3.2. HyperFlex Settings

The **HyperFlex** settings page controls whether Fast Forward Compilation analyzes and reports results for specific logical structures in the Hyperflex architecture. You access this page by clicking **Assignments ➤ Settings ➤ HyperFlex**. Turn on **Run Fast Forward Timing Closure Recommendations during compilation** to enable Fast Forward analysis during the compilation flow by default. To access the following additional settings, click **Advanced Settings**.

Table 23. Advanced HyperFlex Settings

Option	Description
Fast Forward Compile Asynchronous Clears	Specifies how Fast Forward analysis accounts for registers with asynchronous clear signals. The options are: • **Auto**—the Compiler identifies asynchronous clears as asynchronous until they limit timing performance during Fast Forward Compilation, at which point the Compiler identifies the asynchronous clears as removed. • **Preserve**—the Compiler never assumes removal or conversion of asynchronous clears for Fast Forward analysis.
Fast Forward Compile Cut All Clock Transfers	Cuts all clock transfers in Fast Forward Compilation analysis.
Fast Forward Compile Fully Registered DSP Blocks	Specifies how Fast Forward analysis accounts for DSP blocks that limit performance. Enable this option to generate results as if all DSP blocks are fully registered.
Fast Forward Compile Fully Registered RAM Blocks	Specifies how Fast Forward analysis accounts for RAM blocks that limit performance. Enable this option to analyze the blocks as fully registered.
Fast Forward Compile Maximum Additional Pipeline Stages	Specifies the maximum number of pipeline stages that Fast Forward compilation explores.
Fast Forward Compile User Preserve Directives	Specifies how Fast Forward compilation accounts for restrictions from user-preserve directives.

1.10.4. Step 4: Review Fast Forward Results

After running Fast Forward Compilation, review the reports in the **Fast Forward Timing Closure Recommendations** folder of the Compilation Report to determine which recommendations are appropriate and practical for your design functionality and performance goals. The following section describes these reports.

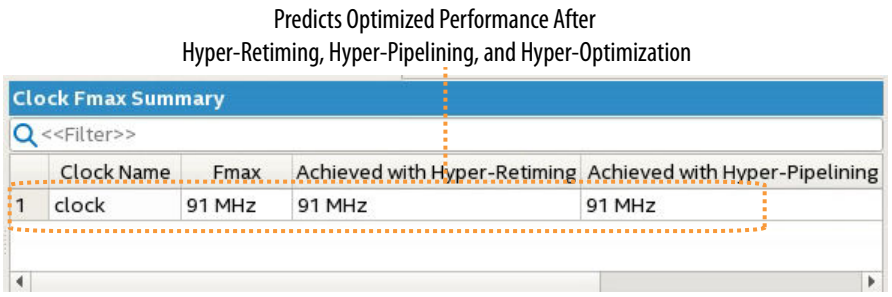
Related Information

Hyperflex Architecture High-Performance Design Handbook

1.10.4.1. Clock Fmax Summary Report

The Clock Fmax Summary in the **Fast Forward Timing Closure Recommendations** report folder reports the current f_{max} and potential performance achievable for each clock domain after Hyper-Retiming, Hyper-Pipelining, and Hyper-Optimization steps. Review the Clock Fmax Summary data to determine whether each potential performance improvement warrants further investigation and potential optimization of design RTL.

Figure 108. Current and Potential Performance in Clock Fmax Summary



Related Information

Viewing Clocks in the Timing Analyzer
in Hyperflex Architecture High-Performance Design Handbook

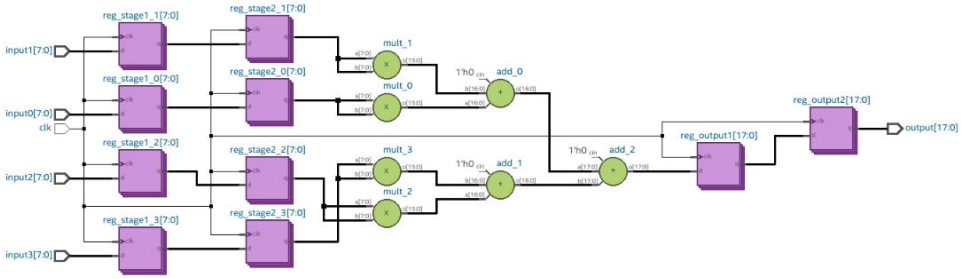
1.10.4.2. Fast Forward Details Report

The Fast Forward Details report recommends the design modifications necessary to achieve Fast Forward compilation performance levels. Some recommendations might be functionally impossible or impractical for your design. Consider the recommendations that you can implement in RTL to achieve similar performance improvement.

Click any optimization **Step** in the report to view the implementation details and performance calculations for that step.

To illustrate the effectiveness of Fast Forward Timing Closure recommendations in enhancing the timing of your design, consider the following simple design composed of adders, multipliers, and registers:

Figure 109. Simple Design with Adders, Multipliers, and Registers

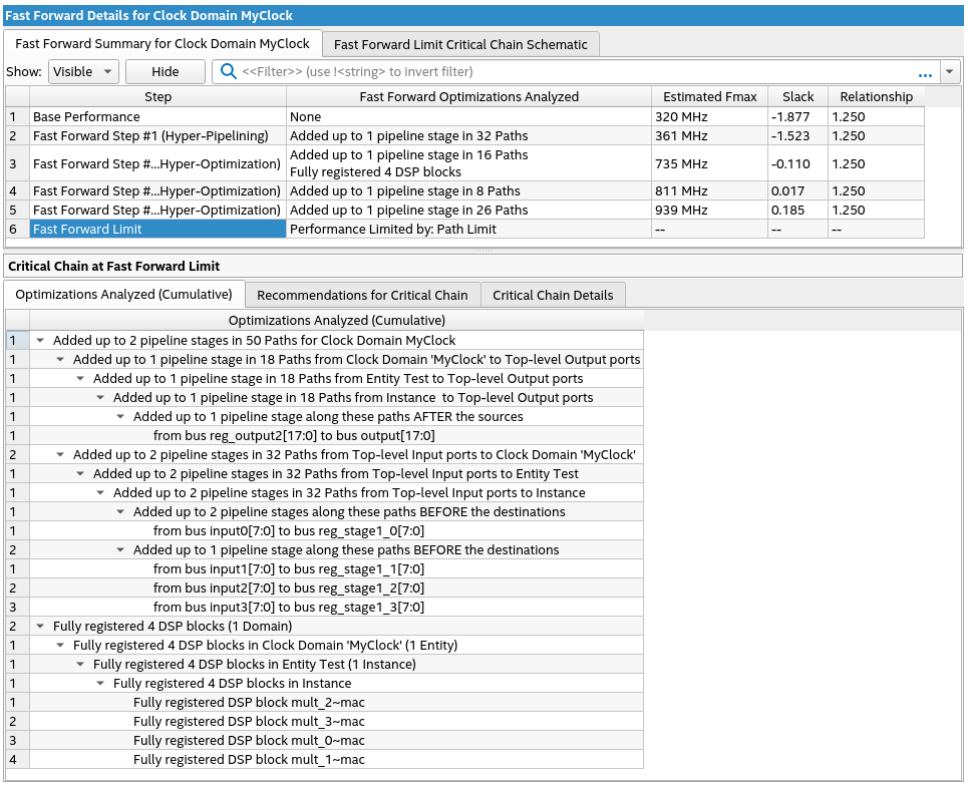


Initially, this design is required to meet a maximum frequency of 800MHz. However, due to certain constraints, the maximum frequency is currently limited to 388.5MHz.

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay	Worst-Case Operating Conditions
1	-1.324	reg_stage2...uplicate_2	mult_1~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.177	Slow vid3b 100C Model
2	-1.324	reg_stage2..._Duplicate	mult_0~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.177	Slow vid3b 100C Model
3	-1.324	reg_stage2..._Duplicate	mult_3~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.177	Slow vid3b 100C Model
4	-1.324	reg_stage2...uplicate_2	mult_2~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.177	Slow vid3b 100C Model
5	-1.301	reg_stage2...uplicate_1	mult_1~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.181	Slow vid3b 100C Model
6	-1.301	reg_stage2..._Duplicate	mult_0~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.181	Slow vid3b 100C Model
7	-1.301	reg_stage2..._Duplicate	mult_3~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.181	Slow vid3b 100C Model
8	-1.301	reg_stage2...uplicate_1	mult_2~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.181	Slow vid3b 100C Model
9	-1.298	reg_stage2..._Duplicate	mult_1~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.175	Slow vid3b 100C Model
10	-1.298	reg_stage2...uplicate_4	mult_0~mac~OUTPUT_REG	MyClock	MyClock	1.250	-0.046	0.175	Slow vid3b 100C Model

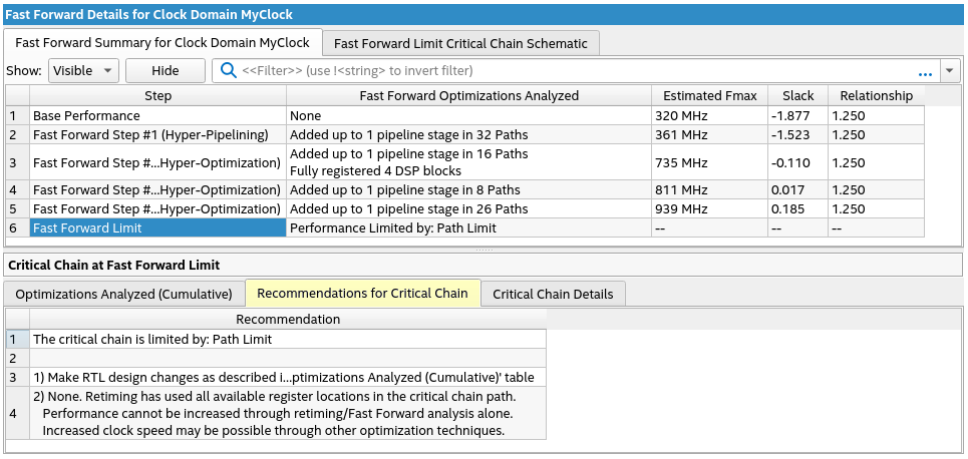
Upon executing the Fast Forward Timing Closure recommendations, proceed to review the Fast Forward Details Report. Within this report, you can find a breakdown of the optimization steps aimed at improving timing, as shown in the following image:

Figure 110. Fast-Forward Details Report



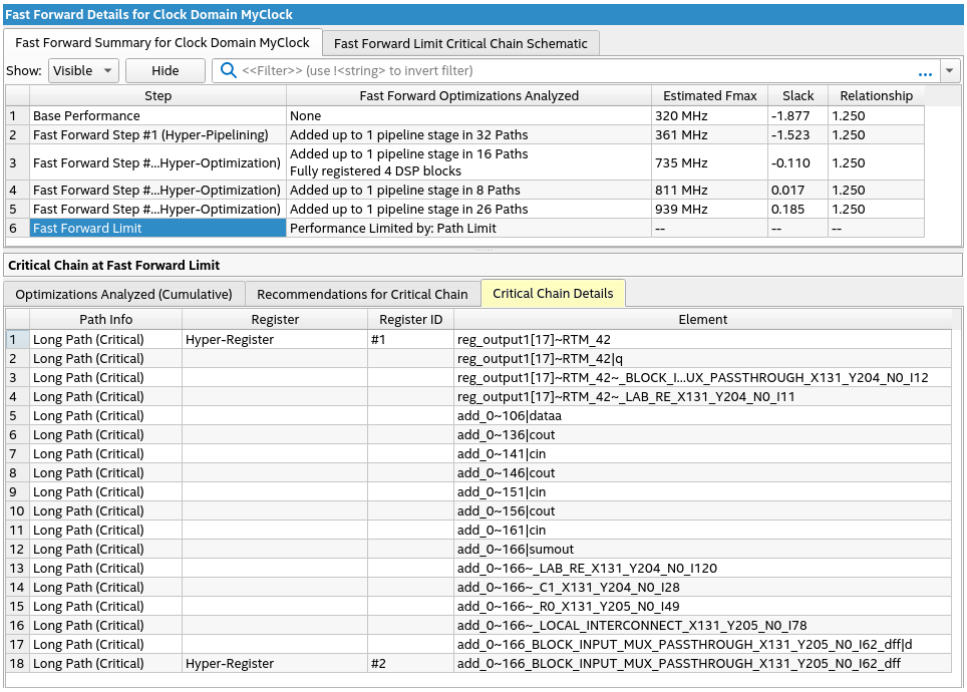
The **Optimizations Analyzed** tab shows two primary recommendations: adding pipeline stages in 50 paths and fully registering four DSP blocks. View the **Recommendations for Critical Chain** tab for specific guidance on implementing these optimizations at the RTL level.

Figure 111. Recommendations for Critical Chain



Additionally, explore the **Critical Chain Details** tab for a comprehensive description of the elements constituting the critical chain.

Figure 112. Critical Chain Details



Refer to the following table for further details on each field:

Table 24. Fast Forward Details Report Data

Report Field	Description
Step	Displays the pre-optimized Base Performance f_{MAX}, the recommended Fast Forward optimization steps, and the Fast Forward Limit critical path that prevents further optimization.
Fast Forward Optimizations Analyzed	Summarizes the optimizations necessary to implement each optimization step.
Estimated Fmax	Specifies the potential f_{MAX} performance if you implement all Fast Forward optimization steps.
Optimizations Analyzed For Fast Forward Step	Lists design recommendations hierarchically for the selected **Step**. Click the text to expand the report and view the clock domain, the affected module, and the bus and bits that require modification.
Optimizations Analyzed (Cumulative)	Accumulated list of all design changes necessary to reach the selected **Step**.
Critical Chain at Fast Forward Limit	Displays information about any path that continues to limit Hyper-Retiming even after application of all Fast Forward steps. The critical chain is any path that limits further Hyper-Retiming. Click the **Fast Forward Limit** step to display this field.
Recommendations for Critical Chain	Lists register timing path associated with the retiming limitations. Right-click any path to **Locate Critical Chain in Fast Forward Viewer**.

For a deeper analysis of the critical chains' structure, utilize the Fast Forward Viewer's cross-probing feature, as shown in the following image:

Figure 113. Locate Critical Chain in Fast Forward Viewer

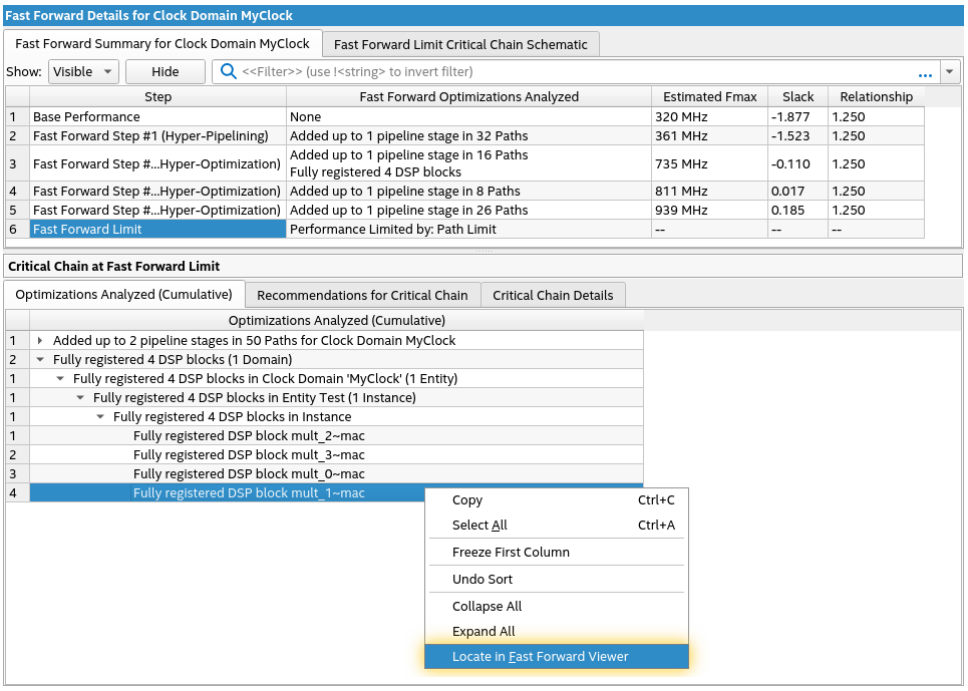
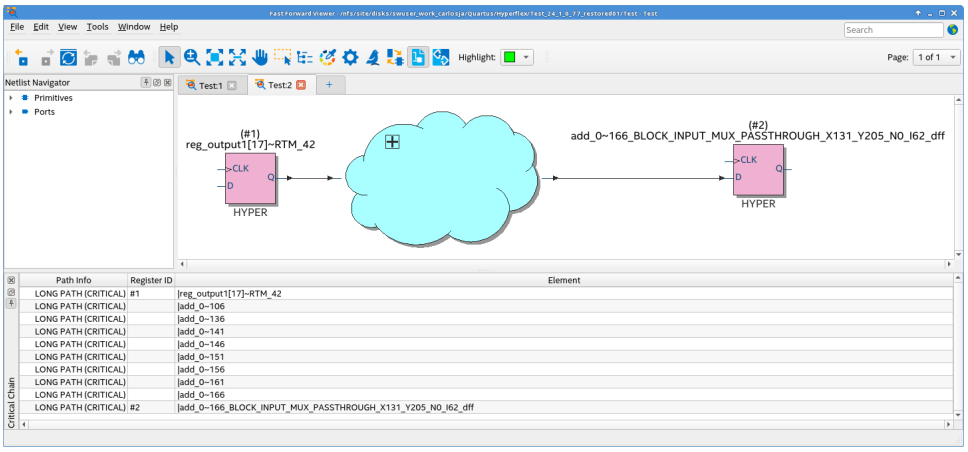
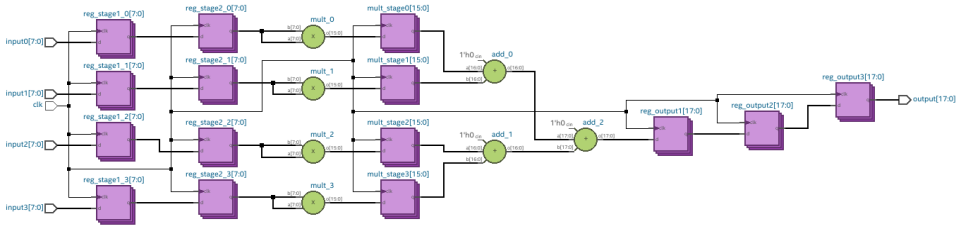


Figure 114. Fast Forward Viewer Shows Predictive Results



Based on the recommendations of the Fast Forward Details report, for this example, implement additional registers in the DSP blocks and an additional pipeline stage between `reg_output_2` and `output`. When you implement these modifications, the updated design takes the following form:

Figure 115. Design Modification



By repeating the compilation process and conducting a thorough analysis using the Timing Analyzer, you can observe tangible improvements in the timing performance of your designs.

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay	Worst-Case Operating Conditions
1	0.174	add_0~26_...0_I20_dff	reg_output3[12]	MyClock	MyClock	1.250	-0.070	1.019	Slow vid3b 100C Model
2	0.178	add_0~6_E...N0_I1_dff	reg_output3[12]	MyClock	MyClock	1.250	-0.070	1.015	Slow vid3b 100C Model
3	0.180	add_0~11_...0_I13_dff	reg_output3[12]	MyClock	MyClock	1.250	-0.070	1.013	Slow vid3b 100C Model
4	0.184	add_0~26_...0_I20_dff	reg_output3[10]	MyClock	MyClock	1.250	-0.062	1.029	Slow vid3b 100C Model
5	0.188	add_0~6_E...N0_I1_dff	reg_output3[10]	MyClock	MyClock	1.250	-0.062	1.025	Slow vid3b 100C Model
6	0.190	add_0~11_...0_I13_dff	reg_output3[10]	MyClock	MyClock	1.250	-0.062	1.023	Slow vid3b 100C Model
7	0.192	add_0~1_E...N0_I0_dff	reg_output3[12]	MyClock	MyClock	1.250	-0.070	1.001	Slow vid3b 100C Model
8	0.195	add_0~56_...rtm_bwd_1	reg_output3[12]	MyClock	MyClock	1.250	-0.073	0.995	Slow vid3b 100C Model
9	0.200	add_0~21_...0_I21_dff	reg_output3[12]	MyClock	MyClock	1.250	-0.070	0.993	Slow vid3b 100C Model
10	0.202	add_0~1_E...N0_I0_dff	reg_output3[10]	MyClock	MyClock	1.250	-0.062	1.011	Slow vid3b 100C Model

Although some of the Fast Forward Timing Closure recommendations may present challenges in practical implementation within complex designs, they nonetheless offer valuable insights and opportunities for analysis. These recommendations serve as a valuable addition to your design optimization, providing a fresh perspective and uncovering potential ways for enhancing timing performance.

Related Information

Interpreting Critical Chain Reports
in Hyperflex Architecture High-Performance Design Handbook

1.10.5. Step 5: Implement Fast Forward Recommendations

Implement the Fast Forward timing closure recommendations in your design RTL and rerun synthesis and the **Retime** stage to perform Hyper-Retiming and realize the predictive performance gains. The amount and type of changes that you implement depends on your performance goals. For example, if you can achieve the target f_{MAX} with simple asynchronous clear removal or conversion, you can stop design optimization after making those changes. For more information, refer to Retiming Restrictions and Workarounds on page 130.

1. Implement one or more Fast Forward recommendations in your design RTL, such as any of the following techniques:

- Remove limitations of control logic, such as long feedback loops and state machines.
- Restructure logic to use functionally equivalent feed-forward or pre-compute paths, rather than long combinatorial feedback paths.
- Reduce the delay of 'Long Paths' in the chain. Use standard timing closure techniques to reduce delay. Excessive combinational logic, sub-optimal placement, and routing congestion cause delay on paths.
- Insert more pipeline stages in 'Long Paths' in the chain. Long paths have the most delay between registers in the critical chain.
- Increase the delay (or add pipeline stages to 'Short Paths' in the chain).
- Explore performance and implement the RTL changes to your code until you reach the desired performance target.

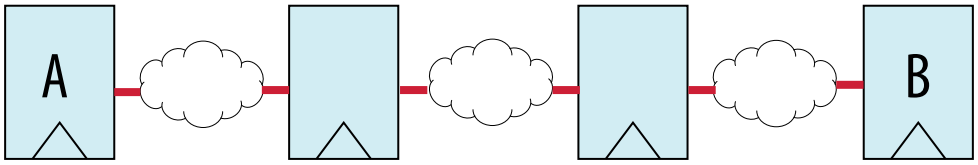
2. Implement your RTL changes and perform Hyper-Retiming by re-running the **Retime** stage on the Compilation Dashboard (which also reruns prerequisite synthesis and fitting stages).

1.10.6. Retiming Restrictions and Workarounds

The Compiler identifies the register chains in your design that limit further optimization through Hyper-Retiming. The Compiler refers to these related register-to-register paths as a critical chain. The f_{MAX} of the critical chain and its associated clock domain is limited by the average delay of a register-to-register path, and quantization delays of indivisible circuit elements like routing wires. There are a variety of situations that cause retiming restrictions. Retiming restrictions exist because of hardware characteristics, software behavior, or are inherent to the design. The **Retiming Limit Details** report the limiting reasons preventing further retiming, and the registers and combinational nodes that comprise the chain. The Fast Forward recommendations list the steps you can take to remove critical chains and enable additional register retiming.

In the diagram of a simple critical chain that follows, the red line represents the same critical chain. Timing restrictions prevent register A from retiming forward. Timing restrictions also prevent register B from retiming backwards. A loop occurs when register A and register B are the same register.

Figure 116. Sample Critical Chain



Particular registers in critical chains can limit performance for many other reasons. The Compiler classifies the following types of reasons that limit further optimization by retiming:

- **Insufficient Registers**—indicates insufficient quantity of registers at either end of the chain for retiming. Adding more registers can improve performance.
- **Short Path/Long Path**—indicates that the critical chain has dependent paths with conflicting characteristics. For example, one path improves performance with more registers, and another path has no place for additional hyper-registers.
- **Path Limit**—indicates that there are no further Hyper-Register locations available on the critical path, or the design reached a performance limit of the current place and route.
- **Loops**—indicates a feedback path in a circuit. When the critical chain includes a feedback loop, retiming cannot change the number of registers in the loop without changing functionality. The Compiler can retime around the loop without changing functionality. However, the Compiler cannot place additional registers in the loop.

After understanding why a particular critical chain limits your design's performance, you can then make RTL changes to eliminate that bottleneck and increase performance.

Table 25. Hyper-Register Support for Various Design Conditions

Design Condition	Hyper-Register Support
Initial conditions that cannot be preserved	Hyper-Registers do have initial condition support. However, you cannot perform some retiming operations while preserving the initial condition stage of all registers (that is, the merging and duplicating of Hyper-Registers). If this condition occurs in the design, the Fitter does not retime those registers. This retiming limit ensures that the register retiming does not affect design functionality.
Register has an asynchronous clear	Hyper-Registers support only data and clock inputs. Hyper-Registers do not have control signals such as asynchronous clears, presets, or enables. The Fitter cannot retime any register that has an asynchronous clear. Use asynchronous clears only when necessary, such as state machines or control logic. Often, you can avoid or remove asynchronous clears from large parts of a datapath.
Register drives an asynchronous signal	This design condition is inherent to any design that uses asynchronous resets. Focus on reducing the number of registers that are reset with an asynchronous clear.
Register has don't touch or preserve attributes	The Compiler does not retime registers with these attributes. If you use the `preserve` attribute to manage register duplication for high fan-out signals, try removing the `preserve` attribute. The Compiler may be able to retime the high fan-out register along each of the routing paths to its destinations. Alternatively, use the `dont_merge` attribute. The Compiler retimes registers in ALMs, DDIOs, single port RAMs, and DSP blocks.
Register is a clock source	This design condition is uncommon, especially for performance-critical parts of a design. If this retiming restriction prevents you from achieving the required performance, consider whether a PLL can generate the clock, rather than a register.
Register is a partition boundary	This condition is inherent to any design that uses design partitions. If this retiming restriction prevents you from achieving the required performance, add additional registers inside the partition boundary for Hyper-Retiming.
Register is a block type modified by an ECO operation	This restriction is uncommon. Avoid the restriction by making the functional change in the design source and recompiling, rather than performing an ECO.

continued...

Design Condition	Hyper-Register Support
Register location is an unknown block	This restriction is uncommon. You can often work around this condition by adding extra registers adjacent to the specified block type.
Register is described in the RTL as a latch	Hyper-Registers cannot implement latches. The Compiler infers latches because of RTL coding issues, such as incomplete assignments. If you do not intend to implement a latch, change the RTL.
Register location is at an I/O boundary	All designs contain I/O, but you can add additional pipeline stages next to the I/O boundary for Hyper-Retiming.
Combinational node is fed by a special source	This condition is uncommon, especially for performance-critical parts of a design.
Register is driven by a locally routed clock	Only the dedicated clock network clocks Hyper-Registers. Using the routing fabric to distribute clock signals is uncommon, especially for performance-critical parts of a design. Consider implementing a small clock region instead.
Register is a timing exception end-point	The Compiler does not retime registers that are sources or destinations of `.sdc` constraints.
Register with inverted input or output	This condition is uncommon.
Register is part of a synchronizer chain	The Fitter optimizes synchronizer chains to increase the mean time between failure (MTBF), and the Compiler does not retime registers that are detected or marked as part of a synchronizer chain. Add more pipeline stages at the clock domain boundary adjacent to the synchronizer chain to provide flexibility for the retiming. Alternatively, you can reduce the detection number for that particular synchronizer chain **Synchronization Register Chain Length** (default is 3). In some cases a synchronizer chain isn't necessary, and shouldn't be inferred.
Register with multiple period requirements for paths that start or end at the register (cross-clock boundary)	This situation occurs at any cross-clock boundary, where a register latches data on a clock at one frequency, and fans out to registers running at another frequency. The Compiler does not retime registers at cross-clock boundaries. Consider adding additional pipeline stages at one side of the clock domain boundary, or the other, to provide flexibility for retiming.

1.11. Full Compilation Flow

Use these steps to run a full compilation of an Quartus Prime project. A full compilation includes IP Generation, Analysis & Elaboration, Synthesis, Early Timing Analysis, Fitter, Timing Analyzer, and any optional Compiler modules you enable.

1. Before running a full compilation, specify any of the following project settings:
   - To specify the target FPGA device or development kit, click **Assignments ➤ Device**.
   - To specify device and pin options for the target FPGA device, click **Assignments ➤ Device ➤ Device and Pin Options**.
   - To specify options that affect compilation processing time and netlist preservation, click **Assignments ➤ Settings ➤ Compilation Process Settings**.

- To specify the Compiler's high-level optimization strategy, click **Assignments ➤ Settings ➤ Compiler Settings**. Specify a **Balanced** strategy, or optimize for **Performance**, **Area**, **Routability**, **Power**, or **Compile Time**. The Compiler targets the optimization goal you specify. Compiler Optimization Modes on page 149 describes these options in detail.

  For projects with a long compilation time, consider running full compilations with temporarily modified compiler optimization strategies without changing the project compiler settings. For details, see Full Compilation Flow with Temporary Optimization Mode on page 133.
- To specify synthesis algorithm and other **Advanced Settings** for synthesis and fitting, click **Assignments ➤ Settings ➤ Compiler Settings**. Turn on **Enable Intermediate Fitter Snapshots** to preserve the planned, placed, routed, and retimed snapshots by default during full compilation.
- To specify required timing conditions for proper operation of your design, click **Tools ➤ Timing Analyzer**.

2. To run full compilation, click **Processing ➤ Start Compilation**.

   Note: • To save processing time, the Compiler only preserves the planned, placed, routed, and retimed snapshots during full compilation if you turn on **Enable Intermediate Fitter Snapshots** (**Assignments ➤ Settings ➤ Compiler Settings**).

Related Information

- Full Compilation Flow with Temporary Optimization Mode on page 133
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
- Intel Quartus Prime Pro Edition User Guide: Programmer
  For complete information about generating device programming files with the Assembler and running the Quartus Prime Programmer.

1.11.1. Full Compilation Flow with Temporary Optimization Mode

You can run a full compilation flow that temporarily overrides the compiler optimization mode set in the Quartus settings file (`.qsf`) for your design. The optimization mode set for your project in your settings file does not change and remains the default compilation strategy for your project.

Overriding the compiler optimization mode set in your project can be helpful when your project has long compile times and you want to quickly produce a bitstream for on-chip testing.

To run a full compilation with a temporary optimization mode, select a compilation mode from the **Processing ➤ Start Optimization Mode Compilation** menu in the Quartus Prime Pro Edition GUI.

The following temporary optimization mode compilation flows are available in the **Start Optimization Mode Compilation** menu:

- **Start Aggressive Compile Time Optimization Mode Compilation**

  In this optimization mode, the Compiler reduces its performance optimization efforts and performs minimal reporting to provide a shorter compilation time.
- **Start Fast Functional Test Optimization Mode Compilation**

  In this optimization mode, the Compiler minimizes its setup-timing optimization efforts to provide an even shorter compilation time.

The selected optimization mode is enabled only for the duration of the compilation. After the compilation completes, Quartus Prime returns to the compilation optimization strategy that is set in the project settings.

Tip: You can customize the Quartus Prime Processing toolbar to include buttons for any of these temporary optimization mode compilations.

Important: With these optimization modes, the clocks in the resulting compilation might not meet setup. You might need to slow down the clocks on your design, such as by using PLL ECOs post-compile, before generating the bitstream.

Related Information

- Compiler Optimization Modes on page 149
- Temporarily Overriding the Compiler Optimization Mode for a Compilation in *Intel Quartus Prime Pro Edition User Guide: Scripting*
- Using the ECO Compilation Flow

1.12. Compilation Monitoring Mode

You can compile your design using the command-line interface (CLI) and the interactive Quartus Prime Pro Edition GUI.

- The CLI is generally helpful when your design is part of a larger script-based environment or the compilation is expected to take a significant amount of time. A log file is produced that you can examine at a later stage. However, it is difficult to examine and scroll through the textual output generated during a very long compilation. In such cases, use the Monitoring Mode.
- The GUI presents the entire flow of a design along with tools and utilities for design, configuration, and debugging. The entire design is assumed to be contained in a single project directory.

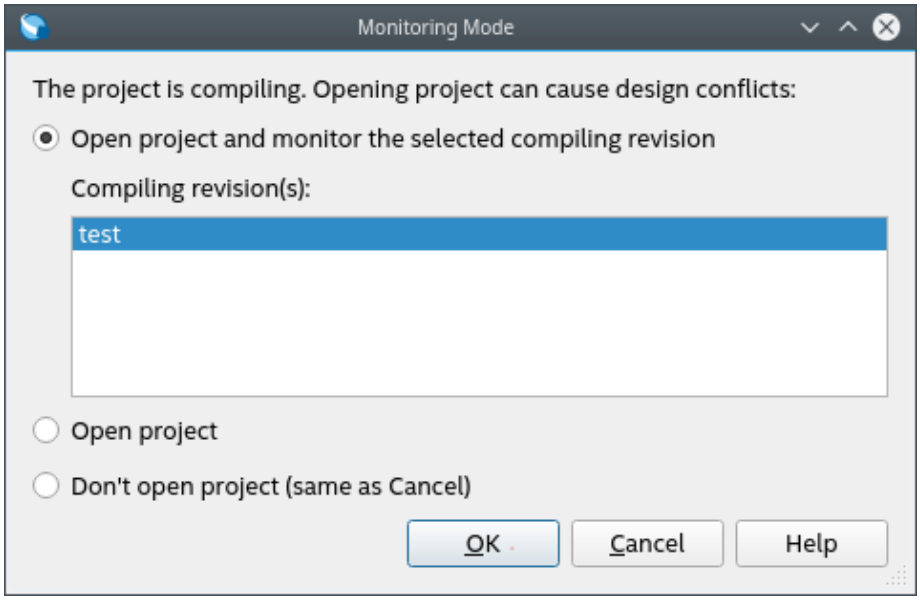
Monitoring Mode

With the "Monitoring Mode," you can bring up the Quartus Prime Pro Edition GUI, connect to a project whose compilation has already been initiated on the CLI, and monitor the execution of the compilation in the GUI. It helps examine compilation-related messages in the interactive message window of the GUI, and in general, interact with the GUI in the same way as you would in the CLI. You can also view reports, view messages, and cross-probe to source files.

To use the Monitoring Mode, perform these steps:

1. Start compiling your project through the CLI. For more information, refer to Compilation with `quartus_sh --flow` in the *Quartus Prime Pro Edition Scripting User Guide*.
2. Launch the Quartus Prime Pro Edition GUI.
3. Open the same project in the GUI that you are already compiling in the CLI. The **Monitor Mode** dialog appears, as shown in the following image:

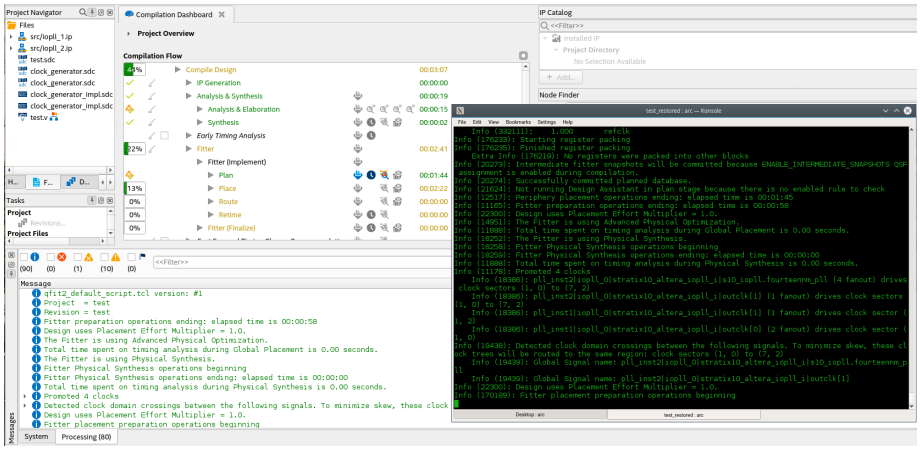
Figure 117. Monitoring Mode



4. Select **Open project and monitor the selected compiling revision** and click **OK**. The Compilation Dashboard appears, displaying the status of the compilation, as shown in the following image:

   Note: Selecting **Open project** opens the project in the non-monitoring mode, but doing so might lead to design conflicts, where the underlying database might change unexpectedly and get corrupted. Selecting **Don't open project** cancels launching the project in the GUI.

Figure 118. Monitoring the Compilation



Once the compilation ends, the **Monitoring Mode Ended** message appears on the GUI, indicating you can continue in the GUI and perform other operations on your project.

Guidelines for Using the Monitoring Mode

Consider the following when using the monitoring mode:

- You can start and exit the Quartus Prime Pro Edition GUI in monitoring mode as often as you like without affecting the progress of the compilation underway.
- If you are compiling multiple revisions of a project simultaneously, you can choose which revision compilation to monitor.
- You cannot stop a monitored compilation in the GUI. You must stop it from the CLI.
- After a monitored compilation finishes, the Quartus Prime Pro Edition GUI automatically switches out of monitoring mode. However, if a new command-line compilation for the same project and revision starts, you are prompted to reenter the monitoring mode.
- If you are currently compiling a project and revision from the CLI and open that design in the GUI without choosing the monitoring mode, you might cause design conflicts and corrupt the compilation database.
- For a project being compiled, the project shown when the GUI is not in the monitoring mode might not reflect the most recent state of the project.
- If you launch a new compilation of the project, the new compilation can corrupt the command-line compilation.

Related Information

Compilation with quartus_sh --flow in *Intel Quartus Prime Pro Edition User Guide: Scripting*

1.13. Exporting Compilation Results

The Quartus Prime Compiler writes the results to a set of database files. You can run a command to export the compilation results database as a single Quartus Database File (.qdb).

After running design compilation, the exported `.qdb` file contains the data to reproduce similar compilation results in another project, or in a later software version. You can export your project's compilation results database for import to another project or migration to a later Quartus Prime software version.

You can export the `.qdb` for your entire project or for a design partition that you define in your project. When migrating the database for an entire project, you can export the compilation database in a *version-compatible* format to ensure compatibility for import to a later software version. Although you cannot directly read the contents of the `.qdb` file after export, you can view attributes of the database file in the Quartus Database File Viewer.

Table 26. Exporting Compilation Results

To Export Compilation Results For	Method	Description
Complete Design	Click **Project ➤ Export Design**	Saves compilation results for the current project revision in a version-compatible Quartus database file (`.qdb`) that you can import to another project or migrate to a later version of the Quartus Prime software. You can export the results for the synthesized or final compilation snapshot. *Note:* Not supported for Agilex 7 devices.
Design Partition	Click **Project ➤ Export Design Partition**	Saves compilation results for a design partition as a Partition Database File (`.qdb`) that you can import to another project using the same version of the Quartus Prime software. You can export the results for the synthesized or final compilation snapshot.

Related Information

- Archiving Projects
- Creating Database-Only Archives

1.13.1. Exporting a Version-Compatible Compilation Database

You can export a project compilation database to a format that ensures version-compatibility with a later version of the Quartus Prime software. The Quartus Prime Pro Edition software version supports export of version-compatible databases for the following software versions and devices:

Table 27. Version-Compatible Compilation Database Support

The first table column indicates the first version to support version-compatible compilation database export for the specified devices.

Note:

- Database import supports two major versions back. For example, a database that you export from version 19.3, you can then import using version 19.3, 20.1, and 20.3. However, you cannot import version 19.3 to 21.1.
- You can export from any version that follows a supported version, if the version still supports the devices.

First Version with 'Export Design' Support	Stratix 10 and Devices	Arria 10 and Cyclone 10 GX Devices
18.0	No Support.	Supports all devices.
18.1	• 1SG250L • 1SG280H_S2 • 1SG280L • 1SG280L_S3 • 1SX250L • 1SX280L • 1SX280L_S3	Supports all devices.
19.1	• 1SM16BH • 1SM21BH • 1SM16CH • 1SM21CH • 1SM21KH • 1SM16KH • 1SM21LH • 1SM16LH	Supports all devices.
19.3	• 1SG10MH_U1 • 1SG10MH_U2 • 1ST250E • 1ST280E • 1SM16E • 1SM21E • 1ST165E • 1ST210E • 1SG166H • 1SG211H	Supports all devices.
20.1	• 1SD280P • 1ST040E • 1ST085E • 1ST110E	Supports all devices.
20.3	• 1SD21BP • 1SG040H • 1SX040H	Supports all devices.
20.4	• 1SN21BH • 1SN21CE	Supports all devices.

1. In the Quartus Prime software, open the project that you want to export.
2. Generate synthesis or final compilation results by running one of the following commands:

- Click **Processing ➤ Start ➤ Start Analysis & Synthesis** to generate synthesized compilation results.
- Click **Processing ➤ Start Compilation** to generate final compilation results.

3. Click **Project ➤ Export Design**. Select the **synthesized** or **final Snapshot**.

Figure 119. Export Design Dialog Box

4. Specify a name for the **Quartus Database File** to contain the exported results, and click **OK**.
5. To include the exported design's settings and constraint files, copy the `.qsf` and `.sdc` files to the import project directory.

1.13.2. Importing a Version-Compatible Compilation Database

Follow these steps to import a project compilation database into a newer version of the Quartus Prime software:

Note: Designs exported from the Quartus Prime Pro Edition software versions 23.2 or earlier cannot be imported into version 23.3 due to the new DNI database.

1. Export a version-compatible compilation database for a complete design, as Exporting a Version-Compatible Compilation Database on page 137 describes.
2. In a newer version of the Quartus Prime software, open the original project. Click **Yes** if prompted to open a project created with a different software version.
3. Click **Project ➤ Import Design** and specify the **Quartus Database File**. To remove previous results, turn on **Overwrite existing project's databases**

Figure 120. Import Design Dialog Box

4. Click **OK**. When you compile the imported design, run only Compiler stages that occur after the stage the `.qdb` preserves, rather than running a full compilation. For example, if you import a version-compatible database that contains the synthesized snapshot, start compilation with the Fitter (**Processing ➤ Start ➤**

Start Fitter). If you import a version-compatible database that contains the final snapshot, start compilation with Timing Analysis (Signoff) (**Processing ➤ Start ➤ Start Timing Analysis (Signoff)**).

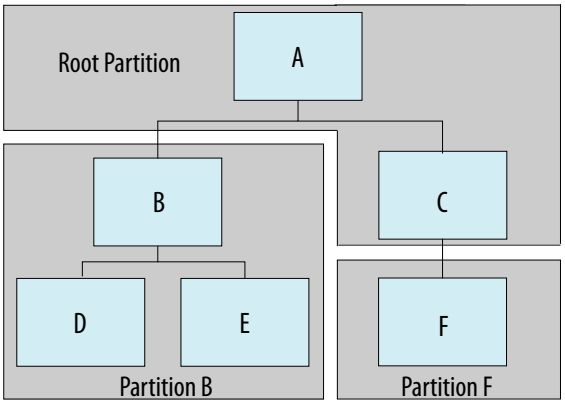
Related Information

Design Netlist Infrastructure on page 10

1.13.3. Creating a Design Partition

A design partition is a logical, named, hierarchical boundary that you can assign to an instance in your design. Defining a design partition allows you to optimize and lock down the compilation results for individual blocks. You can then optionally export the compilation results of a design partition for reuse in another context, such as reuse in another project.

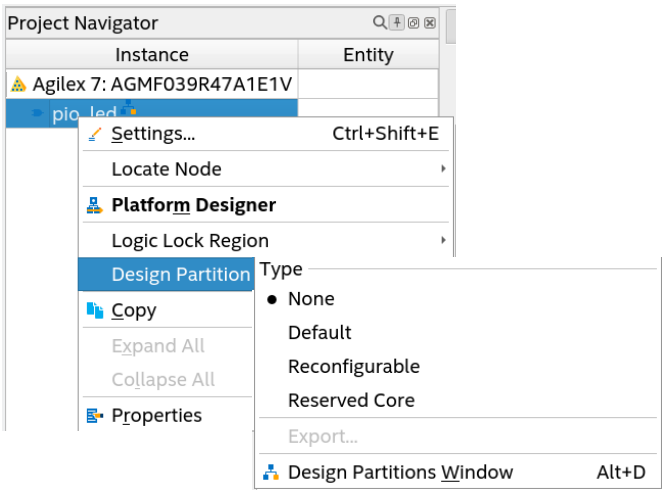
Figure 121. Design Partitions in Design Hierarchy



Follow these steps to create and modify design partitions:

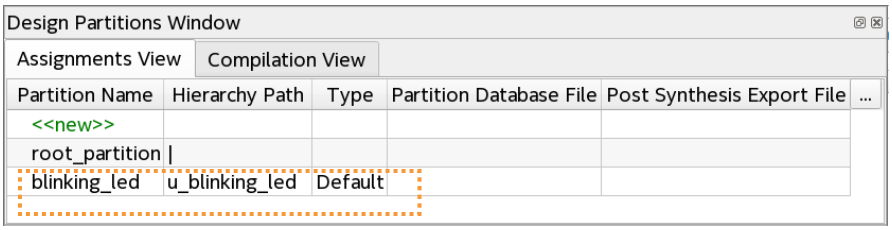
1. In the Quartus Prime software, open the project that you want to partition.
2. Generate synthesis or final compilation results by running one of the following commands:
   - Click **Processing ➤ Start ➤ Start Analysis & Synthesis** to generate synthesized compilation results.
   - Click **Processing ➤ Start Compilation** to generate final compilation results.
3. In the Project Navigator, right-click an instance in the **Hierarchy** tab, click **Design Partition ➤ Set as Design Partition**.

Figure 122. Creating a Design Partition from the Project Hierarchy



4. To view and edit all design partitions in the project, click **Assignments ➤ Design Partitions Window**.

Figure 123. Design Partitions Window



5. Specify the properties of the design partition in the Design Partitions Window. The following settings are available:

Table 28. Design Partition Settings

Option	Description
Partition Name	Specifies the partition name. Each partition name must be unique and consist of only alphanumeric characters. The Quartus Prime software automatically creates a top-level (|) "root_partition" for each project revision.
Hierarchy Path	Specifies the hierarchy path of the entity instance that you assign to the partition. You specify this value in the **Create New Partition** dialog box. The root partition hierarchy path is |.
Type	Double-click to specify one of the following partition types that control how the Compiler processes and implements the partition: • **Default**—Identifies a standard partition. The Compiler processes the partition using the associated design source files. • **Reconfigurable**—Identifies a reconfigurable partition in a partial reconfiguration flow. Specify the **Reconfigurable** type to preserve synthesis results, while allowing refit of the partition in the PR flow. • **Reserved Core**—Identifies a partition in a block-based design flow that is reserved for core development by a Consumer reusing the device periphery.
Empty	Specifies an empty partition that the Compiler skips. This setting is incompatible with the **Reserved Core** and **Partition Database File** settings for the same partition.

continued...

Option	Description
Partition Database File	Specifies a Partition Database File (`.qdb`) that the Compiler uses during compilation of the partition. You export the `.qdb` for the stage of compilation that you want to reuse (synthesized or final). Assign the `.qdb` to a partition to reuse those results in another context.
Entity Re-binding	• PR Flow—specifies the entity that replaces the default persona in each implementation revision. • Root Partition Reuse Flow —specifies the entity that replaces the reserved core logic in the consumer project.
Color	Specifies the color-coding of the partition in the Chip Planner and Design Partition Planner displays.
Post Synthesis Export File	Automatically exports post-synthesis compilation results for the partition to the specified `.qdb` file each time Analysis & Synthesis runs. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.
Post Final Export File	Automatically exports post-final compilation results for the partition to the specified `.qdb` file each time the final stage of the Fitter runs. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.

1.13.4. Exporting a Design Partition

The following steps describe export of design partitions that you create in your project.

Note: Design partitions exported from the Quartus Prime Pro Edition software versions 23.2 or earlier cannot be imported into version 23.3 or later due to the new DNI database.

When you compile a design containing design partitions, the Compiler can preserve a synthesis or final snapshot of results for each partition. You can export the synthesized or final compilation results for individual design partitions with the **Export Design Partition** dialog box.

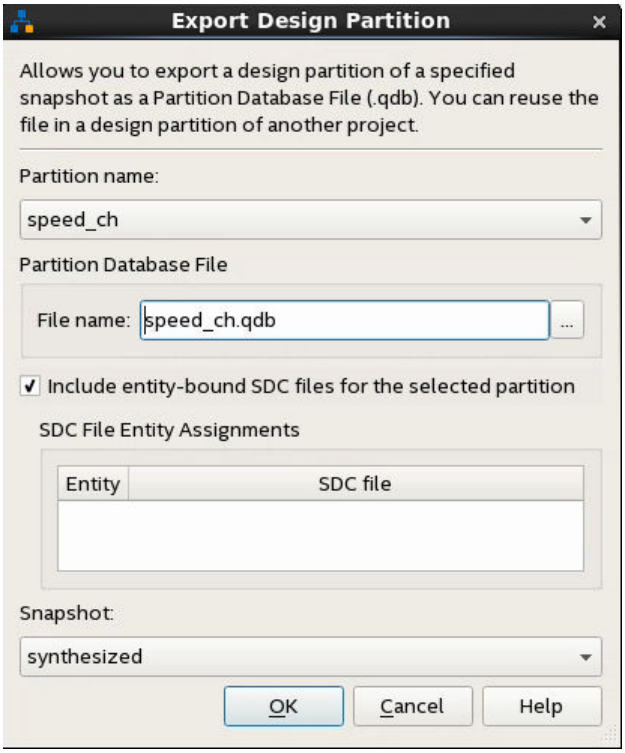
If the partition includes any entity-bound `.sdc` files, you can include those constraints in the `.qdb`. In addition, you can automate export of one or more partitions in the Design Partitions Window.

Manual Design Partition Export

Follow these steps to manually export a design partition with the **Export Design Partition** dialog box:

1. Open a project and create one or more design partitions. Creating a Design Partition on page 140 describes this process.
2. Run synthesis (**Processing ➤ Start ➤ Start Analysis & Synthesis**) or full compilation (**Processing ➤ Start Compilation**), depending on which compilation results that you want to export.
3. Click **Project ➤ Export Design Partition**, and specify one or more options in the **Export Design Partition** dialog box:

Figure 124. Export Design Partition Dialog Box



- Select the **Partition name** and the compilation **Snapshot** for export.
- To include any entity-bound `.sdc` files in the exported `.qdb`, turn on **Include entity-bound SDC files for the selected partition**.

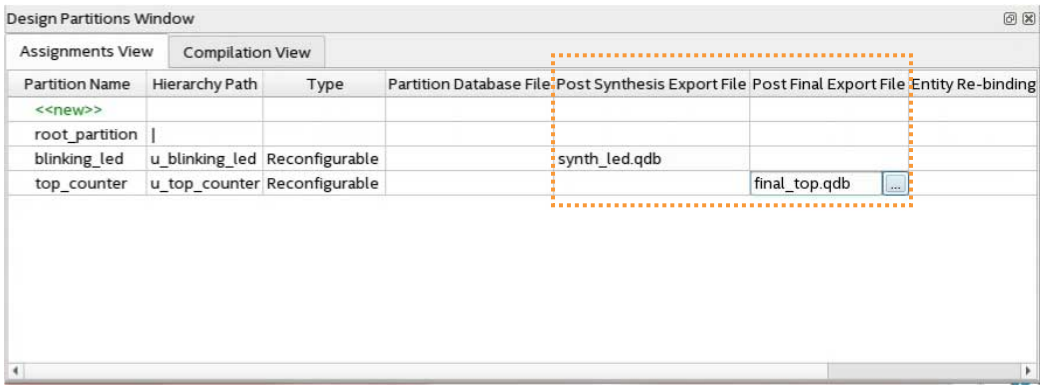
4. Click **OK**. The compilation results for the design partition exports to the file that you specify.

Automated Design Partition Export

Follow these steps to automatically export one or more design partitions following each compilation:

1. Open a project containing one or more design partitions. Creating a Design Partition on page 140 describes this process.
2. To open the Design Partitions Window, click **Assignments ➤ Design Partitions Window**.
3. To automatically export a partition with synthesis results after each time you run synthesis, specify the a `.qdb` export path and file name for the **Post Synthesis Export File** option for that partition. If you specify only a file name without a path, the file exports to the `output_files` directory after compilation.
4. To automatically export a partition with final snapshot results each time you run the Fitter, specify a `.qdb` file name for the **Post Final Export File** option for that partition. If you specify only a file name without a path, the file exports to the `output_files` directory after compilation.

Figure 125. Specifying Export File in Design Partitions Window



`.qsf` Equivalent Assignment:

```
set_instance_assignment -name EXPORT_PARTITION_SNAPSHOT_<FINAL|SYNTHESIZED> \
     <hpath> -to <file_name>.qdb
```

1.13.5. Reusing a Design Partition

You can reuse the compilation results of a design partition exported from another Quartus Prime project. Reuse of a design partition allows you to share a synthesized or final design block with another designer. Refer to *Intel Quartus Prime Pro Edition User Guide: Block-Based Design* for more information about reuse of design partitions.

To reuse an exported design partition in another project, you assign the exported partition `.qdb` to an appropriately configured design partition in the target project via the Design Partition Window:

1. Export a design partition with the appropriate snapshot, as Exporting a Design Partition on page 142 describes.
2. Open the target Quartus Prime project that you want to reuse the exported partition.
3. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
4. Click **Assignments ➤ Design Partitions Window**, and then create a design partition to contain the logic and compilation results of the exported `.qdb`.
5. Click the **Partition Database File** option for the new partition and select the exported `.qdb` file.

Figure 126. Partition Database File Setting in Design Partitions Window

6. Specify any other properties of the design partition in the Design Partitions Window. The Compiler uses the partition's assigned `.qdb` as the source.

1.13.6. Viewing Quartus Database File Information

Although you cannot directly read a `.qdb` file, you can view helpful attributes about the file to quickly identify its contents and suitability for use.

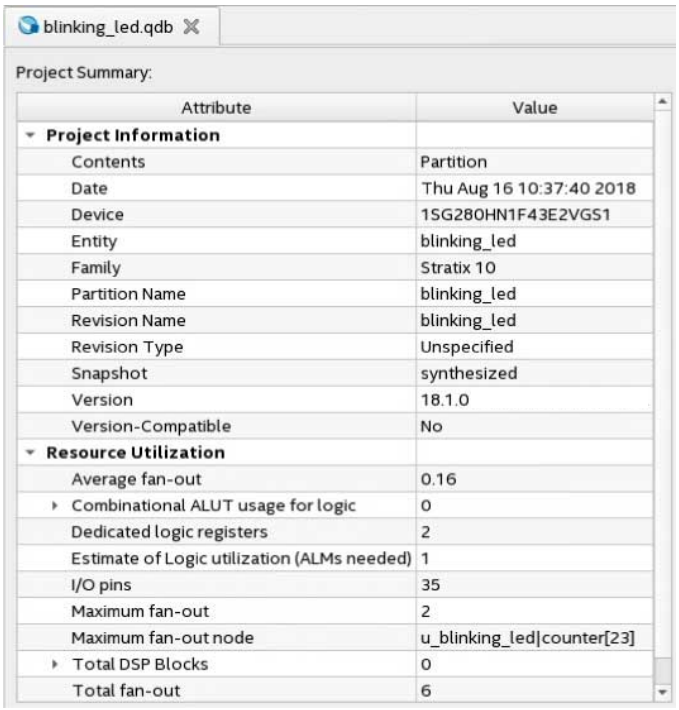
The Quartus Prime software automatically stores metadata about the project of origin when you export a Quartus Database File (`.qdb`). You can then use the Quartus Database File Viewer to display the attributes of any of these `.qdb` files.
Follow these steps to view the attributes of a `.qdb` file:

1. In the Quartus Prime software, click **File ➤ Open**, select **Design Files** for **Files of Type**, and select a `.qdb` file.
2. Click **Open**. The Quartus Database File Viewer displays project and resource utilization attributes of the `.qdb`.

   Alternatively, run the following command-line equivalent:

```
quartus_cdb --extract_metadata --file <archive_name.qdb> \
     --type quartus --dir <extraction_directory> \
     [--overwrite]
```

Figure 127. Quartus Database File Viewer



1.13.6.1. QDB File Attribute Types

The Quartus Database Viewer can display the following attributes of a `.qdb` file:

Table 29. QDB File Attributes

QDB Attribute Types	Attribute	Example
Project Information	Contents	`Partition`
	Date	`Thu Jan 23 10:56:23 2018`
	Device	`10AX016C3U19E2LG`
	Entity (if Partition)	`Counter`
	Family	`Arria 10`
	Partition Name	`root_partition`
	Revision Name	`Top`
	Revision Type	`PR_BASE`
	Snapshot	`synthesized`
	Version	`18.1.0 Pro Edition`
	Version-Compatible	`Yes`
Resource Utilization (exported for partition QDB only)	For synthesized snapshot partition lists data from the **Synthesis Resource Usage Summary** report.	`Average fan-out.16` `Dedicated logic registers:14` `Estimate of Logic utilization:1` `I/O pins:35` `Maximum fan-out:2` `Maximum fan-out node:counter[23]` `Total DSP Blocks:0` `Total fan-out:6` …
	For the final snapshot partition, lists data from the **Fitter Partition Statistics** report.	`Average fan-out:.16` `Combinational ALUTs: 16` `I/O Registers` `M20Ks` …

1.13.7. Clearing Compilation Results

You can clean the project database if you want to remove prior compilation results for all project revisions or for specific revisions. For example, you must clear previous compilation results before importing a version-compatible database to an existing project.

1. Click **Project > Clean Project**.
2. Select **All revisions** to clear the databases for all revisions of the current project, or specify a **Revision name** to clear only the revision's database you specify.
3. Click **OK**. A message indicates when the database is clean.

Figure 128. Clean Project Dialog Box Cleans the Project Database

1.14. Integrating Other EDA Tools

You can optionally integrate supported EDA synthesis, netlist partitioning, simulation, and signal integrity verification tools into the Quartus Prime design flow.

The Quartus Prime software supports input netlist files from supported EDA synthesis tools. The Compiler's EDA Netlist Writer module (`quartus_eda`) can automatically generate output files for processing in other EDA tools. The EDA Netlist Writer runs optionally as part of a full compilation, or you can run EDA Netlist Writer separately from the GUI or at the command line. The following functions are available to simplify EDA tool integration:

Table 30. EDA Tool Integration Functions

EDA Integration Task	EDA Integration Function
Specify settings for generation of output files for processing in other EDA tools.	Click **Assignments ➤ Settings ➤ EDA Tool Settings** to specify options for supported tools.
Generate output files for processing in other EDA tools.	Click **Processing ➤ Start ➤ Start EDA Netlist Writer** (or run `quartus_eda`) to generate files.
Compile RTL and gate-level simulation model libraries for your device, supported EDA simulators, and design language.	Click **Tools ➤ Launch Simulation Library Compiler** to compile simulation libraries easily.
Generate EDA tool-specific setup scripts to compile, elaborate, and simulate Intel FPGA IP models and simulation model library files.	Specify options for **Simulation** file output when generating Intel FPGA IP with IP parameter editor.
Generate files that allow supported EDA tools to perform netlist modifications, such as adding new modules, partitioning the netlist, and changing module connectivity.	Use the `quartus_eda –resynthesis` command to generate a Verilog Quartus Mapping File (`.vqm`) that contains a node-level (or atom) representation of the netlist in standard structural Verilog RTL.
Include files generated by other EDA design entry or synthesis tools in your project as synthesized design files.	Click **Project ➤ Add/Remove Files In Project** to add supported **Design File** files from other EDA tools.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Third-party Simulation
- Intel Quartus Prime Pro Edition User Guide: Third-party Synthesis
- Intel Quartus Prime Pro Edition User Guide: PCB Design Tools

1.14.1. Generating a VQM Netlist for other EDA Tools

The EDA Netlist Writer (`quartus_eda`) can generate a node-level netlist in Verilog Quartus Mapping File (`.vqm`) format for use in other EDA tools. You can process the `.vqm` netlist in other EDA tools to add new modules, partition the netlist, or change connectivity. After third-party tool changes, you resynthesize and compile the `.vqm` in the Quartus Prime software.

The `.vqm` format is standard structural Verilog RTL. The modules can be any Intel FPGA family-specific WYSIWYG type for core logic (such as, flip-flop, LUT, DSP, M20K). EDA Netlist Writer does not support `.vqm` for periphery modules (such as transceivers, memory interfaces, I/O, or IP including these). The RTL is a fully flattened representation of the entire design hierarchy or partition. The module names capture the original hierarchy, although some renaming can occur to legalize names. There is no truncation of the netlist module names.

To perform `.vqm` netlist partitioning in other EDA tools, define a design partition that includes only core logic elements. Generate the partition netlist as step 3 on page 149 describes. After processing the `.vqm` in third-party tools, resynthesize the `.vqm` files either independently or as a design partition. If including a black box module instantiation in the `.vqm`, make connections between existing logic in the `.vqm` and the black box. Prior to resynthesis, specify the source file (`.ip`, `.v`, or `.vqm`) for the black box in the project `.qsf`.

Table 31. VQM Netlist Generation Requirements and Limitations

Requirement or Limitation	Description
Design partitions must only include core logic.	Design partitions must include only flip-flops, LUTs, DSPs, and on-chip memory. The EDA Netlist Writer does not support `.vqm` output for periphery modules (such as transceivers, memory interfaces, I/O, or IP that includes these).
Analysis & Synthesis does not support some special characters in instance names that are legal in SystemVerilog.	Analysis & Synthesis resolves these characters by placing the standard escape character '\' to escape the special character present in the RTL. If any of the hierarchical constraints (for example, SDC timing constraints) explicitly reference such a special character, modify these characters manually.
Generate `.vqm` only for a synthesized netlist.	The post-fit netlist includes atoms, such as wire-luts, that are not appropriate for resynthesis.
Avoid module or entity name collisions	If you add a `.vqm`, generated from an RTL design file, to the same Quartus Prime project that generated the `.vqm`, beware of potential entity or module name collisions. Name collisions can occur if the original RTL file from which the `.vqm` derives, and the `.vqm` file itself, both specify the same entity or module name. When the RTL and `.vqm` files are both present in the project **Files** list, the Compiler uses the last entry in the list.
Partition assignments might not align with the original design.	`.vqm` generation flattens all logic within a partition unless you specify the `–exclude_sub_partitions` argument. Compiling a design that has assignments pertaining to a flattened partition causes an error.

To generate a `.vqm` for processing in other EDA tools, follow these steps:

1. In the Quartus Prime software, click **Processing ➤ Start ➤ Start Analysis & Synthesis** (or run `quartus_syn`) to synthesize the design netlist.
2. Create a design partition containing only core logic elements for the `.vqm`, as Creating a Design Partition on page 140 describes.
3. To generate the `.vqm` in the `resynthesis` directory, run any of the following commands at the command prompt:
   - To write out the entire design netlist to `.vqm`:
     ```
     quartus_eda --resynthesis=on <project_name>
     ```
   - To write out only a specific design partition netlist to `.vqm`:
     ```
     quartus_eda --resynthesis=on -partition=<name> <project_name>
     ```
   - To write out any sub partition as a black-box netlist to `.vqm`:
     ```
     quartus_eda --resynthesis=on –exclude_sub_partitions <project_name>
     ```
     You can also combine `–exclude_sub_partitions` with `-partition`.
4. View the resulting `.vqm` in the `resynthesis` directory, and specify the `.vqm` as input to your EDA tool.
5. After processing the `.vqm` in another EDA tool, add the `.vqm` as an Quartus Prime project design file by clicking **Project ➤ Add/Remove Files In Project**. Avoid module or entity name collisions, as the VQM Netlist Generation Requirements and Limitations table describes.
6. Run Analysis & Synthesis on the project, followed by the remaining Compiler stages.

1.15. Compiler Optimization Techniques

You can apply various optimization techniques via settings and entity assignments to achieve your design requirements during compilation. For example, you can specify options to preserve specific registers through synthesis processing, apply fractal synthesis, enable register retiming, and various other targeted Compiler optimizations.

1.15.1. Compiler Optimization Modes

To apply the compiler optimization settings, click **Assignments ➤ Settings ➤ Compiler Settings**. Alternatively, you can apply the optimization settings using the `OPTIMIZATION_MODE` QSF. For example:

```
set_global_assignment -name OPTIMIZATION_MODE "Aggressive Area"
```

The default value for `OPTIMIZATION_MODE` is "Balanced". For all other values, refer to the Optimization Modes table.

You can enable one of the following optimization modes to focus the Compiler's optimization effort. The settings compilation mode you select affects synthesis and fitting results.

To select an optimization mode, start with the **Balanced** setting. This mode is appropriate for many designs and provides an implementation balanced between optimization and compile time. If this setting does not meet your goals, you can try different optimization modes depending on your requirements.

If your design requires additional performance compared to the **Balanced** setting, use the **High performance effort** setting that enables additional timing optimizations during the fitting stage. You can achieve additional performance using the **Superior performance** setting that further enables additional timing optimizations during the Synthesis stage. However, these synthesis optimizations may result in increased logic area, negatively impacting designs with high utilization. Both settings increase compile time.

Alternatively, use the **Aggressive Area** setting to reduce logic area at the potential expense of performance. Similarly, use the **Aggressive power** setting to reduce dynamic power at the potential expense of performance.

If your design has difficulty routing successfully, the settings **Optimize netlist for routability**, **High placement routability effort**, and **High packing routability effort** offer a variety of optimizations to improve routability. Which optimizations work best is design-dependent, so try each if you encounter routability issues.

Finally, use **Aggressive Compile Time** and **Fast Functional Test** settings to reduce compile time. These settings reduce performance but may be helpful early in a design cycle when only functionality is being verified.

Table 32. Optimization Modes (Compiler Settings Page)

Optimization Mode	QSF Value	Description	Implications
Balanced (normal flow)	`Balanced`	The Compiler optimizes synthesis for balanced implementation that respects timing constraints.	The default setting that produces a balance between optimization effort and compile time.
High performance effort	`High Performance Effort`	The Compiler increases the timing optimization effort during placement and routing, and enables timing-related Physical Synthesis optimizations (per register optimization settings).	Increases compilation time for better performance compared to **Balanced** setting.
High performance with maximum placement effort	`High Performance With Maximum Placement Effort`	Enables the same Compiler optimizations as **High performance effort**, with additional placement optimization effort.	Increases compilation time for better performance compared to **High performance effort** setting.
High performance with aggressive power effort	`High Performance With Aggressive Power Effort`	Enables the same Compiler optimizations as **High performance effort**, while performing additional optimizations to reduce dynamic-power.	Increases compilation time for lower power compared to **High performance effort** setting.
Superior performance	`Superior Performance`	Enables the same Compiler optimizations as **High performance effort**, and adds more optimizations during Analysis & Synthesis to maximize design performance with a potential increase to logic area.	Increases compilation time for better performance compared to **High performance effort** setting. If design utilization is very high, this mode can cause difficulty in fitting, which can also negatively affect overall optimization quality.

continued...

Optimization Mode	QSF Value	Description	Implications
Superior performance with maximum placement effort	`Superior Performance With Maximum Placement Effort`	Enables the same Compiler optimizations as **Superior performance**, with additional placement optimization effort.	Increases compilation time for better performance compared to **Superior performance** setting.
Aggressive Area (reduces performance)	`Agressive Area`	The Compiler makes aggressive effort to reduce the device area required to implement the design at the potential expense of design performance.	Reduces performance for reduced area compared to **Balanced** setting.
High placement routability effort	`High Placement Routability Effort`	The Compiler makes high effort to route the design at the potential expense of design area, performance, and compilation time. The Compiler spends additional time reducing routing utilization, which can improve routability and also saves dynamic power.	Increases compilation time for better routability compared to **Balanced** setting.
High packing routability effort	`High Packing Routability Effort`	The Compiler makes high effort to route the design at the potential expense of design area, performance, and compilation time. The Compiler spends additional time packing registers, which can improve routability and also saves dynamic power.	Increases compilation time for better routability compared to **Balanced** setting.
Optimize netlist for routability	`Optimize Netlist for Routability`	The Compiler implements netlist modifications to increase routability at the possible expense of performance.	Increases compilation time for better routability compared to **Balanced** setting.

continued...

Optimization Mode	QSF Value	Description	Implications
Aggressive power (reduces performance)	`Agressive Power`	Makes aggressive effort to optimize synthesis for low power. The Compiler further reduces the routing usage of signals with the highest specified or estimated toggle rates, saving additional dynamic power but potentially affecting performance.	Reduces performance for lower power compared to **Balanced** setting.
Aggressive Compile Time (reduces performance)	`Aggressive Compile Time`	Especially useful during early design iterations, this mode reduces the compilation run time by 30% (on average) at the expense of design f_{MAX} of 15% (on average). Run time reduction occurs through reduced effort and fewer performance optimizations. This mode also disables some detailed reporting functions. This mode produces the fastest full-flow timing estimation with an approximate correlation to the high-effort modes.	• This mode reduces performance. • Reduced effort levels can cause no-fits, especially on highly congested designs. Mitigate this potential by either partitioning and constraining the placement of congested parts of design, or by using a high effort or routability mode • This mode may not identify the same critical paths as a full-effort compile (similar to Compiler seed-effects). • This mode disables some detailed reporting functions and enables `.qsf` settings that cannot be overridden by other `.qsf` settings.
Fast Functional Test (hold-timing optimization only)	`Fast Functional Test`	This mode produces a `.sof` bitstream file that you can use for on-board functional testing with minimal compile time. This mode further reduces compile time beyond Aggressive Compile Time mode by limiting timing optimizations to only those for hold requirements.	• Reduced effort levels can cause no-fits, especially on highly congested designs. Mitigate this potential by either partitioning and constraining the placement of congested parts of design, or by using a high effort or routability mode. Refer to the *Creating a Partition* topic in this document and the *Intel Quartus Prime Pro Edition User Guide: Design Constraints* • This mode can require clock speeds outside the lock range of the PLL Intel FPGA IP. Mitigate this effect by using the `adjust_pll` ECO command to update the PLL IP after fitting. • This mode disables some detailed reporting functions and enables `.qsf` settings that cannot be overridden by other `.qsf` settings.

Note: If you enable extended optimization modes for Design Space Explorer II by use of `.qsf` assignments, and then subsequently open the **Compiler Settings** tab for that project revision, the **Compiler Settings** tab indicates that the extended optimization mode reverts to one of the **Compiler Settings** tab **Optimization Modes**.

Related Information

- Creating a Design Partition on page 140
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
- Full Compilation Flow with Temporary Optimization Mode on page 133

1.15.2. Allow Register Retiming

The **Allow Register Retiming** option on the **Register Optimization** tab controls whether or not to globally disable retiming. When turned on, the Compiler automatically performs register retiming optimizations, moving registers through combinational logic. When turned off, the Compiler prevents any retiming optimizations on a global scale.

Optionally, assign **Allow Register Retiming** to any design entity or instance for specific portions of the design. Click **Assignments ➤ Assignment Editor** to specify entity- and instance-level assignments, or use the following syntax to make the assignment in the `.qsf` directly.

Remember: For devices that use the Hyperflex architecture (such as Agilex 7 devices), you can use the **Allow Register Retiming** optimization option alongside the Hyper-Retiming optimization.

Example 11. Disable register retiming for entity abc

```
set_global_assignment –name ALLOW_REGISTER_RETIMING ON

set_instance_assignment –name ALLOW_REGISTER_RETIMING OFF –to “abc|”

set_instance_assignment –name ALLOW_REGISTER_RETIMING ON –to “abc|def|”
```

Example 12. Disable register retiming for the whole design, except for registers in entity abc

```
set_global_assignment –name ALLOW_REGISTER_RETIMING OFF

set_instance_assignment –name ALLOW_REGISTER_RETIMING ON –to “abc|”

set_instance_assignment –name ALLOW_REGISTER_RETIMING OFF –to “abc|def|”
```

1.15.3. Automatic Gated Clock Conversion

Clock gating saves power in ASIC designs by adding more logic to a circuit to prune the clock tree. Pruning the clock tree disables portions of the circuitry so that the flip-flops are not required to switch states. When using an Quartus Prime FPGA to prototype ASIC designs, you must convert clock gates to clock enables in your design.

Table 33. Gated Clock Conversion Example

ASIC Gated Clock Example

```
module infer_enable (clk, reset, d, en, q);

input d, en, clk, reset;
output q;

wire gated_clk;
reg q;

assign gated_clk = clk & en;
always@(posedge gated_clk or reset)
 begin
               if (!reset)
                      q <= 1’b0;
              else
                      q <= d ;
              end
endmodule
```

FPGA Clock Enable Example

```
module infer_enable (clk, reset, d, en, q);

input d, en, clk, reset;
output q;

reg q;

always@(posedge clk or reset)
 begin
               if (!reset)
                      q <= 1’b0;
              else if (en)
                      q <= d;
              else
                      q <= q ;
              end
endmodule
```

Rather than manually converting gated clocks in your RTL, you can specify the **Auto Gated Clock Conversion** setting to automatically convert gated base clocks in the design to clock enables. You can apply this setting globally to all gated base clocks in the design, or to one or more specific clock signals.

Table 34. Gated Clock Conversion Settings

Setting Scope	Description
Global	Enable the **Auto Gated Clock Conversion** option at **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. Alternatively, add the global assignment to the project `.qsf`: `set_global_assignment -name SYNTH_GATED_CLOCK_CONVERSION on`
Instance-specific	Specify the **Auto Gated Clock Conversion** for one or more instances in the Assignment Editor (**Assignments ➤ Assignment Editor**). Alternatively, add the instance assignment to the project `.qsf`: `set_instance_assignment -name SYNTH_GATED_CLOCK_CONVERSION on -to clk_in`

Following design synthesis, view the results of gated clock conversion in the Gated Clock Conversion Details report. The report lists all converted and unconverted gated clocks with their base clocks. For unconverted gated clocks, the report specifies the reason the clock is not converted.

Note: Automatic gated clock conversion supports explicit RAMs (such as WYSIWYG RAMs and Intel FPGA memory IP), but does not support inferred RAMs.

Figure 129. Gated Clock Conversion Details Report

Gated Clock Conversion Details

	Gated Clock	Base Clock	Converted to Clock Enable	Reason not Converted
1	clk1	clk_in	No	Found register out1_dff fed by the gated clock tree has in-use clock enable
2	clk2	clk_in	No	Found unconvertable IO pin clk_out2 fed by the gated clock tree
3	clk3	clk_in	No	Found unsupported WYSIWYG wys_io fed by the gated clock tree
4	clk4~0	clk_in	No	Found unsupported gate clk4~0 in the gated clock tree
5	clk5	clk_in	No	Found dangerous cascaded clock from clk5~0 to clk5

1.15.4. Enable Intermediate Fitter Snapshots

To save compilation time, the Compiler does not save the planned, placed, routed, or retimed snapshots by default during full compilation.

However, you can turn on **Enable Intermediate Fitter Snapshots** (**Assignments ➤ Settings ➤ Compiler Settings**) to generate and preserve snapshots for the Plan, Place, Route, and Retime stages any time you run full compilation. You can also run any intermediate Fitter stage independently to generate the snapshot for that stage.

With the **Enable Intermediate Fitter Snapshots** option enabled, multiple snapshots are saved during the Fitter stages, allowing more flexible iteration of intermediate Fitter steps. For example, you can resume and rerun from the Place snapshot as opposed to starting from the Fitter Plan stage, saving compilation time. However, with this option enabled, you might incur extra runtime to write the intermediate snapshots during Fitter stage and disk space consumption is more in order to accommodate the additional snapshots.

Note: The **Enable Intermediate Fitter Snapshots** option is off by default. However, if **Run Fast Forward Timing Closure Recommendations during compilation** option is on, then the **Enable Intermediate Fitter Snapshots** option is forced enabled.

1.15.5. Fast Preserve Option

Enabling the **Fast Preserve** option on the **Incremental Compile** tab specifies that the Compiler can simplify a preserved partition to only interface logic.

Interface logic is logic at the partition boundary that interfaces with the rest of the design.

1.15.6. Fractal Synthesis Optimization

Fractal synthesis optimizations can be useful for deep-learning accelerators and other high-throughput, arithmetic-intensive designs that exceed all available DSP resources. For such designs, fractal synthesis optimization can achieve 20-45% area reduction.

Fractal synthesis is a set of synthesis optimizations that use FPGA resources in an optimal way for arithmetic-intensive designs. These synthesis optimizations consist of multiplier regularization and retiming, as well as continuous arithmetic packing. The optimizations target designs with large numbers of low-precision arithmetic operations (such as additions and multiplications). You can enable fractal synthesis globally or for specific multipliers, as Enabling or Disabling Fractal Synthesis on page 160 describes.

Project-Wide Fractal Synthesis Considerations

Note: Fractal synthesis optimization is most suitable for designs with deep-learning accelerators or other high-throughput, arithmetic-intensive functions that exceed all DSP resources. Enabling fractal synthesis project-wide can cause unnecessary bloat on modules that are not suitable for fractal optimizations. Consider the following factors before enabling fractal synthesis optimization project wide:

- Intel FPGA devices contain thousands of hard DSP blocks that are perfectly suited for arithmetic operations. If the total amount of arithmetic functions in your design is small, then there is no need to enable Fractal Synthesis. In such cases, all the arithmetic functions map directly into DSPs by default. Enable global Fractal Synthesis only if there are not enough DSP blocks available to implement all arithmetic components. Enable Fractal Synthesis only for modules that you do not want the Compiler to map into DSPs.
- In the current version of the Quartus Prime Pro Edition software, fractal synthesis optimizations target low-precision multiplication. Implement high-precision multipliers (where width of every operand exceeds 11 bits) using DSP blocks.
- If you enable project-wide Fractal Synthesis, the following information message number 20193 may generate during compilation:

  ```
  Applied dense packing to "<entity>". Area: 2 LABs. Logic density: 0.775.
  ```

  This information indicates the effort the Compiler is packing computational logic into a smaller number of LABs. If the design is already highly utilized, the Compiler can skip this stage.

  — Verify that the Area the message reports does not exceed 100 LABs. If the Area exceeds 100 LABs, divide fractal synthesis blocks to sub-blocks, and then assign the fractal synthesis optimizations to the sub-blocks independently.

  — Verify that the Logic density the message reports is greater than 0.75. If the logic density is less than 0.75, disable **Fractal Synthesis** for this entity because standard synthesis typically achieves better density.

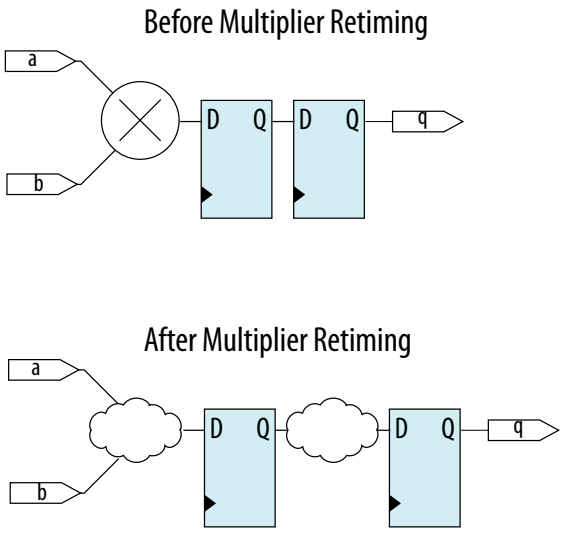
Table 35. Fractal Synthesis Area Improvement

		Area (LABs)	
Device	**Dot-product**	**Fractal Synthesis ON**	**Fractal Synthesis OFF**
Arria 10 and Cyclone 10 GX	Sum of 16 4x4sm	12	19
	Sum of 16 5x5sm	19	32
	Sum of 16 6x6sm	25	36
	Sum of 16 7x7sm	34	44
	Sum of 16 8x8sm	45	60
Stratix 10 and Agilex 7 Devices	Sum of 16 4x4sm	15	22
	Sum of 16 5x5sm	21	39
	Sum of 16 6x6sm	29	47
	Sum of 16 7x7sm	39	55
	Sum of 16 8x8sm	55	71

Multiplier Regularization and Retiming

Multiplier regularization and retiming performs inference of highly optimized soft multiplier implementations. The Compiler may apply backward retiming to two or more pipeline stages if required. When you enable fractal synthesis, the Compiler applies multiplier regularization and retiming to signed and unsigned multipliers.

Figure 130. Multiplier Retiming



Note:

- Multiplier regularization uses only logic resources, and does not use DSP blocks.
- Multiplier regularization and retiming is applied to both signed and unsigned multipliers in modules where the FRACTAL_SYNTHESIS QSF assignment is set.

Multiplier Regularization Example

The following simple, unsigned dot-product design example contains multiplication operators with 5-bit operands. These short multipliers are perfect candidates for multiplier regularization.

```
(* altera_attribute = "-name FRACTAL_SYNTHESIS ON" *)
module dot_product(
   input clk,
   input [4:0] a, b, c, d, e, f, g, h,
   output reg [11:0] out
);
reg [9:0] ab, cd, ef, gh;
reg [10:0] ab_cd, ef_gh;

always @(posedge clk)
begin
    ab <= a * b;
    cd <= c * d;
    ef <= e * f;
    gh <= g * h;
    ab_cd <= ab + cd;
    ef_gh <= ef + gh;
    out <= ab_cd + ef_gh;
end
endmodule

module top(
   input clk,
   input [4:0] a1, b1, c1, d1, e1, f1, g1, h1,
   input [4:0] a2, b2, c2, d2, e2, f2, g2, h2,
   output [11:0] out1, out2
);
dot_product core1(.clk(clk), .a(a1), .b(b1), .c(c1), .d(d1),
    .e(e1), .f(f1), .g(g1), .h(h1), .out(out1));
```

```
dot_product core2(.clk(clk), .a(a2), .b(b2), .c(c2), .d(d2),
    .e(e2), .f(f2), .g(g2), .h(h2), .out(out2));
endmodule
```

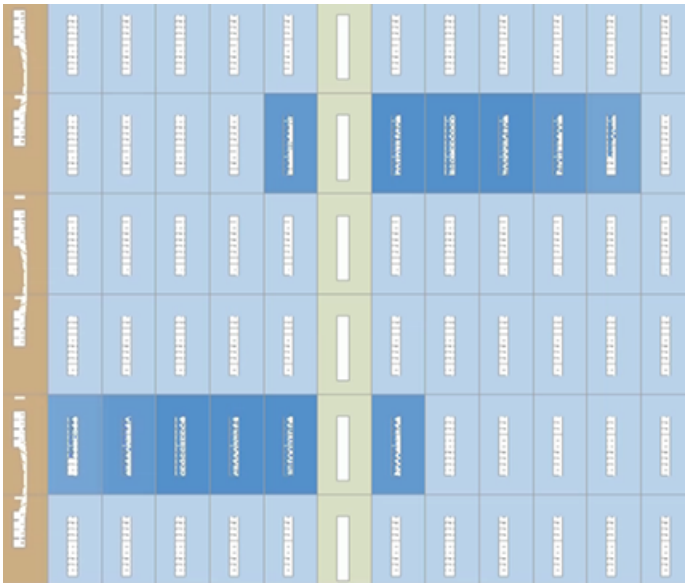
Quartus Prime synthesis prints the following messages to the console:

Figure 131. Console Messages

Message	Message ID
Inferred optimized multiplier from the following logic: "core1|mult_2"	20192
Inferred optimized multiplier from the following logic: "core1|mult_1"	20192
Inferred optimized multiplier from the following logic: "core1|mult_0"	20192
Inferred optimized multiplier from the following logic: "core2|mult_3"	20192
Inferred optimized multiplier from the following logic: "core2|mult_2"	20192
Inferred optimized multiplier from the following logic: "core2|mult_1"	20192
Inferred optimized multiplier from the following logic: "core2|mult_0"	20192
Timing-Driven Synthesis is running	286030
Applied dense packing to "core2". Area: 6 LABs. Logic density: 0.941667.	20193
Applied dense packing to "core1". Area: 6 LABs. Logic density: 0.941667.	20193

In the Chip Planner, you can observe this design having two unsigned dot-product cores. These cores are independently optimized and placed. The LAB resources are nearly 100% optimized, as the following image shows:

Figure 132. Design Placement



Signed dot-products are common for deep-learning applications. The following demonstrates an example of a signed dot-product:

```
(* altera_attribute = "-name FRACTAL_SYNTHESIS ON" *)
module dot_product(
   input signed clk,
   input signed [4:0] a, b, c, d, e, f, g, h,
   output reg signed [11:0] out
);
reg signed [9:0] ab, cd, ef, gh;
reg signed [10:0] ab_cd, ef_gh;

always @(posedge clk)
begin
    ab <= a * b;
```

```
    cd <= c * d;
    ef <= e * f;
    gh <= g * h;
    ab_cd <= ab + cd;
    ef_gh <= ef + gh;
    out <= ab_cd + ef_gh;
end
endmodule

module top(
   input clk,
   input signed [4:0] a1, b1, c1, d1, e1, f1, g1, h1,
   input signed [4:0] a2, b2, c2, d2, e2, f2, g2, h2,
   output signed [11:0] out1, out2
);
dot_product core1(.clk(clk), .a(a1), .b(b1), .c(c1), .d(d1),
    .e(e1), .f(f1), .g(g1), .h(h1), .out(out1));
dot_product core2(.clk(clk), .a(a2), .b(b2), .c(c2), .d(d2),
    .e(e2), .f(f2), .g(g2), .h(h2), .out(out2));
endmodule
```

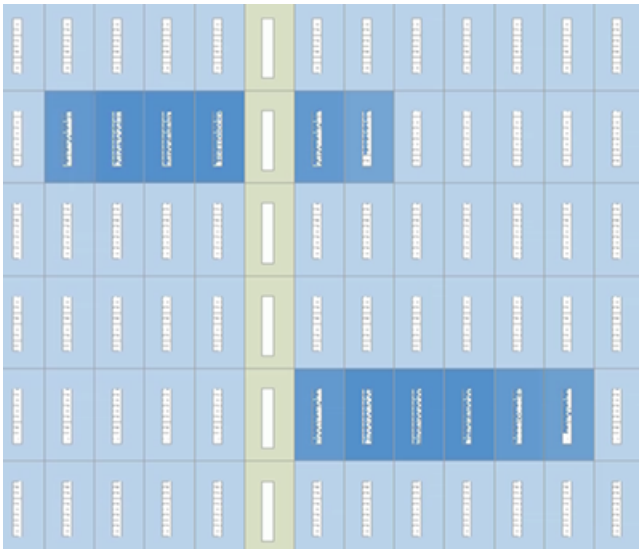
Quartus Prime synthesis displays the following messages in the console:

Figure 133. Console Messages

Message	Message ID
Inferred optimized multiplier from the following logic: "core1|mult_2"	20192
Inferred optimized multiplier from the following logic: "core1|mult_1"	20192
Inferred optimized multiplier from the following logic: "core1|mult_0"	20192
Inferred optimized multiplier from the following logic: "core2|mult_3"	20192
Inferred optimized multiplier from the following logic: "core2|mult_2"	20192
Inferred optimized multiplier from the following logic: "core2|mult_1"	20192
Inferred optimized multiplier from the following logic: "core2|mult_0"	20192
Timing-Driven Synthesis is running	286030
Applied dense packing to "core2". Area: 6 LABs. Logic density: 0.95.	20193
Applied dense packing to "core1". Area: 6 LABs. Logic density: 0.95.	20193

In the Chip Planner, you can observe this design having two signed dot-product cores independently optimized and placed:

Figure 134. Design Placement

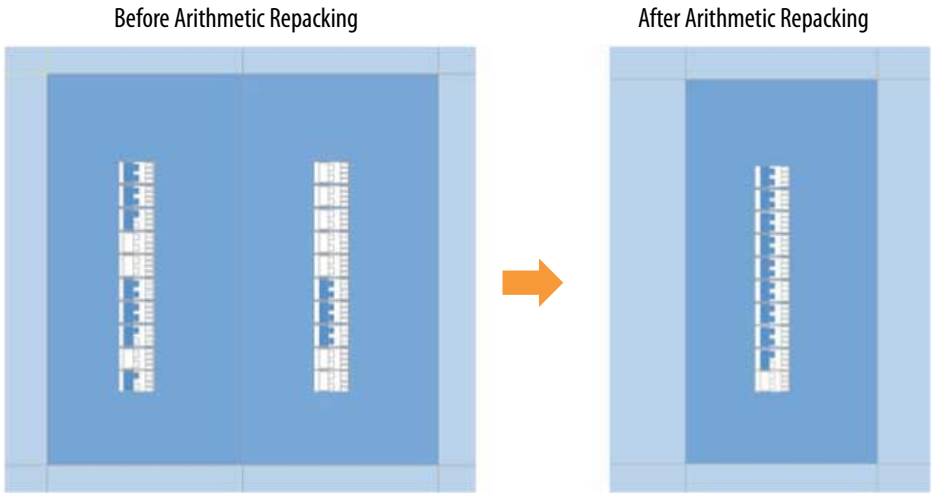


Continuous Arithmetic Packing

Continuous arithmetic packing re-synthesizes arithmetic gates into logic blocks optimally sized to fit into Intel FPGA LABs. This optimization allows up to 100% utilization of LAB resources for the arithmetic blocks.

When you enable fractal synthesis, the Compiler applies this optimization to all carry chains and two-input logic gates. This optimization can pack adder trees, multipliers, and any other arithmetic-related logic.

Figure 135. Continuous Arithmetic Packing



Note that continuous arithmetic packing works independently of multiplier regularization. So, if you are using a multiplier that is not regularized (such as writing your own multiplier) then continuous arithmetic packing can still operate.

1.15.6.1. Enabling or Disabling Fractal Synthesis

For Stratix 10 and Agilex 7 devices, fractal synthesis optimization runs automatically for small multipliers (any A*B statement in Verilog HDL or VHDL where bit-width of the operands is 7 or less). You can also disable automatic fractal synthesis for small multipliers for these devices using either of the following methods:

- In RTL, set the DSP multstyle, as "Multstyle Verilog HDL Synthesis Attribute" describes. For example:

```
(* multstyle = "dsp" *) module foo(...);
module foo(..) /* synthesis multstyle = "dsp" */;
```

- In the `.qsf` file, add as an assignment as follows:

```
set_instance_assignment -name DSP_BLOCK_BALANCING_IMPLEMENTATION \
     DSP_BLOCKS -to r
```

In addition, for Stratix 10, Agilex 7, Arria 10, and Cyclone 10 GX devices, you can enable fractal synthesis globally or for specific multipliers with the **Fractal Synthesis** GUI option or the corresponding FRACTAL_SYNTHESIS `.qsf` assignment:

- In RTL, use `altera_attribute` as follows:

```
(* altera_attribute = "-name FRACTAL_SYNTHESIS ON" *)
```

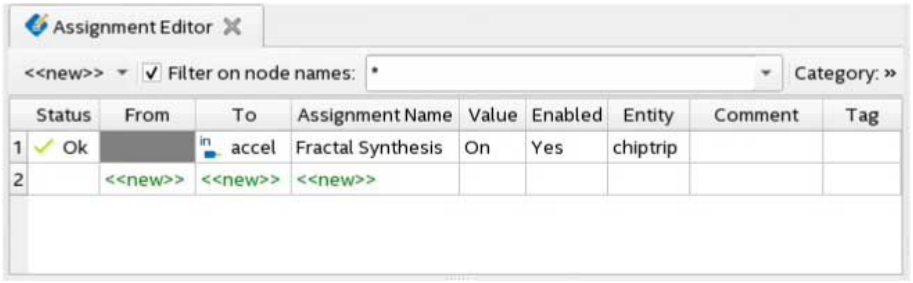
- In the `.qsf` file, add as an assignment as follows:

```
set_global_assignment -name FRACTAL_SYNTHESIS ON -entity <module name>
```

In the user interface, follow these steps:

1. Click **Assignments ➤ Assignment Editor**.
2. Select **Fractal Synthesis** for **Assignment Name**, **On** for the **Value**, the arithmetic-intensive entity name for **Entity**, and an instance name in the **To** column. You can enter a wildcard (*) for **To** to assign all instances of the entity.

Figure 136. Fractal Synthesis Assignment in Assignment Editor



Related Information

Multstyle Verilog HDL Synthesis Attribute
In Quartus Prime Help.

1.16. Synthesis Language Support

The Quartus Prime software synthesizes standard Verilog HDL, VHDL, and SystemVerilog design files.

1.16.1. Verilog and SystemVerilog Synthesis Support

Quartus Prime synthesis supports the following Verilog HDL language standards:

- Verilog-1995 (IEEE Standard 1364-1995)
- Verilog-2001 (IEEE Standard 1364-2001)
- SystemVerilog-2005 (IEEE Standard 1800-2005)
- SystemVerilog-2009 (IEEE Standard 1800-2009)
- SystemVerilog-2012 (IEEE Standard 1800-2012)

The following important guidelines apply to Quartus Prime synthesis of Verilog HDL and SystemVerilog:

- The Compiler uses the Verilog-2001 standard by default for files with an extension of `.v`, and the SystemVerilog standard for files with the extension of `.sv`.
- If you use scripts to add design files, you can use the `-HDL_VERSION` command to specify the HDL version for each design file.
- Compiler support for Verilog HDL is case sensitive in accordance with the Verilog HDL standard.
- The Compiler supports the compiler directive `` `define ``, in accordance with the Verilog HDL standard.
- The Compiler supports the `include` compiler directive to include files with absolute paths (with either "/" or "\" as the separator), or relative paths.
- When searching for a relative path, the Compiler initially searches relative to the project directory. If the Compiler cannot find the file, the Compiler next searches relative to all user libraries. Finally, the Compiler searches relative to the current file's directory location.
- Quartus Prime Pro Edition synthesis searches for all modules or entities earlier in the synthesis process than other Quartus software tools. This earlier search produces earlier syntax errors for undefined entities than other Quartus software tools.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations

1.16.1.1. Verilog HDL Input Settings (Settings Dialog Box)

Click **Assignments ➤ Settings ➤ Verilog HDL Input** to specify options for the synthesis of Verilog HDL input files.

Figure 137. Verilog HDL Input Settings Dialog Box

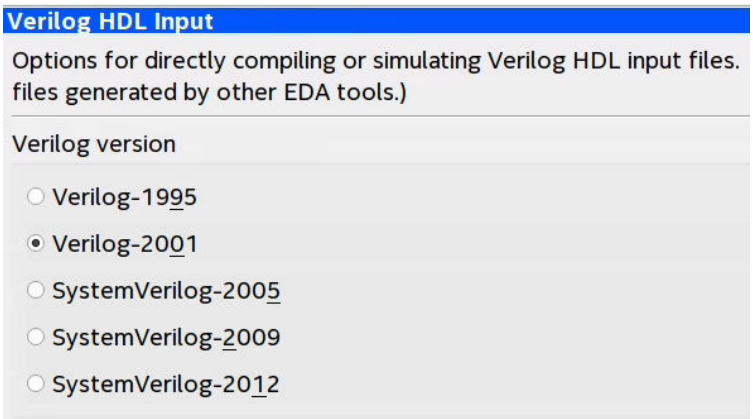


Table 36. Verilog HDL Input Settings

Setting	Description
Verilog Version	Directs synthesis to process Verilog HDL input design files using the specified standard. You can select any of the supported language standards to match your Verilog HDL files or SystemVerilog design files.
Library Mapping File	Allows you to optionally specify a provided Library Mapping File (`.lmf`) for use in synthesizing Verilog HDL files that contain non-Intel FPGA functions mapped to IP cores. You can specify the full path name of the LMF in the **File name** box.
Verilog HDL Macro	Verilog HDL macros are pre-compiler directives which can be added to Verilog HDL files to define constants, flags, or other features by **Name** and **Setting**. Macros that you add appear in the **Existing Verilog HDL macro settings** list.

1.16.1.2. Design Libraries

By default, the Compiler processes all design files into one or more libraries.

- When compiling a design instance, the Compiler initially searches for the entity in the library associated with the instance (which is the work library if you do not specify any library).
- If the Compiler cannot locate the entity definition, the Compiler searches for a unique entity definition in all design libraries.
- If the Compiler finds more than one entity with the same name, the Compiler generates an error. If your design uses multiple entities with the same name, you must compile the entities into separate libraries.

1.16.1.3. Verilog HDL Configuration

Verilog HDL configuration is a set of rules that specify the source code for particular instances. Verilog HDL configuration allows you to perform the following tasks:

- Specify a library search order for resolving cell instances (as does a library mapping file).
- Specify overrides to the logical library search order for specified instances.
- Specify overrides to the logical library search order for all instances of specified cells.

1.16.1.3.1. Hierarchical Design Configurations

A design can have more than one configuration. For example, you can define a configuration that specifies the source code you use in particular instances in a sub-hierarchy, and then define a configuration for a higher level of the design.

For example, suppose a subhierarchy of a design is an eight-bit adder, and the RTL Verilog code describes the adder in a logical library named `rtllib`. The gate-level code describes the adder in the `gatelib` logical library. If you want to use the gate-level code for the 0 (zero) bit of the adder and the RTL level code for the other seven bits, the configuration might appear as follows:

Example 13. Gate-level code for the 0 (zero) bit of the adder

```
config cfg1;
design aLib.eight_adder;
default liblist rtllib;
instance adder.fulladd0 liblist gatelib;
endconfig
```

If you are instantiating this eight-bit adder eight times to create a 64-bit adder, use configuration `cfg1` for the first instance of the eight-bit adder, but not in any other instance. A configuration that performs this function is shown below:

Example 14. Use configuration `cfg1` for first instance of eight-bit adder

```
config cfg2;
design bLib.64_adder;
default liblist bLib;
instance top.64add0 use work.cfg1:config;
endconfig
```

Note: The name of the unbound module may be different from the name of the cell that is bounded to the instance.

1.16.1.4. Initial Constructs and Memory System Tasks

The Quartus Prime software infers power-up conditions from the Verilog HDL `initial` constructs. The Quartus Prime software also creates power-up settings for variables, including RAM blocks. If the Quartus Prime software encounters non-synthesizable constructs in an `initial` block, it generates an error.

To avoid such errors, enclose non-synthesizable constructs (such as those intended only for simulation) in `translate_off` and `translate_on` synthesis directives. Synthesis of initial constructs enables the power-up state of the synthesized design to match the power-up state of the original HDL code in simulation.

Note: Initial blocks do not infer power-up conditions in some third-party EDA synthesis tools. If you convert between synthesis tools, you must set your power-up conditions correctly.

Quartus Prime synthesis supports the `$readmemb` and `$readmemh` system tasks to initialize memories.

Example 15. Verilog HDL Code: Initializing RAM with the readmemb Command

```
reg [7:0] ram[0:15];
initial
begin
$readmemb("ram.txt", ram);
end
```

When creating a text file to use for memory initialization, specify the address using the format @<*location*> on a new line, and then specify the memory word such as `110101` or `abcde` on the next line.

The following example shows a portion of a Memory Initialization File (`.mif`) for the RAM.

Example 16. Text File Format: Initializing RAM with the readmemb Command

```
@0
00000000
@1
00000001
@2
00000010
…
@e
00001110
@f
00001111
```

1.16.1.5. Verilog HDL Macros

The Quartus Prime software fully supports Verilog HDL macros, which you can define with the `'define` compiler directive in your source code. You can also define macros in the Quartus Prime software or on the command line.

To set Verilog HDL macros at the command line for the Quartus Prime Pro Edition synthesis (`quartus_syn`) executable, use the following format:

```
quartus_syn <PROJECT_NAME> --set=VERILOG_MACRO=a=2
```

This command adds the following new line to the project `.qsf` file:

```
set_global_assignment -name VERILOG_MACRO "a=2"
```

To avoid adding this line to the project `.qsf`, add this option to the `quartus_syn` command:

```
--write_settings_files=off
```

1.16.2. VHDL Synthesis Support

Quartus Prime synthesis supports the following VHDL language standards.

- VHDL 1987 (IEEE Standard 1076-1987)
- VHDL 1993 (IEEE Standard 1076-1993)
- VHDL 2008 (IEEE Standard 1076-2008)
- VHDL 2019 (IEEE Standard 1076-2019)(3)

The Quartus Prime Compiler uses the VHDL 1993 standard by default for files that have the extension `.vhdl` or `.vhd`.

(3) Only a subset of the features introduced by in VHDL 2019 are supported: Interfaces and Conditional Analysis.

Note: The VHDL code samples follow the VHDL 1993 standard.

Related Information

- Migrating to Quartus Prime Pro Edition
- VHDL-2019 Conditional Analysis Tool Directives on page 167

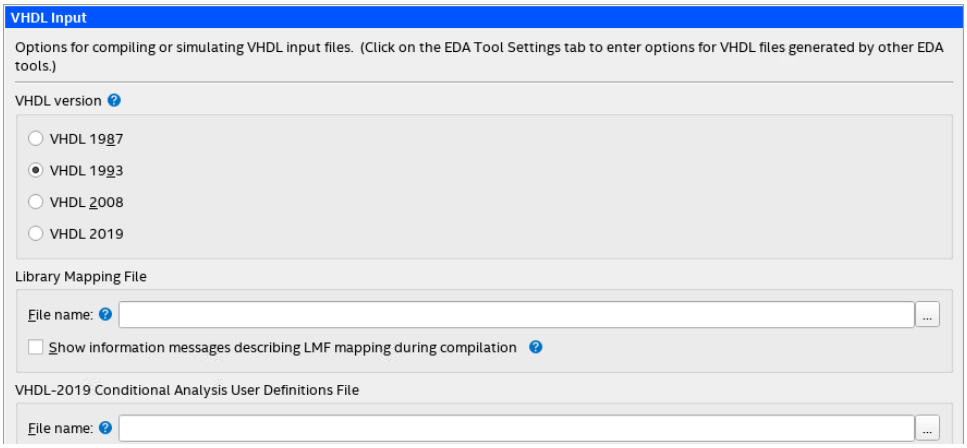
1.16.2.1. VHDL Input Settings (Settings Dialog Box)

Click **Assignments ➤ Settings ➤ VHDL Input** to specify options for the synthesis of VHDL input files.

Table 37. VHDL Input Settings

Setting	Description
VHDL Version	Specifies the VHDL standard for use during synthesis of VHDL input design files. Select the language standards that corresponds with the VHDL files.
Library Mapping File	Specifies a Library Mapping File (`.lmf`) for use in synthesizing VHDL files that contain IP cores. Specify the full path name of the LMF in the **File name** box.
VHDL-2019 Conditional Analysis User Definitions File	Specifies the `.ini` file that contains your user-defined VHDL 2019 conditional analysis identifier-value pairs.

Figure 138. VHDL Input Settings Dialog Box



Related Information

VHDL-2019 Conditional Analysis Tool Directives on page 167

1.16.2.2. VHDL Standard Libraries and Packages

The Quartus Prime software includes the standard IEEE libraries and several vendor-specific VHDL libraries. The IEEE library includes the standard VHDL packages `std_logic_1164`, `numeric_std`, `numeric_bit`, and `math_real`.

The STD library is part of the VHDL language standard and includes the packages `standard` (included in every project by default) and `textio`. For compatibility with older designs, the Quartus Prime software also supports the following vendor-specific packages and libraries:

- Synopsys* packages such as `std_logic_arith` and `std_logic_unsigned` in the IEEE library.
- Mentor Graphics* packages such as `std_logic_arith` in the ARITHMETIC library.
- Primitive packages `altera_primitives_components` (for primitives such as GLOBAL and DFFE) and `maxplus2` in the ALTERA library.
- IP core packages `altera_mf_components` in the ALTERA_MF library for specific IP cores including LCELL. In addition, `lpm_components` in the LPM library for library of parameterized modules (LPM) functions.

Note: Import component declarations for primitives such as GLOBAL and DFFE from the `altera_primitives_components` package and not the `altera_mf_components` package.

1.16.2.3. VHDL wait Constructs

The Quartus Prime software supports one VHDL `wait until` statement per process block. However, the Quartus Prime software does not support other VHDL wait constructs, such as `wait for` and `wait on` statements, or processes with multiple `wait` statements.

Example 17. VHDL `wait until` construct example

```
architecture dff_arch of ls_dff is
begin
output: process begin
wait until (CLK'event and CLK='1');
Q <= D;
Qbar <= not D;
end process output;
end dff_arch;
```

1.16.2.4. VHDL-2019 Conditional Analysis Tool Directives

Quartus Prime Pro Edition provides support for VHDL-2019 (IEEE Std 1076-2019) section 24.2 Conditional analysis tool directives.

With conditional analysis tool directives, your VHDL description can be varied according to directives stored in a separate `.ini` file.

Create the `.ini` file and specify the path in the **File name** field of the **VHDL-2019 Conditional Analysis User Definitions File** panel of the **VHDL Input** compiler settings page.

The format for the file is as follows:

- One *identifier*="*value*" pair per line
- Use ; or # characters to start line or trailing comments
- Identifiers must follow the requirements for a basic identifier as specified by the VHDL standard:
  - It must start with a letter.
  - It must contain only alphanumeric and underscore ("_") characters.
- Values must be surrounded by double quotes.

A line in the file with invalid syntax is ignored and generates a warning message. Lines that attempt to redefine the standard identifiers are ignored.

An example of a conditional analysis user definitions file is as follows:

```
USER_VAR1="ABC"
USER_VAR2 = "xyz"

# line comment
; line comment
USER_VAR3 = "TEST" # trailing comment
USER_VAR4 = "lorem" ; trailing comment

USER_VAR5=";# comment characters in quotes are ignored"
```

Standard Conditional Analysis Identifiers

Quartus Prime provides the following standard conditional analysis identifiers:

- `VHDL_VERSION = "<version>"`

  For example, `VHDL_VERSION = "2019"`. The values for <*version*> are restricted by the IEEE standard.

  `VHDL_VERSION` is set per file and can have a different value in each file.
- `TOOL_TYPE = "SYNTHESIS"`

  The values for `TOOL_TYPE` are restricted by the IEEE standard.
- `TOOL_VENDOR = "INTEL CORPORATION"`
- `TOOL_NAME = "QUARTUS"`
- `TOOL_EDITION = "PRIME PRO"`
- `TOOL_VERSION = "<major and minor version>"`

  For example, `TOOL_VERSION = "21.3.0"`

1.17. Synthesis Settings Reference

This section provides a reference to all synthesis settings. Use these settings to customize synthesis processing for your design goals.

1.17.1. Advanced Synthesis Settings

The following section is a quick reference of all Advanced Synthesis Settings. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** to modify these settings.

Table 38. Advanced Synthesis Settings (1 of 13)

Option	Description
Allow Any RAM Size for Recognition	Allows the Compiler to infer RAMs of any size, even if the RAMs do not meet the current minimum requirements.
Allow Any ROM Size for Recognition	Allows the Compiler to infer ROMs of any size even if the ROMs do not meet the design's current minimum size requirements.
Allow Any Shift Register Size for Recognition	Allows the Compiler to infer shift registers of any size even if they do not meet the design's current minimum size requirements.
Allow Register Duplication	Controls whether the Compiler duplicates registers to improve design performance. When enabled, the Compiler performs optimization that creates a second copy of a register and move a portion of its fan-out to this new node. This technique improves routability and reduces the total routing wire required to route a net with many fan-outs. If you disable this option, retiming of registers is also disabled. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.
Allow Register Merging	Controls whether the Compiler removes (merges) identical registers. When enabled, in cases where two registers generate the same logic, the Compiler may delete one register and fan-out the remaining register to the deleted register's destinations. This option is useful if you want to prevent the Compiler from removing duplicate registers that you have used deliberately. When disabled, retiming optimizations are also disabled. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.
Allow Shift Register Merging Across Hierarchies	Allows the Compiler to take shift registers from different hierarchies of the design and put the registers in the same RAM.
Allow Synchronous Control Signals	Allows the Compiler to utilize synchronous clear and synchronous load signals in normal mode logic cells. Turning on this option helps to reduce the total number of logic cells used in the design, but can negatively impact the fitting. This negative impact occurs because all the logic cells in a LAB share synchronous control signals.

Table 39. Advanced Synthesis Settings (2 of 13)

Option	Description
Analysis & Synthesis Message Level	Specifies the type of Analysis & Synthesis messages the Compiler display. **Low** displays only the most important Analysis & Synthesis messages. **Medium** displays most messages, but hides the detailed messages. **High** displays all messages.
Auto Clock Enable Replacement	Allows the Compiler to locate logic that feeds a register and move the logic to the register's clock enable input port.
Auto DSP Block Replacement	Allows the Compiler to find a multiply-accumulate function or a multiply-add function that can be replaced with a DSP block.
Auto Gated Clock Conversion	Automatically converts gated clocks to use clock enable pins. Clock gating logic can contain AND, OR, MUX, and NOT gates. Turning on this option may increase memory use and overall run time. You must use the Timing Analyzer for timing analysis, and you must define all base clocks in Synopsys Design Constraints (`.sdc`) format.

Table 40. Advanced Synthesis Settings (3 of 13)

Option	Description
Auto Open-Drain Pins	Allows the Compiler to automatically convert a tri-state buffer with a strong low data input into the equivalent open-drain buffer.
Auto RAM Replacement	Allows the Compiler to identify sets of registers and logic that it can replace with the altsyncram or the lpm_ram_dp IP core. Turning on this option may change the functionality of the design.
Auto ROM Replacement	Allows the Compiler to identify logic that it can replace with the altsyncram or the lpm_rom IP core. Turning on this option may change the power-up state of the design.

continued...

Option	Description
Auto Resource Sharing	Allows the Compiler to share hardware resources among many similar, but mutually exclusive, operations in your HDL source code. If you enable this option, the Compiler merges compatible addition, subtraction, and multiplication operations. Merging operations may reduce the area your design requires. Because resource sharing introduces extra muxing and control logic on each shared resource, it may negatively impact the final f_{MAX} of your design.
Auto Shift Register Placement	Allows the Compiler to find a group of shift registers of the same length that are replaceable with the altshift_taps IP core. The shift registers must all use the same clock and clock enable signals. The registers must not have any other secondary signals. The registers must have equally spaced taps that are at least three registers apart.
Automatic Parallel Synthesis	Option to enable/disable automatic parallel synthesis. Use this option to speed up synthesis compile time by using multiple processors when available.

Table 41. Advanced Synthesis Settings (4 of 13)

Option	Description
Block Design Naming	Specifies the naming scheme for the block design. The Compiler ignores the option if you assign the option to anything other than a design entity.
Clock MUX Protection	Causes the multiplexers in the clock network to decompose to 2-to-1 multiplexer trees. The Compiler protects these trees from merging with, or transferring to, other logic. This option helps the Timing Analyzer to analyze clock behavior.
DSP Block Balancing	Allows you to control the conversion of certain DSP block slices during DSP block balancing.

Table 42. Advanced Synthesis Settings (5 of 13)

Option	Description
Disable DSP Negate Inferencing	Allows you to specify whether to use the negate port on an inferred DSP block.
Disable Register Merging Across Hierarchies	Specifies whether the Compiler allows merging of registers that are in different hierarchies if their inputs are the same.
Enable Formal Verification Support	Enables the Compiler to write scripts for use with the OneSpin* formal verification tool.
Enable State Machines Inference	Allows the Compiler to infer state machines from VHDL or Verilog HDL design files. The Compiler optimizes state machines to reduce area and improve performance. If set to **Off**, the Compiler extracts and optimizes state machines in VHDL or Verilog HDL design files as regular logic.
Enable SystemVerilog static assertion support	Enables immediate assertions in the Compiler for information, warning, and error messages for SystemVerilog designs.
Enable VHDL static assertion support	Enables immediate assertions in the Compiler for information, warning, and error messages for VHDL designs.
Force Use of Synchronous Clear Signals	Forces the Compiler to utilize synchronous clear signals in normal mode logic cells. Enabling this option helps to reduce the total number of logic cells in the design, but can negatively impact the fitting. All the logic cells in a LAB share synchronous control signals.
Fractal Synthesis	Turning this option **On** directs the Compiler to apply dense packing to arithmetic blocks, minimizing the area of the design for arithmetic-intensive designs.
HDL Message Level	Specifies the type of HDL messages you want to view, including messages that display processing errors in the HDL source code. **Level1** displays only the most important HDL messages. **Level2** displays most HDL messages, including warning and information based messages. **Level3** displays all HDL messages, including warning and information based messages and alerts about potential design problems or lint errors.

Table 43. Advanced Synthesis Settings (6 of 13)

Option	Description
Ignore GLOBAL Buffers	Ignores GLOBAL buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual GLOBAL buffer, or a design entity containing GLOBAL buffers.
Ignore LCELL Buffers	Ignores LCELL buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual LCELL buffer, or a design entity containing LCELL buffers.
Ignore Maximum Fan-Out Assignments	Directs the Compiler to ignore the Maximum Fan-Out Assignments on a node, an entity, or the whole design.
Ignore SOFT Buffers	Ignores SOFT buffers in the design. The Compiler ignores this option if you apply the option to anything other than an individual SOFT buffer or a design entity containing SOFT buffers.

Table 44. Advanced Synthesis Settings (7 of 13)

Option	Description
Ignore translate_off and synthesis_off Directives	Ignores all `translate_off/synthesis_off` synthesis directives in Verilog HDL and VHDL design files. Use this option to disable these synthesis directives and include previously ignored code during elaboration.
Infer RAMs from Raw Logic	Infers RAM from registers and multiplexers. The Compiler initially converts some HDL patterns differing from RAM templates into logic. However, these structures function as RAM. As a result, when you enable this option, the Compiler may substitute the altsyncram IP core instance for them at a later stage. When you enable this assignment, the Compiler may use more device RAM resources and fewer LABs.
Iteration Limit for Constant Verilog Loops	Defines the iteration limit for Verilog loops with loop conditions that evaluate to compile-time constants on each loop iteration. This limit exists primarily to identify potential infinite loops before they exhaust memory or trap the software in an actual infinite loop.
Iteration Limit for non-Constant Verilog Loops	Defines the iteration limit for Verilog HDL loops with loop conditions that do not evaluate to compile-time constants on each loop iteration. This limit exists primarily to identify potential infinite loops before they exhaust memory or trap the software in an actual infinite loop.

Table 45. Advanced Synthesis Settings (8 of 13)

Option	Description
Maximum DSP Block Usage	Specifies the maximum number of DSP blocks that the DSP block balancer assumes exist in the current device for each partition. This option overrides the usual method of using the maximum number of DSP blocks the current device supports.
Maximum Number of LABs	Specifies the maximum number of LABs that Analysis & Synthesis should try to utilize for a device. This option overrides the usual method of using the maximum number of LABs the current device supports, when the value is non-negative and is less than the maximum number of LABs available on the current device.
Maximum Number of M4K/M9K/M20K/M10K Memory Blocks	Specifies the maximum number of M4K, M9K, M20K, or M10K memory blocks that the Compiler may use for a device. This option overrides the usual method of using the maximum number of M4K, M9K, M20K, or M10K memory blocks the current device supports, when the value is non-negative and is less than the maximum number of M4K, M9K, M20K, or M10K memory blocks available on the current device.

Table 46. Advanced Synthesis Settings (9 of 13)

Option	Description
Maximum Number of Registers Created from Uninferred RAMs	Specifies the maximum number of registers that Analysis & Synthesis uses for conversion of uninferred RAMs. Use this option as a project-wide option or on a specific partition by setting the assignment on the instance name of the partition root. The

continued...

Option	Description
	assignment on a partition overrides the global assignment (if any) for that particular partition. This option prevents synthesis from causing long compilations and running out of memory when many registers are used for uninferred RAMs. Instead of continuing the compilation, the Quartus Prime software issues an error and exits.
NOT Gate Push-Back	Allows the Compiler to push an inversion (that is, a NOT gate) back through a register and implement it on that register's data input if it is necessary to implement the design. When this option is on, a register may power-up to an active-high state, and may need explicit clear during initial operation of the device. The Compiler ignores this option if you apply it to anything other than an individual register or a design entity containing registers. When you apply this option to an output pin that is directly fed by a register, the assignment automatically transfers to that register.
Number of Inverted Registers Reported in Synthesis Report	Specifies the maximum number of inverted registers that the Synthesis report displays.
Number of Protected Registers Reported in Synthesis Report	Specifies the maximum number of protected registers that the Synthesis Report displays.
Number of Removed Registers Reported in Synthesis Migration Checks	Specifies the maximum number of rows that the Synthesis Migration Check report displays.
Number of Swept Nodes Reported in Synthesis Report	Specifies the maximum number of swept nodes that the Synthesis Report displays. A swept node is any node which was eliminated from your design because the Compiler found the node to be unnecessary.
Number of Rows Reported in Synthesis Report	Specifies the maximum number of rows that the Synthesis report displays. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.
Optimization Technique	Specifies an overall optimization goal for Analysis & Synthesis. Specify a **Balanced** strategy, or optimize for **Performance**, **Area**, **Routability**, **Power**, or **Compile Time**. The Compiler targets the optimization goal you specify.

Table 47. Advanced Synthesis Settings (10 of 13)

Option	Description
Perform WYSIWYG Primitive Resynthesis	Specifies whether to perform WYSIWYG primitive resynthesis during synthesis. This option uses the setting specified in the **Optimization Technique** logic option.
Power-Up Don't Care	Causes registers that do not have a **Power-Up Level logic** option setting to power-up with a do not care logic level (X). When the **Power-Up Don't Care** option is on, the Compiler determines when it is beneficial to change the power-up level of a register to minimize the area of the design. The Compiler maintains a power-up state of zero, unless there is an immediate area advantage.
Power Optimization During Synthesis	Controls the power-driven compilation setting of Analysis & Synthesis. This option determines how aggressively Analysis & Synthesis optimizes the design for power. When this option is **Off**, the Compiler does not perform any power optimizations. **Normal compilation** performs power optimizations provided that they are not expected to reduce design performance. **Extra effort** performs additional power optimizations which may reduce design performance.

Table 48. Advanced Synthesis Settings (11 of 13)

Option	Description
Remove Duplicate Registers	Removes a register if it is identical to another register. If two registers generate the same logic, the Compiler deletes the duplicate. The first instance fans-out to the duplicates destinations. Also, if the deleted register contains different logic option assignments, the Compiler ignores the options. This option is useful if you want to

continued...

Option	Description
	prevent the Compiler from removing intentionally duplicate registers. The Compiler ignores this option if you apply it to anything other than an individual register or a design entity containing registers.
Remove Redundant Logic Cells	Removes redundant LCELL primitives or WYSIWYG primitives. Turning this option on optimizes a circuit for area and speed. The Compiler ignores this option if you apply it to anything other than a design entity.
Report Parameter Settings	Specifies whether the Synthesis report includes the reports in the **Parameter Settings by Entity Instance** folder.
Report PR Initial Values as Errors	Allows you to flag explicitly defined initial values found in PR partitions as Errors instead of Warnings.
Report Source Assignments	Specifies whether the Synthesis report includes reports in the **Source Assignments** folder.

Table 49. Advanced Synthesis Settings (12 of 13)

Option	Description
Resource Aware Inference for Block RAM	Specifies whether RAM, ROM, and shift-register inference should take the design and device resources into account.
Restructure Multiplexers	Reduces the number of logic elements synthesis requires to implement multiplexers in a design. This option is useful if your design contains buses of fragmented multiplexers. This option repacks multiplexers more efficiently for area, allowing the design to implement multiplexers with a reduced number of logic elements: • **On**—minimizes your design area, but may negatively affect design clock speed (f_{MAX}). • **Off**—disables multiplexer restructuring; it does not decrease logic element usage and does not affect design clock speed (f_{MAX}). • **Auto**—allows the Quartus Prime software to determine whether multiplexer restructuring should be enabled. The **Auto** setting decreases logic element usage, but may negatively affect design clock speed (f_{MAX}).
SDC Constraint Protection	Verifies `.sdc` constraints in register merging. This option helps to maintain the validity of `.sdc` constraints through compilation.
Safe State Machine	The **Safe State Machine** option implements state machines that can recover from an illegal state. The following settings are available: • **Auto**—for Stratix 10 or Agilex 7 designs, this default setting enables **Safe State Machine** whenever the Compiler determines this setting is advantageous in state machines of 6 or less states. The setting helps to allow for unexpected initial power-up conditions. For Arria 10 and Cyclone 10 GX, the **Auto** setting is the same as **Never**. • **On**—directs the Compiler to always use **Safe State Machine**. • **Never**—never uses **Safe State Machine**.
Shift Register Replacement – Allow Asynchronous Clear Signal	Allows the Compiler to find a group of shift registers of the same length that can be replaced with the altshift_taps IP core. The shift registers must all use the same `aclr` signals, must not have any other secondary signals, and must have equally spaced taps that are at least three registers apart. To use this option, you must turn on the **Auto Shift Register Replacement** logic option.
Size of the Latch Report	Allows you to specify the maximum number of latches that the Synthesis Report should display.
Size of the PR Initial Conditions Report	Allows you to specify the maximum number of registers that the PR Initial Conditions Report should display.

Table 50. Advanced Synthesis Settings (13 of 13)

Option	Description
State Machine Processing	Specifies the processing style the Compiler uses to process a state machine. You can use your own **User-Encoded** style, or select **One-Hot**, **Minimal Bits**, **Gray**, **Johnson**, **Sequential,** or **Auto** (Compiler-selected) encoding.
Strict RAM Replacement	When this option is **On**, the Compiler replace RAM only if the hardware matches the design exactly.
Synchronization Register Chain Length	Specifies the maximum number of registers in a row that the Compiler considers as a synchronization chain. Synchronization chains are sequences of registers with the same clock and no fan-out in between, such that the first register is fed by a pin, or by logic in another clock domain. The Compiler considers these registers for metastability analysis. The Compiler prevents optimizations of these registers, such as retiming. When gate-level retiming is enabled, the Compiler does not remove these registers. The default length is set to two.
Synthesis Effort	Controls the synthesis trade-off between compilation speed, performance, and area. The default is **Auto**. You can select **Fast** for faster compilation speed at the cost of performance and area.
Synthesis Migration Check for Stratix 10	Enables synthesis checks on Arria 10 to Stratix 10 design migration.
Timing-Driven Synthesis	For Arria 10 and Cyclone 10 GX designs, allows synthesis to use timing information to better optimize the design. The **Timing-Driven Synthesis** logic option impacts the following **Optimization Technique** options: • **Optimization Technique Speed**—optimizes timing-critical portions of your design for performance at the cost of increasing area (logic and register utilization) • **Optimization Technique Balanced**—also optimizes the timing-critical portions of your design for performance, but the option allows only limited area increase • **Optimization Technique Area**—optimizes your design only for area

1.18. Fitter Settings Reference

Use Fitter settings to customize the place and route of your design. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to access Fitter settings.

Table 51. Advanced Fitter Settings (1 of 8)

Option	Description
ALM Register Packing Effort	Guides aggressiveness of the Fitter in packing ALMs during register placement. Use this option to increase secondary register locations. Increasing ALM packing density may lower the number of ALMs needed to fit the design, but it may also reduce routing flexibility and timing performance. • **Low**—the Fitter avoids ALM packing configurations that combine LUTs and registers which have no direct connectivity. Avoiding these configurations may improve timing performance but increases the number of ALMs to implement the design. • **Medium**—the Fitter allows some configurations that combine unconnected LUTs and registers to be implemented in ALM locations. The Fitter makes more use of secondary register locations within the ALM. • **High**—the Fitter enables all legal and desired ALM packing configurations. In dense designs, the Fitter automatically increases the ALM register packing effort as required to enable the design to fit.
Advanced Physical Synthesis	Enables the Physical Synthesis engine that includes combinational and sequential optimization during fitting to improve circuit performance.
Allow Delay Chains	Allows the Fitter to choose the optimal delay chain to meet t_{SU} and t_{CO} timing requirements for all I/O elements. Enabling this option may reduce the number of t_{SU} violations, while introducing a minimal number of t_H violations. Enabling this option does not override delay chain settings on individual nodes.

continued...

Option	Description
Allow DSP Retiming	Allow retiming through DSP blocks.
Allow Early Global Retiming in the Fitter	Allows the Compiler to run global retiming early in the Fitter.
Allow Hyper-Aware Register Chain Area Optimizations in the Fitter	Reduces ALM usage by automatically forcing some back-to-back registers into Hyper Registers. Turning on this area reduction technique may reduce performance and increase compile time.
Allow RAM Retiming	Allow retiming through RAM blocks.
Allow Register Duplication	Allows the Compiler to duplicate registers to improve design performance. When you enable this option, the Compiler copies registers and moves some fan-out to this new node. This optimization improves routability and can reduce the total routing wire in nets with many fan-outs. If you disable this option, this disables optimizations that retime registers. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.
Allow Register Merging	Allows the Compiler to remove registers that are identical to other registers in the design. When you enable this option, in cases where two registers generate the same logic, the Compiler deletes one register, and the remaining registers fan-out to the deleted register's destinations. This option is useful if you want to prevent the Compiler from removing intentional use of duplicate registers. If you disable register merging, the Compiler disables optimizations that retime registers. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.
Auto Delay Chains for High Fanout Input Pins	Allows the Fitter to choose how to optimize the delay chains for high fan-out input pins. You must enable **Auto Delay Chains** to enable this option. Enabling this option may reduce the number of t_{SU} violations, but the compile time increases significantly, as the Fitter tries to optimize the settings for all fan-outs.
Auto Fit Effort Desired Slack Margin	Specifies the default worst-case slack margin the Fitter maintains for. If the design is likely to have at least this much slack on every path, the Fitter reduces optimization effort to reduce compilation time. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.

Table 52. Advanced Fitter Settings (2 of 8)

Option	Description
Auto Global Clock	Allows the Compiler to choose the global clock signal. The Compiler chooses the signal that feeds the most clock inputs to flip-flops. This signal is available throughout the device on the global routing paths. To prevent the Compiler from automatically selecting a particular signal as global clock, set the **Global Signal** option to **Off** on that signal.
Auto Global Register Control Signals	Allows the Compiler to choose global register control signals. The Compiler chooses signals that feed the most control signal inputs to flip-flops (excluding clock signals) as the global signals. These global signals are available throughout the device on the global routing paths. Depending on the target device family, these control signals can include asynchronous clear and load, synchronous clear and load, clock enable, and preset signals. If you want to prevent the Compiler from automatically selecting a particular signal as a global register control signal, set the **Global Signal** option to **Off** on that signal.
Auto Packed Registers	Allows the Compiler to combine a register and a combinational function, or to implement registers using I/O cells, RAM blocks, or DSP blocks instead of logic cells. This option controls how aggressively the Fitter combines registers with other function blocks to reduce the area of the design. Generally, the **Auto** or **Sparse Auto** settings are appropriate.

continued...

Option	Description
	The other settings limit the flexibility of the Fitter to combine registers with other function blocks and can result in no fits. • **Auto**—the Fitter attempts to achieve the best performance with good area. If necessary, the Fitter combines additional logic to reduce the area of the design to within the current device. • **Sparse Auto**—the Fitter attempts to achieve the highest performance, but may increase device usage without exceeding the device logic capacity. • **Off**—the Fitter does not combine registers with other functions. The **Off** setting severely increases the area of the design and may cause a no fit. • **Sparse**—the Fitter combines functions in a way which improves performance for many designs. • **Normal**—the Fitter combines functions that are expected to maximize design performance and reduce area. • **Minimize Area**—the Fitter aggressively combines unrelated functions to reduce the area required for placing the design, at the expense of performance. • **Minimize Area with Chains**—the Fitter even more aggressively combines functions that are part of register cascade chains or can be converted to register cascade chains. If this option is set to any value but **Off**, registers combine with I/O cells to improve I/O timing. This remains true provided that the **Optimize IOC Register Placement For Timing** option is enabled.
Auto RAM to MLAB Conversion	Specifies whether the Fitter converts RAMs of **Auto** block type to use LAB locations. If this option is set to **Off,** only MLAB cells or RAM cells with a block type setting of **MLAB** use LAB locations to implement memory.
Auto Register Duplication	Allows the Fitter to automatically duplicate registers within a LAB that contains empty logic cells. This option does not alter the functionality of the design. The Compiler ignores the **Auto Register Duplication** option if you select **OFF** as the setting for the **Logic Cell Insertion -- Logic Duplication** logic option. Turning on this option allows the **Logic Cell Insertion -- Logic Duplication** logic option to improve a design's routability, but can make formal verification of a design more difficult. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.

Table 53. Advanced Fitter Settings (3 of 8)

Option	Description
Enable Auto-Pipelining	Turns on the auto-pipelining and latency-insensitive false path feature. Use this setting in conjunction with the **Maximum Additional Pipelining** and optional **Additional Pipelining Group** assignments in the Assignment Editor to automatically add pipeline registers at the locations you specify. *Note:* Only available for Stratix 10 and Agilex 7 devices.
Enable Bus-Hold Circuitry	Enables bus-hold circuitry during device operation. When this option is **On**, a pin retains its last logic level when it is not driven, and does not go to a high impedance logic level. Do not use this option with the **Weak Pull-Up Resistor** option because doing so enables the location of the Critical Chain Viewer from the Fast option. The Compiler ignores this option if you apply it to anything other than a pin.
Enable Critical Chain Viewer	Enables critical chain visualization in the Fast Forward Timing Closure Recommendations report for Stratix 10 and Agilex 7 devices.
Equivalent RAM and MLAB Paused Read Capabilities	Specifies whether RAMs implemented in MLAB cells must have equivalent paused read capabilities as RAMs implemented in block RAM. Pausing a read is the ability to keep around the last read value when reading is disabled. Allowing differences in paused read capabilities provides the Fitter more flexibility in implementing RAMs using MLAB cells. To allow the Fitter the most flexibility in deciding which RAMs are implemented using MLAB cells, set this option to **Don't Care**. The following options are available: • **Don't Care**—the Fitter can convert RAMs to MLAB cells, even if they do not have equivalent paused read capabilities to a block RAM implementation. The Fitter generates an information message about RAMs with different paused read capabilities. • **Care**—the Fitter does not convert RAMs to MLAB cells unless they have the equivalent paused read capabilities to a block RAM implementation.

continued...

Option	Description
Equivalent RAM and MLAB Power Up	Specifies whether RAMs implemented in MLAB cells must have equivalent power-up conditions as RAMs implemented in block RAM. Power-up conditions occur when the device powers-up or globally resets. Allowing non-equivalent power-up conditions provides the Fitter more flexibility in implementing RAMs using MLAB cells. To allow the Fitter the most flexibility in deciding which RAMs are implemented using MLAB cells, set this option to **Auto** or **Don't Care**. The following options are available: • **Auto**—the Fitter may convert RAMs to MLAB cells, even if the MLAB cells lack equivalent power-up conditions to a block RAM implementation. The Fitter also outputs a warning message about RAMs with non-equivalent power up conditions. • **Don't Care**—the same behavior as **Auto** applies, but the message is an information message. • **Care**—the Fitter does not convert RAMs to MLAB cells unless they have equivalent power up conditions to a block RAM implementation.
Final Placement Optimizations	Specifies whether the Fitter performs final placement optimizations. Performing final placement optimizations may improve timing and routability, but may also require longer compilation time.
Fitter Aggressive Routability Optimizations	Specifies whether the Fitter aggressively optimizes for routability. Performing aggressive routability optimizations may decrease design speed, but may also reduce routing wire usage and routing time. The **Automatically** setting allows the Fitter to decide whether aggressive routability is beneficial.

Table 54. Advanced Fitter Settings (4 of 8)

Option	Description
Fitter Effort	Specifies the level of physical synthesis optimization during fitting: • **Auto**—adjusts the Fitter optimization effort to minimize compilation time, while still achieving the design timing requirements. Use the **Auto Fit Effort Desired Slack Margin** option to apply sufficient optimization effort to achieve additional timing margin. • **Standard**—uses maximum effort regardless of the design's requirements, leading to higher compilation time and more margin on easier designs. For difficult designs, **Auto** and **Standard** both use maximum effort. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.
Fitter Initial Placement Seed	Specifies the seed for the current design. The value can be any non-negative integer value. By default, the Fitter uses a seed of 1. The Fitter uses the seed as the initial placement configuration when optimizing design placement to meet timing requirements f_{MAX}. Because each different seed value results in a somewhat different fit, you can try several different seeds to attempt to obtain superior fitting results. The seeds that lead to the best fits for a design may change if the design changes. Also, changing the seed may or may not result in a better fit. Therefore, specify a seed only if the Fitter is not meeting timing requirements by a small amount. *Note:* You can also use the Design Space Explorer II (DSEII) to sweep complex flow parameters, including the seed, in the Quartus Prime software to optimize design performance.
Logic Cell Insertion	Allows the Fitter to automatically insert buffer logic cells between two nodes without altering the functionality of the design. The Compiler creates buffer logic cells from unused logic cells in the device. This option also allows the Fitter to duplicate a logic cell within a LAB when there are unused logic cells available in a LAB. Using this option can increase compilation time. The default setting of **Auto** allows these operations to run when the design requires them to fit the design. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.
MLAB Add Timing Constraints for Mixed-Port Feed-Through Mode Setting Don't Care	Specifies whether the Timing Analyzer evaluates timing constraints between the write and the read operations of the MLAB memory block. Performing a write and read operation simultaneously at the same address might result in metastability issues because no timing constraints between those operations exist by default. Turning on this option introduces timing constraints between the write and read operations on the MLAB memory block and

continued...

Option	Description
	thereby avoids metastability issues. However, turning on this option degrades the performance of the MLAB memory blocks. If your design does not perform write and read operations simultaneously at the same address, you do not need to set this option.
Number of Example Nodes Reported in Fitter Messages	Allows you to specify the maximum number of example nodes Fitter messages should display.

Table 55. Advanced Fitter Settings (5 of 8)

Option	Description
Optimize Design for Metastability	This setting improves the reliability of the design by increasing its Mean Time Between Failures (MTBF). When you enable this setting, the Fitter increases the output setup slacks of synchronizer registers in the design. This slack can exponentially increase the design MTBF. This option only applies when using the Timing Analyzer for timing-driven compilation. Use the Timing Analyzer `report_metastability` command to review the synchronizers detected in your design and to produce MTBF estimates.
Optimize Hold Timing	Directs the Fitter to optimize hold time within a device to meet timing requirements and assignments. The following settings are available: • **I/O Paths and Minimum TPD Paths—**directs the Fitter to meet the following timing requirements and assignments: — t_H from I/O pins to registers. — Minimum t_{CO} from registers to I/O pins. — Minimum t_{PD} from I/O pins or registers to I/O pins or registers. • **All Paths**—directs the Fitter to meet the following timing requirements and assignments: — t_H from I/O pins to registers. — Minimum t_{CO} from registers to I/O pins. — Minimum t_{PD} from I/O pins or registers to I/O pins or registers. When you disable the **Optimize Timing** logic option, the **Optimize Hold Timing** option is not available.
Optimize IOC Register Placement for Timing	Specifies whether the Fitter optimizes I/O pin timing by automatically packing registers into I/Os to minimize delays. • **Normal**—the Fitter opportunistically packs registers into I/Os that should improve I/O timing. • **Pack All I/O Registers**— the Fitter aggressively packs any registers connected to input, output, or output enable pins into I/Os, unless prevented by your constraints or other legality restrictions. • **Off**—performs no periphery to core optimization.
Optimize Multi-Corner Timing	Directs the Fitter to consider all timing corners during optimization to meet timing requirements. These timing delay corners include both fast-corner timing and slow-corner timing. By default, this option is **On**, and the Fitter optimizes designs considering multi-corner delays in addition to slow-corner delays. When this option is **Off**, the Fitter optimizes designs considering only slow-corner delays from the slow-corner timing model (slowest manufactured device for a given speed grade, operating in low-voltage conditions). Turning this option **On** typically creates a more robust design implementation across process, temperature, and voltage variations. When you turn **Off** the **Optimize Timing** option, the **Optimize Multi-Corner Timing** option is not available.
Optimize Timing	Specifies whether the Fitter optimizes to meet the maximum delay timing requirements (for example, clock cycle time). By default, this option is set to **Normal compilation**. Turning this option **Off** helps fit designs that with extremely high interconnect requirements. Turning this option **Off** can also reduce compilation time at the expense of timing performance (because the Fitter ignores the design's timing requirements). If this option is **Off**, other Fitter timing optimization options have no effect (such as **Optimize Hold Timing**).

Table 56. Advanced Fitter Settings (6 of 8)

Option	Description
Periphery to Core Placement and Routing Optimization	Specifies whether the Fitter should perform targeted placement and routing optimization on direct connections between periphery logic and registers in the FPGA core. The following options are available: • **Auto**—the Fitter automatically identifies transfers with tight timing windows, places the core registers, and routes all connections to or from the periphery. The Fitter performs these placement and routing decisions before the rest of core placement and routing. This sequence ensures that these timing-critical connections meet timing, and also avoids routing congestion. • **On**— the Fitter optimizes all transfers between the periphery and core registers, regardless of timing requirements. Do not set this option to **On** globally. Instead, use the Assignment Editor to assign optimization to a targeted set of nodes or entities. • **Off**—the Fitter performs no periphery to core optimization. *Note:* Only available for Arria 10 and Cyclone 10 GX devices.
Physical Placement Effort	Controls how much effort the Fitter spends during advanced physical placement optimization. **High** and **Maximum** effort settings result in additional compile time to further optimization the placement solution.
Placement Effort Multiplier	Specifies the relative time the Fitter spends in placement. The default value is 1.0, and legal values must be greater than 0. Specifying a floating-point number allows you to control the placement effort. A higher value increases CPU time but may improve placement quality. For example, a value of '4' increases fitting time by approximately 2 to 4 times but may increase quality.
Power Optimization During Fitting	Directs the Fitter to perform optimizations targeted at reducing the total power devices consume. The available settings for power-optimized fitting are: • **Off**—performs no power optimizations. • **Normal compilation**—performs power optimizations that are unlikely to adversely affect compilation time or design performance. • **Extra effort**—performs additional power optimizations that might affect design performance or result in longer compilation time.

Table 57. Advanced Fitter Settings (7 of 8)

Option	Description
Programmable Power Maximum High-Speed Fraction of Used LAB Tiles	Sets the upper limit on the fraction of the high-speed LAB tiles. Legal values must be between 0.0 and 1.0. The default value is 1.0. A value of 1.0 means that there is no restriction on the number of high-speed tiles, and the Fitter uses the minimum number needed to meet the timing requirements of your design. Specifying a value lower than 1.0 might degrade timing quality, because some timing critical resources might be forced into low-power mode.
Programmable Power Technology Optimization	Controls how the Fitter configures tiles to operate in high-speed mode or low-power mode. The following options are available: • **Automatic**—specifies that the Fitter minimizes power without sacrificing timing performance. • **Minimize Power Only**—specifies that the Fitter sets the maximum number of tiles to operate in low-power mode. • **Force All Used Tiles to High Speed**—specifies that the Fitter sets all used tiles to operate in high-speed mode. • **Force All Tiles with Failing Timing Paths to High Speed**—sets all failing paths to high-speed mode. For designs that meet timing, the behavior of this setting is similar to the **Automatic** setting. For designs that fail timing, all paths with negative slack are put in high-speed mode. This mode likely does not increase the speed of the design, and it may increase static power consumption. This mode may assist in determining which logic paths need to be re-designed to close timing.

continued...

Option	Description
	Note: Only available for Arria 10 and Cyclone 10 GX devices.
Router Timing Optimization Level	Controls how aggressively the router tries to meet timing requirements. Setting this option to **Maximum** can increase design speed slightly, at the cost of increased compile time. Setting this option to **Minimum** can reduce compile time, at the cost of slightly reduced design speed. The default value is **Normal**.

Table 58. Advanced Fitter Settings (8 of 8)

Option	Description
Synchronizer Identification	Specifies how the Compiler identifies synchronization register chain registers for metastability analysis. A synchronization register chain is a sequence of registers with the same clock with no fan-out in between, which is driven by a pin or logic from another clock domain. The following options are available: • **Off**—the Timing Analyzer does not identify the specified registers, or the registers within the specified entity, as synchronization registers. • **Auto**—the Timing Analyzer identifies valid synchronization registers that are part of a chain with more than one register that contains no combinational logic. Use the **Auto** setting to generate a report of possible synchronization chains in your design. • **Forced if Asynchronous**—the Timing Analyzer identifies synchronization register chains if the software detects an asynchronous signal transfer, even if there is combinational logic or only one register in the chain. • **Forced**—the Timing Analyzer identifies the specified register, or all registers within the specified entity, as synchronizers. Do not apply the **Forced** option to the entire design as it identifies all registers in the design as synchronizers. The Fitter optimizes the registers that it identifies as synchronizers for improved Mean Time Between Failure (MTBF), provided that you enable **Optimize Design for Metastability**. If a synchronization register chain is identified with the **Forced** or **Forced if Asynchronous** option, then the Timing Analyzer reports the metastability MTBF for the chain when it meets the design timing requirements.
Treat Bidirectional Pin as Output Pin	Specifies that the Fitter treats the bidirectional pin as an output pin, meaning that the input path feeds back from the output path.
Use Checkered Pattern as uninitialized RAM Content	Loads a checkered pattern as initial RAM content into all RAM blocks without specified RAM content that supports content initialization. Turning on this option does not affect simulation, which may cause on-chip behavior to differ from simulation results.
Weak Pull-Up Resistor	Enables the weak pull-up resistor when the device is operating in user mode. This option pulls a high-impedance bus signal to VCC. Do not enable this option simultaneously with the **Enable Bus-Hold Circuitry** option. The Fitter ignores this option if you apply to anything other than a pin.

Other Assignments

```
set_global_assignment –name ERROR_ON_INVALID_ENTITY_NAME
```

The software ignores `.qsf` and `.qip` assignments where the entity field is not a name that exists in the design and generates a warning. If you set `ERROR_ON_INVALID_ENTITY_NAME` to `ON`, the software generates these warnings as errors.

1.19. Design Compilation Revision History

This document has the following revision history.

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Added Precompiled Component (PCC) flow to *Compilation Flows*. • Updated images and added information about the "RTL Analysis Debug Mode" in *Analysis & Elaboration Flow*. • Updated the images and enhanced the information in *Using the Node Finder*. • Revised the information in *Fast Forward Details Report*. • Added *Precompiled Component (PCC) Generation Flow*.
2023.12.04	23.4	• Enhanced the *Compiler Optimization Modes* topic with additional information. • Updated file properties image in *Registering the SDC-on-RTL SDC File* and *Using SDC-on-RTL Features*. • In *Design Synthesis*, added information about converting `.bdf` to `.v` or `.vhd` file, and updated the image. • Renamed *DNI-Based Compilation Flow* as *Analysis & Elaboration Flow*. • Added information about Early Timing Analysis flow in *Compilation Overview*. • Reorganized *DNI-Based Analysis & Elaboration Flow* and *Early Timing Analysis After Design Synthesis* sections. • Renamed *DNI-Based Node Finder* as *Using the Node Finder* • Added an image for Property Viewer showing constraints in *Exploring the RTL Analyzer*. • Reorganized most of the topics under *Design Netlist Infrastructure* and moved them into relevant sections of this chapter. • Removed the term "DNI" in the title and content of the following topics: — *Use Case Examples* — *Scripting Routine Tasks Using Tcl Commands* — *Traversing the Design Netlist Using Tcl Commands* • Revised the information and image in *Design Synthesis*. • Added information about SDC-on-RTL file in *Running Synthesis*. • Revised the image, added a note for "Parameter Settings by Entity Instance" and added information about SDC constraints in *Viewing Synthesis Reports*. • Revised the existing information in *Concurrent Analysis During Synthesis or Fitting*. • Added a note about version compatibility in *Importing a Version-Compatible Compilation Database*, and *Exporting a Design Partition*. • Renamed the topic *Compilation Monitoring* as *Compilation Monitoring Mode* and revised the topic entirely. • Revised *Enable Intermediate Fitter Snapshots* with additional information. • Added *Preparing for Design Synthesis*. • Removed *Early Timing Analysis After Design Synthesis* and merged its information with *Post-Synthesis Static Timing Analysis (STA)*.
2023.10.02	23.3	• Enhanced the instructions and removed "Beta" in *Design Netlist Infrastructure* and *Exploring the RTL Analyzer*. • Made minor revisions to the description in *Object Set Console*. • Updated "Viewing Unbundled Instances" section in *Bundled Instances*. • Completely revised the instructions in *Early Timing Analysis After Design Synthesis*. • Made minor updates to *Synopsys* Design Constraint (SDC) on RTL*. • Completely updated "RTL Analyzer" section and added additional information about Constraints viewer in *Inspecting SDC-on-RTL Constraints*.

continued...

Document Version	Quartus Prime Version	Changes
		• Added the following topics — *Entity-Based SDC-on-RTL* — *Using SDC-on-RTL Features* — *DNI-Based Node Finder* • Updated the commands and replaced SYN_SDC_FILE with SDC_FILE `-read_during_post_syn_and_not_post_fit_timing_analysis` in *Post-Synthesis Static Timing Analysis (STA)*. • Replaced SYN_SDC_FILE in *Types of SDC Files Used in the Quartus Prime Software*. • Reorganized the topics in the *Design Compilation* chapter per the DNI-based compilation dashboard. • Updated the compilation dashboard image in the following topics: — *Using the Compilation Dashboard* — *Concurrent Analysis During Synthesis or Fitting* — *Step 1: Run Register Retiming* — *Step 3: Run Fast Forward Compile* — *Fast Forward Compile By Hierarchy* • Added description for "Number of Congested Nets" column in *Global Router Congestion Hotspot Summary Report*. • Revised the descriptions of optimization modes in *Compiler Optimization Modes*. • Removed the topic *Connectivity Tracer*. • Updated the "Hierarchical Project Structure" image along with its description in *Compilation Hierarchy*.
2023.04.03	23.1	• In *Design Netlist Infrastructure (Beta)*, updated the images and added a note about the incompatibilities between classic and DNI compilation flows. • In *Exploring the RTL Analyzer (Beta)*, updated the images and improved their clarity. • Enhanced the *Sweep Hints Viewer* topic with additional information and images. • Enhanced the *Inspecting SDC-on-RTL Constraints* topic with additional information about Object Constraints viewer. • Revised the *Object Set Console* topic entirely. • Revised the *Auto-hide Unconnected Pins* topic entirely. • Renamed the topic *Early Timing Analysis (Beta)* to *Early Timing Analysis After Design Synthesis (Beta)* and revised the information and images. • Enhanced *Applying the SDC-on-RTL Constraints* with additional information about the Constraint Propagation Report. • Updated the images and revised some instructions in *Post-Synthesis Static Timing Analysis (STA)*. • Updated the product family name to "Intel Agilex 7." • Revised description of Fitter (Finalize) command for latest physical synthesis optimizations.
2022.12.19	22.4	• Added *Filtering*. • Added *Expand Connections*. • Revised *Object Set Console* with additional information and images.
2022.09.26	22.3	• Added *Early Timing Analysis (Beta)*. • Added *Synopsys* Design Constraint (SDC) on RTL*. • Added *Registering the SDC-on-RTL SDC File*. • Added *Applying the SDC-on-RTL Constraints*. • Added *Managing SDC-on-RTL Constraints*. • Added *Writing Constraints in SDC-on-RTL SDC Files*. • Added *Post-synthesis Static Timing Analysis (STA)*.

continued...

Document Version	Quartus Prime Version	Changes
		• Added *Types of SDC Files Used in the Quartus Prime Software*. • Added *Object Set Console*. • Added *Module Interfaces*. • Added *Connectivity Tracer*. • Added *DNI Use Case Examples*. • Added *Scripting Routine Tasks Using DNI Tcl Commands*. • Added *Traversing the DNI Netlist Using Tcl Commands*. • Added *Viewing Synthesis Dynamic Report*. • Split the topic *Instances Bundling and Auto-hiding Unconnected Pins* into separate topics *Bundled Instances* and *Auto-hide Unconnected Pins*. • Revised *Bundled Instances* with additional information. • Revised images in *Exploring the RTL Analyzer* and *Design Netlist Infrastructure (DNI)*.
2022.06.21	22.2	• Added *Design Netlist Infrastructure (DNI)*. • Added *Exploring the RTL Analyzer*. • Added *Module Interfaces*. • Added *Instances Bundling and Auto-hiding Unconnected Pins*.
2022.03.28	22.1	• Added *Compilation Monitoring*. • Added *Global Router Congestion Hotspot Summary Report*. • Revised *Full Compilation Flow*. • Added *Full Compilation Flow with Temporary Optimization Mode*.
2022.01.27	21.4	• Revised *Compiler Optimization Modes* topic to provide implication details of various modes.
2021.11.03	21.3	• Made a minor correction in *Reusing a Design Partition* step 4. • Removed the callouts from *Creating a Design Partition* topic.
2021.10.04	21.3	• Added *Preserving Signals for Monitoring and Debugging* topic. • Revised *Preserving Registers During Synthesis* topic for new debugging signal preservation assignments. • Revised *Viewing Synthesis Reports* topic to include new warnings summary reports. • Revised *Optimization Modes* topic to include new optimization modes. • Revised *VHDL Synthesis* Support to include VHDL 2019 support. • Revised *VHDL Input Settings (Settings Dialog Box)* topic to include VHDL 2019 support. • Added *VHDL-2019 Conditional Analysis* topic.
2021.06.21	21.2	• Added *Version-Compatible Compilation Database Support* table.
2021.03.29	21.1	• Added Check Unregistered Ports report to "Validating Timing Constraints with Snapshot Viewer" topic. • Updated "Running Snapshot Viewer" topic to indicate the reports that are available after the final snapshot. • Removed reference to Rapid Recompile from "Enable Intermediate Filter Snapshots". Support for Rapid Recompile has been removed. • Added information to "Using the Compilation Dashboard" to indicate that an interrupted compilation flow can be resumed.
2020.12.14	20.3	• Corrected typo in "Automatic Gated Clock Conversion" topic.
2020.11.09	20.3	• Added new "Integrating Other EDA Tools" topic. • Added new "Generating a VQM Netlist for Other EDA Tools" topic.
2020.09.28	20.3	• Added references to ECO Compilation Flow. • Removed references to deprecated Early Place Compiler flow.

continued...

Document Version	Quartus Prime Version	Changes
2020.05.08	20.1	• Added note about programming file differences between versions to "Compilation Overview" topic.
2020.04.13	20.1	• Added new "Fast Forward Compile by Hierarchy" topic. • Added new assignment to "Fitter Settings Reference" topic.. • Updated "Verilog and SystemVerilog Synthesis Support" topic for SystemVerilog 2012 support. • Added programming file generation support for Intel Agilex devices. • Added "Analyzing with the Snapshot Viewer" topic. • Added "Running the Snapshot Viewer" topic. • Added "Analyzing Failing Paths with Snapshot Viewer" topic. • Added "Analyzing Clocking with Snapshot Viewer" topic. • Added "Analyzing High Fan-Out Nets with Snapshot Viewer" topic. • Added "Analyzing Constraints with Snapshot Viewer" topic. • Added "Analyzing Congestion with Snapshot Viewer" topic. • Removed *Early Place Flow* • Removed *Synthesis Reports* figure and table. • Removed *Heat-Map in Global Signal Visualization Report* figure • Changed sentence in *Fast Forward Compilation* to *The Fitter automatically retimes registers across RAM and DSP blocks* from *The Fitter does not automatically retime registers across RAM and DSP blocks*. • Added more **Preservation Level** information to *Design Partition* table.
2019.10.20	19.3	• Added "Automatic Gated Clock Conversion" topic. • Updated "Fractal Synthesis Optimization" and "Enabling or Disabling Fractal Synthesis" topics for automated fractal synthesis for small multipliers.
2019.09.30	19.3	• Added support for Intel Agilex devices throughout. • Added "Global Signal Visualization Report" topic. • Added "Global Router Wire Utilization Map" topic. • Added "Fast Preserve Option" topic. • Reordering of some topics to match design flow.
2019.07.02	19.1	• Made minor changes in "Fractal Synthesis Optimization" topic. • Added a note in step 3a of "Running Synthesis" about enabling fractal synthesis project-wide. • Added details about synthesis of PRESERVE_FANOUT_FREE_NODE to "Partial Reconfiguration Design Guidelines." • Corrected typo in "Step 3: Run Fast Forward Compile and Hyper-Retiming." • Removed "Enabling Timing-Driven Synthesis" topic.
2019.04.01	19.1	• In "Running Synthesis", removed a step about enabling fractal synthesis project-wide. • Updated the "Fractal Synthesis Optimization" topic to describe signed multiplication feature that is now supported by multiplier regularization and arithmetic packing algorithms.
2019.01.03	18.1.0	• Added snapshot description to "Compilation Overview" and linked to content from "Exporting a Design Partition" and "Exporting a Version-Compatible Compilation Database."

continued...

Document Version	Quartus Prime Version	Changes
2018.10.19	18.1	• Described dependency of Rapid Recompile on Enable Intermediate Fitter Snapshots option.
2018.09.24	18.1	• Described option to enable or disable intermediate Fitter snapshots and updated descriptions of compilation flows and dashboard accordingly. • Added "Exporting Compilation Results section and subtopics." • Described migration of full chip database in "Exporting a Version-Compatible Compilation Database" topic. • Described automated `.qdb` partition export in "Exporting a Design Partition" topic. • Described viewing QDB file metadata in "Viewing Quartus Database File Information." • Added "Fractal Synthesis Optimization" topic and updated "Running Synthesis" topic steps for new option. • Described new Compiler Optimization Modes and described notice that appears for extended optimization modes added via `.qsf`. • Described new Global Signal Visualization Report. • Added "Factors Affecting Compilation Results" topic. • Added "Using the Compilation Dashboard" topic. • Added description of Enable Auto-Pipelining setting. • Added description of Enable Formal Verification Support to "Advanced Synthesis Settings." • Added description of Report PR Initial Values as Errors option to "Advanced Synthesis Settings." • Added description of Size of the Latch Report option to "Advanced Synthesis Settings." • Added description of Size of the PR Initial Conditions Report option to "Advanced Synthesis Settings." • Added description of Advanced Physical Synthesis option to "Fitter Settings Reference." • Added description of Allow DSP Retiming option to "Fitter Settings Reference." • Added description of Allow Early Global Retiming in the Fitter option to "Fitter Settings Reference." • Added description of Allow Hyper-Aware Register Chain Area Optimizations in the Fitter option to "Fitter Settings Reference." • Added description of Allow RAM Retiming option to "Fitter Settings Reference." • Added description of Number of Example Nodes Reported in Fitter Messages option to "Fitter Settings Reference." • Added description of Physical Placement Effort option to "Fitter Settings Reference." • Added description of Use Checkered Pattern as uninitialized RAM Content option to "Fitter Settings Reference." • Updated description of Safe State Machine option for Auto setting. • Removed support for Ignore ROW GLOBAL Buffers option. • Removed support for Ignore CARRY Buffers option. • Removed support for Ignore CASCADE Buffers option.
2018.05.07	18.0	• Updated Optimization Modes topic to add Compile Time (Aggressive). • Relocated concurrent analysis content from the *Early Place Flow* topic to a new *Concurrent Analysis During Synthesis or Fitting* topic. • Rapid Recompile now supports Stratix 10 devices. • Enhanced description of Retime Stage Reports. • Enhanced description of Retime Stage to include classic register retiming.

Table 59. Document Revision History

Date	Version	Changes
2017.11.06	17.1.0	• Added support for Stratix 10 Hyper-Aware design flow, Hyper-Retiming, Fast Forward compilation, and Fast Forward Viewer. • Added Advanced HyperFlex Settings topic. • Added Retiming Restrictions and Workarounds topic. • Added statement about Fast Forward compilation support for retiming across RAM and DSP blocks. • Added Concurrent Analysis topic. • Added Analyzing Fitter Snapshots topic. • Added Compilation Dashboard Early Place stage control image. • Added Running late_place After Early Place topic. • Updated for latest Intel naming conventions.
2017.05.08	17.0.0	• Added reference to initial compilation support for Cyclone 10 GX devices. • Described concurrent analysis following Early Place. • Updated Compilation Dashboard images for Timing Analyzer, Report, Setting, and Concurrent Analysis controls. • Updated description for Auto DSP Block Replacement in Advanced Synthesis Settings. • Updated Advanced Fitter Settings for Allow Register Retiming, and for removal of obsolete SSN Optimization option. • Added Prevent Register Retiming topic. • Added Preserve Registers During Synthesis topic. • Removed limitation for **Safe State Machine** logic option. • Added references to Partial Reconfiguration and Block-Based Design Flows.
2016.10.31	16.1.0	• Implemented Intel re-branding. • Described Compiler snapshots and added Analyzing Snapshot Timing topic. • Updated project directory structure diagram. • Described new Fitter stage menu commands and reports. • Added description of Early Place Flow, Implement Flow, and Finalize Flow. • Added description of Incremental Optimization in the Fitter. • Reorganized order of topics in chapter. • Removed deprecated **Per-Stage Compilation (Beta)** Compilation Flow.
2016.05.03	16.0.0	• Added description of Fitter Plan, Place and Route stages, reporting, and optimization. • Added **Per-Stage Compilation (Beta)** Compilation Flow • Added Compilation Dashboard information. • Removed support for **Safe State Machine** logic option. Encode safe states in RTL. • Added Generating Dynamic Synthesis Reports topic. • Updated Quartus project directory structure.
2015.11.02	15.1.0	• First version of document.

2. Reducing Compilation Time

You can employ various techniques to reduce the time required for synthesis and fitting in the Quartus Prime Compiler.

2.1. Factors Affecting Compilation Results

Almost any change to the following project settings, hardware, or software can impact the results from one compilation to the next.

- Project Files—changes to project settings (`.qsf`, `quartus2.ini`), design files, and timing constraints (`.sdc`) can change the results.
- Any setting that changes the number of processors during compilation can impact compilation results.
- Hardware—CPU architecture, not including hard disk or memory size differences. Windows XP x32 results are not identical to Windows XP x64 results. Linux x86 results is not identical to Linux x86_64.
- Quartus Prime Software Version—including build number and installed interim updates. Click **Help > About** to obtain this information.
- Operating System—Windows or Linux operating system, excluding version updates. For example, Windows XP, Windows Vista, and Windows 7 results are identical. Similarly, Linux RHEL, CentOS 4, and CentOS 5 results are identical.

2.2. Strategies to Reduce the Overall Compilation Time

You can use the following strategies to reduce the overall time required to compile your design:

- Running the ECO Compilation Flow on page 187
- Enabling Multi-Processor Compilation on page 188
- Using Block-Based Compilation on page 191

2.2.1. Running the ECO Compilation Flow

The Quartus Prime Pro Edition software supports last-minute, targeted design changes (also known as engineering change orders (ECOs)), even after you fully compile the design. ECOs typically occur during the design verification stage. Refer to the *Quartus Prime Pro Edition User Guide: Design Optimization* for details.

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Optimization

2.2.2. Enabling Multi-Processor Compilation

The compiler can detect and use multiple processors to reduce total compilation time. By default, the compiler uses the setting specified under **Parallel Compilation** in the **Processing** page of the **Options** dialog box. To reserve some processors for other tasks, specify the maximum number of processors the software must use.

Using multiple processor cores provides several benefits for improving software performance as follows:

- **Faster execution**: The Quartus Prime software can support up to 24 processors, which means the software can run algorithms in parallel. This technique reduces the compilation time by up to 10% on systems with two processing cores and up to 20% on systems with four processors. When running timing analysis independently, two processors reduce the timing analysis time by an average of 10%. This reduction reaches an average of 15% when using four processors.
- **Increased throughput**: With more processing cores, systems can execute multiple tasks simultaneously, which means the software can handle more requests simultaneously. The Quartus Prime software may not necessarily utilize all the processors specified during compilation. The software has the flexibility to scale its usage to use up to the maximum number of processors specified. To achieve the highest possible throughput, Intel recommends using a system equipped with at least four processing cores, allowing the software to take full advantage of the available computing resources.
- **Reduced latency**: With multiple processing cores, the software responds more quickly to your requests, improving the overall experience. The software never uses more than the specified number of processors, so you can work on other tasks in parallel without slowing down your computer.
- **More efficient resource utilization**: By distributing tasks across multiple processor cores, the Quartus Prime software can use available resources more efficiently, reducing the overall cost of running the software.

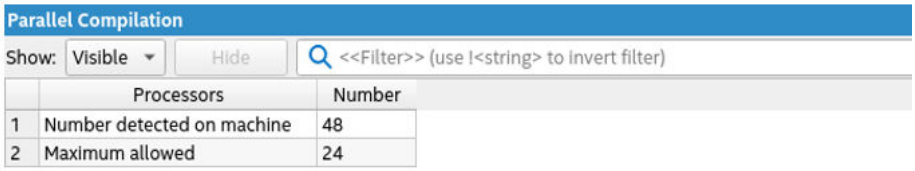
The use of multiple processors does not affect the quality of the fit. The fit is the same and deterministic for a given Fitter seed and given **Maximum processors allowed** setting on a specific design. This remains true regardless of the target system and the number of available processors. Different **Maximum processors allowed** specifications produces different results of the same quality. The impact is similar to changing the Fitter seed setting.

To enable multiprocessor compilation, follow these steps:

1. Open or create an Quartus Prime project.
2. Click **Assignments ➤ Settings ➤ Compilation Process Settings**.
3. Under **Parallel compilation**, specify options for the number of processors the compiler uses.

   View the number of processors detected on your system in the Parallel Compilation report after compilation ends.

Figure 139. Parallel Compilation Report



The following is the QSF setting that controls the maximum number of processors. If this line is in your project's QSF file, do not specify it again.

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS <value>
```

In this case, <*value*> is an integer from 1 to 24.

If you want the Quartus Prime software to detect the number of processors and use all the processors for the compilation, include the following Tcl command in your script:

```
set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
```

Note:
- Using multiple processor cores can help the Quartus Prime software run faster, handle more tasks, and provide a better user experience. However, other factors, such as memory bandwidth or I/O bottlenecks, may limit performance. So, you must consider specific requirements and constraints of each project when deciding how many processor cores to use.
- The compiler detects Intel Hyper-Threading® Technology (Intel HT Technology) as a single processor. If your system includes a single processor with Intel HT Technology, set the number of processors to one. Set the number of processors according to the number of physical processors in your system,

The following are other factors that affect performance in the Quartus Prime software:

2.2.2.1. Processor Base Clock Frequency

Using faster processor cores provides several advantages, including:

- **Faster execution**: With faster processor cores, the Quartus Prime software can execute tasks more quickly, which can improve overall system performance and reduce the time required to complete complex calculations, algorithms, and data processing tasks.
- **Improved multitasking**: Faster processor cores can improve the ability of the Quartus Prime software to handle multiple tasks simultaneously, reducing the risk of system slowdowns when running several algorithms simultaneously.
- **Quicker start-up and shutdown times**: Faster processors can reduce the time required for the Quartus Prime software to open and close side applications, such as Timing Analyzer, Platform Designer, and more. This can improve your productivity and reduce downtime.

For shortest compile times, you should choose processors with the highest base clock frequency available and prioritizing higher CPU frequency over having more cores. While additional cores can be beneficial for tasks that are highly parallelizable,

focusing on higher frequency ensures faster execution of individual threads, leading to improved responsiveness and overall performance of the Quartus Prime Pro Edition software.

2.2.2.2. Random Access Memory (RAM)

There are several advantages to having more RAM, including:

- **Improved performance**: With more RAM, the Quartus Prime software can store more data in memory, thereby reducing the need for the system to access the slower hard drive for frequently used data. This can result in faster application launch times, quicker compilation times, and faster overall system performance.
- **Better multitasking**: More RAM allows a computer to handle more processes simultaneously without slowing down. This is particularly important when you want to simultaneously use the Quartus Prime software and other applications or programs.
- **Improved productivity**: More RAM can improve the productivity of the Quartus Prime software by reducing the time required for complex tasks, such as compilation.

Note:

- The Quartus Prime Pro Edition Software and Device Support Release Notes provides valuable information regarding the software's system requirements and recommended configurations. Consult the release notes to determine the minimum required RAM for your computer to optimize performance.
- To fully leverage the system's capabilities, Intel recommends utilizing the maximum number of DIMM slots available. This configuration provides ample memory capacity and bandwidth, allowing for efficient handling of complex designs.

In addition, by tracking the virtual memory utilization, you can identify potential performance issues and optimize your system for better efficiency when using the Quartus Prime Pro Edition software. Consult the peak virtual memory from a previous compile by viewing the flow report or the Flow Elapsed Time report category from the compilation report in the compilation dashboard.

The following example Flow Elapsed Time report shows the peak virtual memory:

Figure 140. Flow Elapsed Time Report

Flow Elapsed Time

Show: Visible | Hide | <<Filter>> (use !<string> to invert filter)

	Module Name	Elapsed Time	Peak Virtual Memory
1	Synthesis	00:00:38	1513 MB
2	Fitter	00:08:46	13709 MB
3	Timing Analyzer	00:00:35	3749 MB
4	Total	00:09:59	--

Running numerous processes concurrently can consume a significant amount of memory resources and potentially lead to performance issues. so Intel recommends considering the peak virtual memory used for each project and avoiding multiple compilations that exceed the available memory capacity of your computer. This helps prevent congestion in the RAM memory.

Overall, by maximizing the RAM capacity, you can ensure smooth and optimal performance and better multitasking to enhance productivity during resource-intensive tasks, enabling faster processing, reduced latency, and improved productivity of the Quartus Prime Pro Edition software.

2.2.2.3. Storage

Storage can be a factor that limits performance in the Quartus Prime software since the Quartus Prime Pro Edition software reads source files and constraints and reads/writes to the database. For best results, Intel recommends the following to ensure consistent system performance:

- Downloading all the necessary files from the network.
- Completing the compilation process using local disks.
- Uploading the finished results back to the network by avoiding storage latency (varies on each system).

Intel recommends prioritizing the selection of the fastest affordable SSD drive for your storage needs. Opting for an SSD over traditional disk drives significantly enhances load and save times and greatly improves overall operating system performance. SSDs offer superior speed and responsiveness, making them an excellent choice to ensure the best performance experience when using the Quartus Prime Pro Edition software.

2.2.3. Using Block-Based Compilation

During the design process, when making minor modifications to a design, recompiling the entire design can result in longer compilation times than anticipated. This is because every time you recompile a design following a change, the compiler may apply global optimizations to enhance resource utilization and timing performance, thus extending the compilation time. By employing a block-based flow in the Quartus Prime Pro Edition software, you can isolate functional blocks that meet placement and timing requirements from others still undergoing change and optimization. By isolating functional blocks into partitions, the results and performance of unaltered logic within a design are maintained so you can apply optimization techniques to selected areas and only compile those areas. This approach can significantly diminish design compilation time, enabling several iterations per day and facilitating more efficient achievement of timing closure.

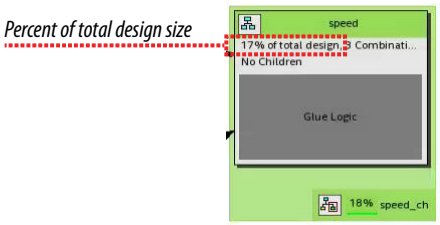
When using block-based compilation, you can enable the **Fast Preserve** option, which masks the partition netlist during the fitter initialization to use only the logic that interfaces with the rest of the design present at the partition boundary during compilation. By implementing this approach, the time required for the compiler to perform synthesis, place, route, and partition is effectively reduced. Consequently, the overall compilation process becomes more efficient, enabling faster generation of the necessary configurations for the partition. This reduction in compilation time allows for quicker iterations and facilitates the timely completion of the design implementation phase.

To create partitions dividing functional blocks:

1. In the Design Partition Planner, identify blocks of a size suitable for partitioning.

   A partition generally represents roughly 15 to 20% of the total design size. You should use the information area below the bar at the top of each entity.

Figure 141. Entity representation in the Design Partition Planner



2. Extract and collapse entities as necessary to achieve stand-alone blocks.
3. For each entity of the desired size containing related blocks of logic, right-click the entity and click **Create Design Partition** to place that entity in its own partition.

   The goal is to achieve partitions containing related blocks of logic.
4. To enable the **Fast Preserve** option that simplifies the logic of the preserved partition to only interface logic during compilation, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Incremental Compile ➤ Fast Preserve**.

Intel recommends consulting the *Quartus Prime Pro Edition User Guide: Block-Based Design* to gain in-depth knowledge about block-based designs. This guide serves as a comprehensive resource that provides detailed information, instructions, and explanations related to the Quartus Prime Pro Edition software.

Related Information

Intel Quartus Prime Pro Edition User Guide: Block-Based Design

2.3. Reducing Synthesis Time and Synthesis Netlist Optimization Time

You can reduce synthesis time without affecting the Fitter time by reducing your use of netlist optimizations. For tips on reducing synthesis time when using third-party EDA synthesis tools, refer to your synthesis software's documentation.

2.3.1. Settings to Reduce Synthesis Time and Synthesis Netlist Optimization Time

Synthesis netlist and physical synthesis optimization settings can significantly increase the overall compilation time for large designs. Refer to Analysis and Synthesis messages to determine the length of optimization time.

If your design already meets performance requirements without synthesis netlist or physical synthesis optimizations, turn off these options to reduce compilation time. If you require synthesis netlist optimizations to meet performance, optimize partitions of your design hierarchy separately to reduce the overall time spent in Analysis and Synthesis.

2.3.2. Use Appropriate Coding Style to Reduce Synthesis Time

Your HDL coding style can also affect the synthesis time. For example, if you want to infer RAM blocks from your code, you must follow the guidelines for inferring RAMs. If RAM blocks are not inferred properly, the software implements those blocks as registers.

If you are trying to infer a large memory block, the software consumes more resources on the FPGA. This can cause routing congestion and increases compilation time significantly. If you see high routing utilization in certain blocks, review the code for such blocks.

2.4. Reducing Placement Time

The time required to place a design depends on two factors:

- The number of ways the logic in your design can be placed in the device.
- The settings that control the amount of effort required to find a good placement.

You can reduce the placement time by changing the settings for the placement algorithm. If you have enabled a higher performance effort compiler optimization mode, you can try reducing the effort setting and observe how it trades off runtime and quality of results (QoR).

You can also observe the placement of major logic blocks in your design (over multiple compiles) to see whether the major blocks tend to get placed in the same places in the floorplan between the compiles. Suppose major blocks get placed in different places in some compiles. If those placements correlate with good QoR, create Logic Lock regions to ensure the blocks are placed in those regions with good QoR, which should help reduce compile time.

Sometimes there is a trade-off between placement time and routing time. Routing time can increase if the placer does not run long enough to find a good placement. When you reduce placement time, ensure that it does not increase routing time and negate the overall time reduction.

2.4.1. Placement Effort Multiplier Settings

The **Placement Effort Multiplier** option controls how much time the Fitter spends in placement. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** and specify a value for Placement Effort Multiplier.

The default value is 1.0, and valid values are greater than 0. Specifying a floating-point number allows you to control the placement effort. A lower value decreases the CPU time but may reduce placement quality. You cannot directly increase the **Placement Effort Multiplier** to a value greater than 1.0. Use predefined Optimization Mode settings to increase placement effort for improved timing optimization.

2.5. Reducing Routing Time

The routing time is usually not a significant amount of the compilation time. The time required to route a design depends on three factors: the device architecture, the placement of your design in the device, and the connectivity between different parts of your design.

If your design requires a long time to route, perform one or more of the following actions:

- Check for routing congestion.
- Turn off **Fitter Aggressive Routability Optimization**.

2.5.1. Identifying Routing Congestion with the Chip Planner

To identify areas of routing congestion in your design:

1. Click **Tools ➤ Chip Planner**.
2. To view the routing congestion in the Chip Planner, double-click the **Report Routing Utilization** command in the **Tasks** list.
3. Click **Preview** in the **Report Routing Utilization** dialog box to preview the default congestion display.
4. Change the **Routing utilization type** to display congestion for specific resources. The default display uses dark blue for 0% congestion and red for 100%.
5. Adjust the slider for **Threshold percentage** to change the congestion threshold level.

The Quartus Prime compilation messages contain information about average and peak interconnect usage. Peak interconnect usage over 75%, or average interconnect usage over 60% indicate possible difficulties fitting your design. Similarly, peak interconnect usage over 90%, or average interconnect usage over 75%, indicate a high chance of not getting a valid fit.

2.5.1.1. Areas with Routing Congestion

Even if average congestion is not high, the design may have areas where congestion is high in a specific type of routing. You can use the Chip Planner to identify areas of high congestion for specific interconnect types.

- You can change the connections in your design to reduce routing congestion
- If the area with routing congestion is in a Logic Lock region or between Logic Lock regions, change or remove the Logic Lock regions and recompile your design.
  - If the routing time remains the same, the time is a characteristic of your design and the placement
  - If the routing time decreases, consider changing the size, location, or contents of Logic Lock regions to reduce congestion and decrease routing time.

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Optimization

2.5.1.2. Congestion due to HDL Coding style

Sometimes, routing congestion may be a result of the HDL coding style used in your design. After identifying congested areas using the Chip Planner, review the HDL code for the blocks placed in those areas to determine whether you can reduce interconnect usage by code changes.

2.6. Reducing Static Timing Analysis Time

If you are performing timing-driven synthesis, the Quartus Prime software runs the Timing Analyzer during Analysis and Synthesis.

The Quartus Prime Fitter also runs the Timing Analyzer during placement and routing. If there are incorrect constraints in the Synopsys Design Constraints File (`.sdc`), the Quartus Prime software may spend unnecessary time processing constraints several times.

- If you do not specify false paths and multicycle paths in your design, the Timing Analyzer may analyze paths that are not relevant to your design.
- If you redefine constraints in the `.sdc` files, the Timing Analyzer may spend additional time processing them. To avoid this situation, look for indications that Synopsis design constraints are being redefined in the compilation messages, and update the `.sdc` file.
- Ensure that you provide the correct timing constraints to your design, because the software cannot assume design intent, such as which paths to consider as false paths or multicycle paths. When you specify these assignments correctly, the Timing Analyzer skips analysis for those paths, and the Fitter does not spend additional time optimizing those paths.

2.7. Setting Process Priority

It might be necessary to reduce the computing resources allocated to the compilation at the expense of increased compilation time. It can be convenient to reduce the resource allocation to the compilation with single processor machines if you must run other tasks at the same time.

Related Information

Processing Page (Options Dialog Box)
In Quartus Prime Help.

2.8. Reducing Compilation Time Revision History

Document Version	Quartus Prime Version	Changes
2023.12.04	23.4	• Enhanced the information in *Reducing Placement Time*. • Revised the information in *Placement Effort Multiplier Settings*.
2023.06.26	23.2	• Revised the information in *Enabling Multi-Processor Compilation* and *Using Block-Based Compilation*. • Added the following new topics: — *Processor Base Clock Frequency* — *Random Access Memory (RAM)* — *Storage*
2022.09.26	22.3	• Updated *Enabling Multi-Processor Compilation* topic for processor limit increase from 16 to 24.
2022.01.27	21.4	• Removed references to obsolete Compilation Time Advisor.
2021.11.03	21.3	• Made minor update to step 1 in *Using Block-Based Compilation*.
2021.10.04	21.3	• Removed "Disabling the Register Power-up Initialization".

continued...

Document Version	Quartus Prime Version	Changes
2021.03.29	21.1	• Support for Rapid Recompile has been removed, resulting in the following changes: — Removed reference to Rapid Recompile from "Strategies to Reduce the Overall Compilation Time". — Removed "Running Rapid Recompile" topic.
2020.09.28	20.3	• Added reference to ECO Compilation flow.
2019.11.11	19.3	• Added support for Fast Preserve option to "Using Block-Based Compilation" topic.
2019.09.30	19.3	• Added support for Intel Agilex devices throughout.
2019.07.02	19.1	Added the *Using the No-Register Initialization Flow* topic.
2018.10.19	18.1	• Described dependency of Rapid Recompile on Enable Intermediate Fitter Snapshots option.
2017.11.06	17.1	• Added topic: Using Block-Based Compilation.

Date	Version	Changes
2017.05.08	17.0.0	• Clarified impact of multiprocessor compilation on fit quality. • Removed reference to deprecated Fitter Effort Logic Option. • Removed section: *Preserving Routing with Incremental Compilation*.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	• Corrected typo in Using Parallel Compilation with Multiple Processors. • Removed information about deprecated physical synthesis options.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Added information about Rapid Recompile feature.
2014.08.18	14.0a10.0	Added restriction about smart compilation in Arria 10 devices.
June 2014	14.0.0	Updated format.
May 2013	13.0.0	Removed the “Limit to One Fitting Attempt”, “Using Early Timing Estimation”, “Final Placement Optimizations”, and “Using Rapid Recompile” sections. Updated “Placement Effort Multiplier Settings” section. Updated “Identifying Routing Congestion in the Chip Planner” section. General editorial changes throughout the chapter.
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	• Updated “Using Parallel Compilation with Multiple Processors”. • Updated “Identifying Routing Congestion in the Chip Planner”. • General editorial changes throughout the chapter.
December 2010	10.1.0	• Template update. • Added details about peak and average interconnect usage. • Added new section “Reducing Static Timing Analysis Time”. • Minor changes throughout chapter.
July 2010	10.0.0	Initial release.

3. Quartus Prime Pro Edition User Guide Design Compilation Archives

For the latest and previous versions of this user guide, refer to *Quartus Prime Pro Edition User Guide: Design Compilation*. If a software version is not listed, the guide for the previous software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Design Optimization

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

1. Design Optimization Overview

In the early stages of FPGA design development, you typically focus on meeting your timing requirement, resource usage, and power consumption goals. After meeting these basic goals, you can focus on optimizing performance.

Optimization of FPGA design performance requires a multi-dimensional approach to reduce resource use, critical path delays, power consumption, and runtime. The Quartus® Prime software provides various tools and techniques for performance optimization.

This chapter provides an overview of the initial techniques and settings in the Quartus Prime software that you can use to optimize your design results and achieve the highest performance in Intel FPGAs.

Related Information

Quartus Prime Pro Edition User Guide: Design Compilation

1.1. Initial FPGA Device Considerations

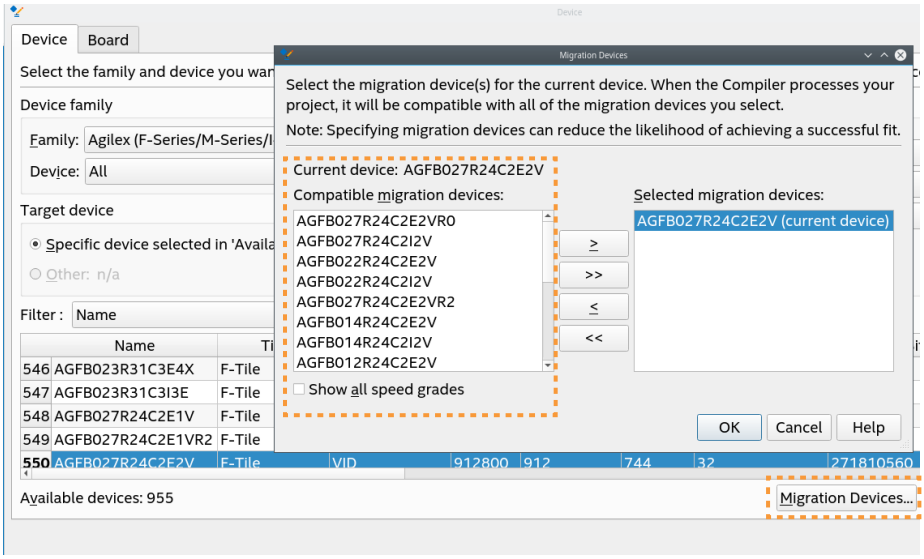
All Intel® FPGAs have a unique timing model that describes the delay information between all physical elements in the device, such as combinational adaptive logic modules, memory blocks, interconnects, and registers. The delay models comprise all valid combinations of operating condition delays for the target FPGA. The Timing Analyzer references these delay models in calculating performance during timing analysis. The device size and package determine pin-out and the resource availability. When selecting your target Intel FPGA device for your design, you must consider the performance specifications and resources available in the device meet the needs of your design.

1.1.1. Device Migration Considerations

If you anticipate that you might later migrate your design to a different target device later in the design cycle (for example, a larger or faster device), you must plan for this migration from the beginning of cycle. Planning for migration early helps you to minimize the complex design changes that you must make later to accommodate the new device.

When choosing a target FPGA device in the **Device** dialog box, you can click the **Migration Devices** button to view a list of all compatible devices.

Figure 1. Devices Available for Migration from Selected Device



Related Information

Migration Devices Dialog Box
In *Quartus Prime Help*

1.2. Initial Compiler Settings

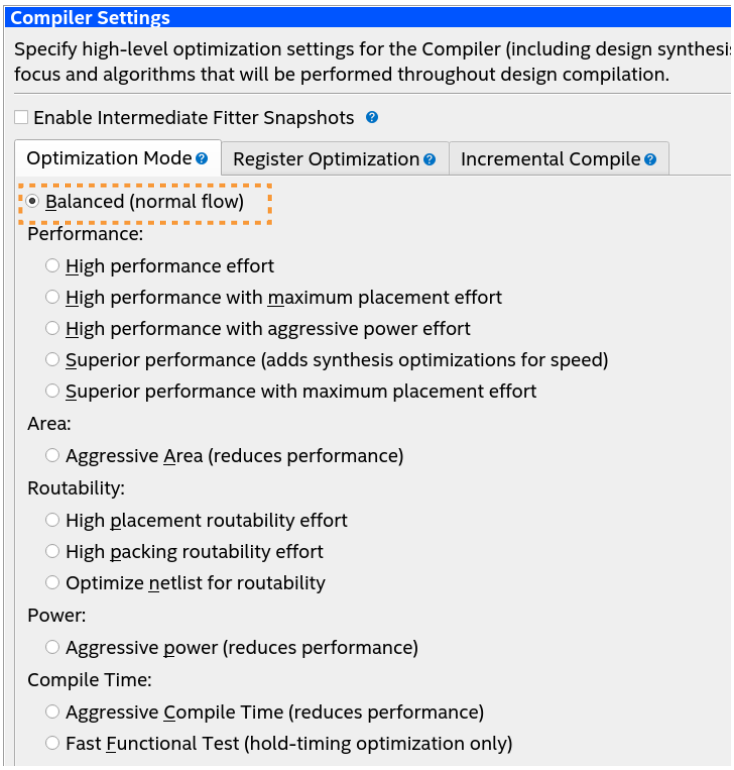
Your design compilation results can vary significantly, depending on the initial assignments and settings that you choose prior to compiling. The Quartus Prime software initial settings for compilation are set to provide a balanced trade-off between the time required for compilation, the device resource utilization, and the design timing performance.

You can easily adjust this trade-off to focus the Compiler's effort more on shortening the total compile time, reducing device resource utilization, or maximizing timing performance.

The initial FPGA device selection and Compiler settings have a very significant impact on design performance and optimization. You should also consider the following guidelines for specifying initial settings before compiling your design for the first time in the Quartus Prime software.

Figure 2. Compiler Optimization Mode Settings

Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Optimization Mode** to adjust the Compiler's effort on **Performance**, **Area**, **Routability**, or **Compile Time**.



1.2.1. Initial I/O Assignment Guidelines

The I/O standard and drive strength requirements that you specify for your design affect the I/O timing. Follow these guidelines when specifying initial I/O assignments:

- When specifying I/O assignments, specify an accurate I/O timing delay for timing analysis and Fitter optimizations.
- If the PCB layout does not indicate pin locations, then leave the pin locations unconstrained. This technique allows the Compiler to search for the best layout. Otherwise, make pin assignments to constrain the compilation appropriately.

Related Information

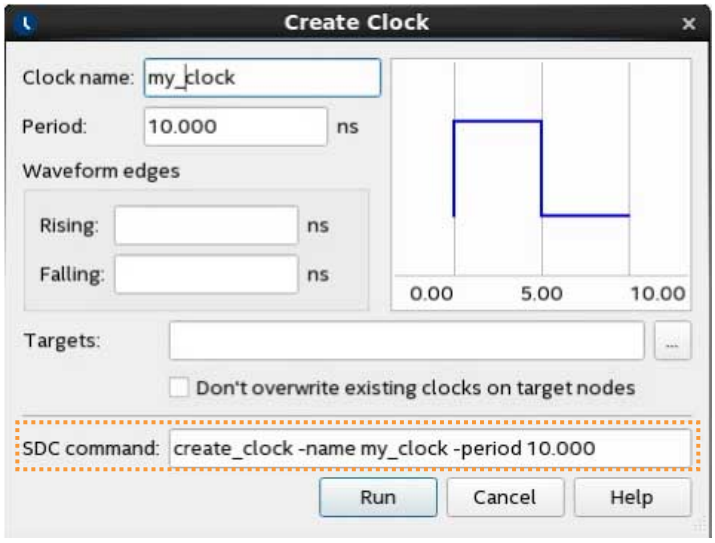
Quartus Prime Pro Edition User Guide: Design Constraints
In *Quartus Prime Pro Edition User Guide: Design Constraints*

1.2.2. Initial Timing Constraint Guidelines

Before running initial compilation or timing analysis, specify realistic timing requirements. Specifying more stringent timing requirements than the design requires causes the Compiler to expend effort to increase performance at the expense of resource usage, power utilization, or compilation time.

Click **Tools ➤ Timing Analyzer** then click the Constraints menu to enter constraints in the GUI, such as defining the clock signals. Alternatively, you can specify timing constraints directly in an `.sdc` file.

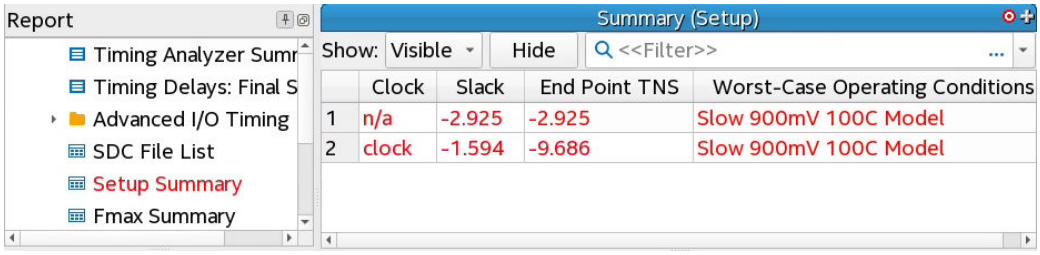
Figure 3. Create Clock Dialog Defines Clock Constraints



Specifying realistic and comprehensive timing requirements up front helps the Compiler to achieve the best results for the following reasons:

- Comprehensive timing assignments enable the Compiler to work hardest to optimize the performance of the timing-critical parts of the design. This optimization can also save area or power utilization in non-critical parts of the design.
- Enables physical synthesis optimizations based on the comprehensive timing requirements.

Figure 4. Timing Analyzer Shows Failing Paths in Red



Following compilation and timing analysis, the Compilation Report reports whether the design meets the timing requirements. You can then use the Quartus Prime Timing Analyzer to fine tune constraints and report detailed information about all timing paths.

Related Information

- Using the Quartus Prime Timing Analyzer
  In *Quartus Prime Pro Edition User Guide: Timing Analyzer*

- Quartus Prime Timing Analyzer Cookbook

1.3. Optimization Trade-Offs and Limitations

Design optimization requires balancing the trade-offs between device performance, resource usage, power utilization, and compilation time. Your application of project settings and constraints determines the balance of these factors in meeting your design goals. When you want to increase optimization of one type, you can consider the trade-offs that might limit that optimization.

Table 1. Design Optimization Trade-Off Examples

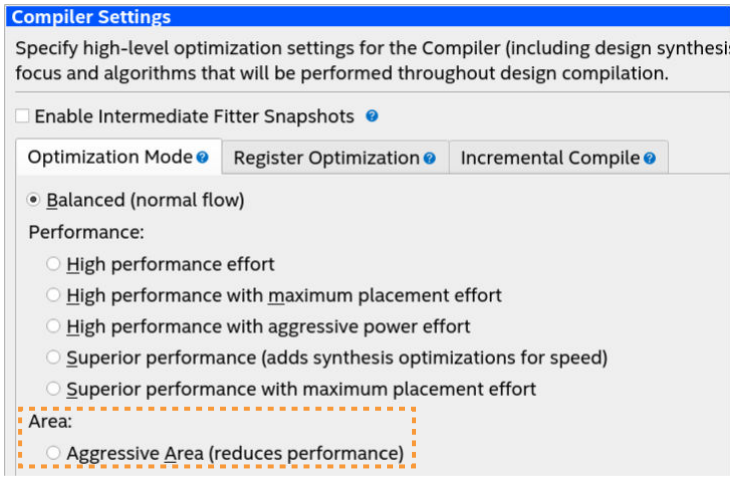
Trade-off	Comments
Resource usage and critical path timing.	Certain techniques (such as logic duplication) can improve timing performance at the cost of increased area.
Power requirements can result in area and timing trade-offs.	For example, reducing the number of available high-speed tiles, or attempting to shorten high-power nets at the expense of critical path nets.
System cost and time-to-market considerations can affect the choice of device.	For example, a device with a higher speed grade or more clock networks can facilitate timing closure at the expense of higher power consumption and system cost.

Finally, constraints that are too stringent can produce a situation with no possible solution for the selected device. If the Fitter cannot resolve a design due to resource limitations, timing constraints, or power constraints, consider rewriting parts of the HDL code.

1.3.1. Area Reduction Trade-Offs

By default, the Quartus Prime Fitter might physically spread a design over the entire device to meet the set timing constraints. If you prefer to optimize your design to use the smallest area, you can change this behavior by selecting **Aggressive Area** for the Compiler **Optimization Mode**. If you require reduced area, you can enable certain physical synthesis options to modify your netlist to create a more area-efficient implementation, but at the cost of increased runtime and decreased performance.

Figure 5. Optimize for Area



Related Information

- Area Optimization on page 47
- Netlist Optimizations and Physical Synthesis on page 40

1.3.2. Critical Path Delay Reduction Trade-Offs

To meet complex timing requirements involving multiple clocks, routing resources, and area constraints, the Quartus Prime software offers a close interaction between synthesis, floorplan editing, place-and-route, and timing analysis processes.

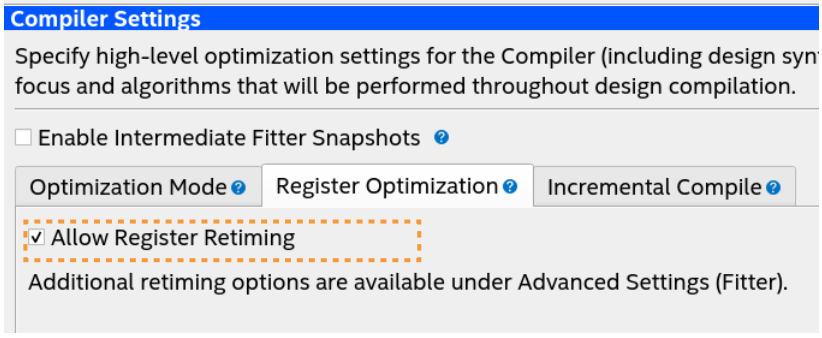
By default, the Quartus Prime Fitter works to meet the timing requirements, and reduces fitting effort once the requirements are met. Therefore, specifying realistic constraints is crucial for achieving timing closure.

Under-constraining your design can lead to sub-optimal results. Over-constraining your design might cause the Fitter to over-optimize non-critical paths at the expense of true critical paths. Over-constraining the design may also increase area and compilation time.

When designs have very high resource usage, the Fitter may struggle to find a legal placement. In such circumstances, the Fitter automatically modifies settings to try to trade off performance for area.

In high-density FPGAs, routing accounts for a major part of critical path timing. Because of this, duplicating or retiming logic can allow the Fitter to reduce delay on critical paths. The Quartus Prime software offers push-button netlist optimizations and physical synthesis options that can improve design performance at the expense of considerable increases of compilation time and area.

Figure 6. Register Optimization



Turn on only those options that help you keep reasonable compilation times and resource usage. Alternately, you can modify the HDL to manually duplicate or adjust the timing logic.

Related Information

Optimize Critical Paths on page 65

1.3.3. Power Consumption Reduction Trade-Offs

The Quartus Prime software has features that help reduce design power consumption. The power optimization options control the power-driven compilation settings for Synthesis and the Fitter. You can adjust these settings

Related Information

Power Optimization
In *Quartus Prime Pro Edition User Guide: Power Analysis and Optimization*

1.3.4. Compilation Time Trade-Offs

Many Fitter settings influence compilation time. Most of the default settings in the Quartus Prime software are set for reduced compilation time. You can modify these settings for your project requirements to trade-off longer compilation time for increased performance.

The Quartus Prime software supports parallel compilation in computers with multiple processors. This technique can reduce compilation times by up to 15%.

Related Information

Quartus Prime Pro Edition User Guide: Design Compilation

1.4. Design Visualization and Optimization Tools

The Quartus Prime software provides various tools to help you visualize and optimize the design settings and constraints for the best design implementation.

1.4.1. Design Visualization Tools

The Quartus Prime software provides tools that display different graphical representations of your design to help you visualize and optimize placement, connectivity, and routing congestion at various stages of the design cycle.

Table 2. Design Visualization Tools

Tool	Description
Snapshot Viewer	The Compiler can preserve the results of each compilation stage as a snapshot for analysis optimization at each stage. The Snapshot Viewer allows you to easily analyze and optimize compilation results for each snapshot. The Snapshot Viewer provides centralized access to functions and tools that allow you to rapidly analyze clocking, congestion, and correct failing paths and high fan-out nets.
RTL Viewer	Provides a schematic representation of the design before synthesis and place-and-route.
Technology Map Viewer	Provides a schematic representation of the design implementation in the selected device architecture after synthesis and place-and-route. Optionally, you can include timing information.

continued...

Tool	Description
Chip Planner	Allows you to make floorplan assignments, such as Logic Lock placement constraints, and visualize critical paths and routing congestion. Click the **Report Routing Utilization** task to display the routing resource congestion.
Interface Planner	Simplifies the planning of accurate constraints for physical implementation. Use Interface Planner to prototype interface implementations, plan clocks, and rapidly define a legal device floorplan.
Design Partition Planner	Displays design entities, I/O banks, connectivity, design hierarchy, and design partition membership. Design Partition Planner can assist you in visualizing a design's structure for creating effective design partitions.

Related Information

- Design Floorplan Analysis in Chip Planner on page 144
- Using Logic Lock Regions in Combination with Design Partitions on page 181
- RTL Viewer Overview on page 17
- Technology Map Viewer Overview on page 18
- Quartus Prime Pro Edition User Guide: Design Compilation
- Quartus Prime Pro Edition User Guide: Design Constraints
  In *Quartus Prime Pro Edition User Guide: Design Constraints*

1.4.2. Design Optimization Tools

The Quartus Prime software provides tools help you identify design RTL and project settings that potentially limit performance.

Table 3. Design Optimization Tools

Tool	Description	To Access
Design Assistant	Automatically reports any violations against a standard set of Intel FPGA-recommended design guidelines, as Correct Design Assistant Rule Violations on page 74 describes.	**Assignments ➤ Settings ➤ Design Assistant Rule Settings**
Fast Forward Timing Closure Recommendations	Fast Forward compilation generates design recommendations to help you to break performance bottlenecks and maximize use of Hyper-Registers to drive the highest performance in Stratix® 10 and Agilex® 7 designs, as Implement Fast Forward Timing Closure Recommendations on page 75 describes.	On the Compilation Dashboard, click **Fast Forward Timing Closure Recommendations**.
Design Space Explorer II	Provides an easy and efficient way to run seed sweeps with different combinations of design settings and constraints to identify the optimal combination for your design, as Optimize Settings with Design Space Explorer II on page 102 describes.	**Tools ➤ Launch Design Space Explorer II**
Assignment Back-Annotation Dialog Box	Back-annotation copies the last compilation's resource assignments to preserve your optimized implementation in subsequent compilations, as Back-Annotating Optimized Assignments on page 100 describes.	**Assignments ➤ Back-Annotate Assignments**

Related Information

- Quartus Prime Pro Edition User Guide: Design Compilation
- Quartus Prime Pro Edition User Guide: Design Recommendations

1.5. Design Optimization Overview Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2023.04.03	23.1	• Updated product family name to "Intel Agilex 7."
2022.01.07	21.4	• Removed references to obsolete Advisors. • Added Snapshot Viewer description to *Design Visualization Tools* topic. • Removed references to Advisors from *Design Optimization Tools* topic.
2020.09.28	20.3.	• Reorganized "Design Optimization Tools" section. • Added references to Design Assistant, Fast Forward Timing Closure Recommendations, and assignment back-annotation to "Design Optimization Tools" topic. • Added Interface Planner and State Machine Viewer to list of "Design Visualization Tools". • Reworded titles in "Optimization Trade-Offs and Limitations" section for greater clarity.
2018.05.07	18.0	• General topic reorganization. • Added how DSE II works, and the main steps to follow when performing a design exploration.
2017.11.06	17.1	• Added mention to the Design Partition Planner in Design Analysis topic.
2016.10.31	16.1	• Implemented Intel rebranding.
2016.05.03	16.0	Removed statements about serial equivalence when using multiple processors.
2015.11.02	15.1	Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Updated DSE II content.
June 2014	14.0	Updated format.
November 2013	13.1	Minor changes for HardCopy.
May 2013	13.0	Added the information about initial compilation requirements. This section was moved from the Area Optimization chapter of the Quartus Prime Handbook. Minor updates to delineate division of Timing and Area optimization chapters.
June 2012	12.0	Removed survey link.
November 2011	10.0	Template update.
December 2010	10.0	Changed to new document template. No change to content.
August 2010	10.0	Corrected link
July 2010	10.0	Initial release. Chapter based on topics and text in Section III of volume 2.

2. Optimizing the Design Netlist

You can use the Quartus Prime Netlist Viewers to analyze and debug your design netlist.

Related Information

- Quartus Prime Design Flow with the Netlist Viewers on page 16
- RTL Viewer Overview on page 17
- Technology Map Viewer Overview on page 18
- Filtering in the Schematic View on page 30
- Viewing a Timing Path on page 37

2.1. When to Use the Netlist Viewers: Analyzing Design Problems

You can use the Netlist Viewers to analyze and debug your design. The following simple examples show how to use the RTL Viewer and Technology Map Viewer to analyze problems encountered in the design process.

Using the RTL Viewer is a good way to view your initial synthesis results to determine whether you have created the necessary logic, and that the logic and connections have been interpreted correctly by the software. You can use the RTL Viewer to check your design visually before simulation or other verification processes. Catching design errors at this early stage of the design process can save you valuable time.

If you see unexpected behavior during verification, use the RTL Viewer to trace through the netlist and ensure that the connections and logic in your design are as expected. Viewing your design helps you find and analyze the source of design problems. If your design looks correct in the RTL Viewer, you know to focus your analysis on later stages of the design process and investigate potential timing violations or issues in the verification flow itself.

You can use the Technology Map Viewer to look at the results at the end of Analysis and Synthesis. If you have compiled your design through the Fitter stage, you can view your post-mapping netlist in the Technology Map Viewer (Post-Mapping) and your post-fitting netlist in the Technology Map Viewer. If you perform only Analysis and Synthesis, both the Netlist Viewers display the same post-mapping netlist.

In addition, you can use the RTL Viewer or Technology Map Viewer to locate the source of a particular signal, which can help you debug your design. Use the navigation techniques described in this chapter to search easily through your design. You can trace back from a point of interest to find the source of the signal and ensure the connections are as expected.

The Technology Map Viewer can help you locate post-synthesis nodes in your netlist and make assignments when optimizing your design. This functionality is useful when making a multicycle clock timing assignment between two registers in your design.

Start at an I/O port and trace forward or backward through the design and through levels of hierarchy to find nodes of interest, or locate a specific register by visually inspecting the schematic.

Throughout your FPGA design, debug, and optimization stages, you can use all of the netlist viewers in many ways to increase your productivity while analyzing a design.

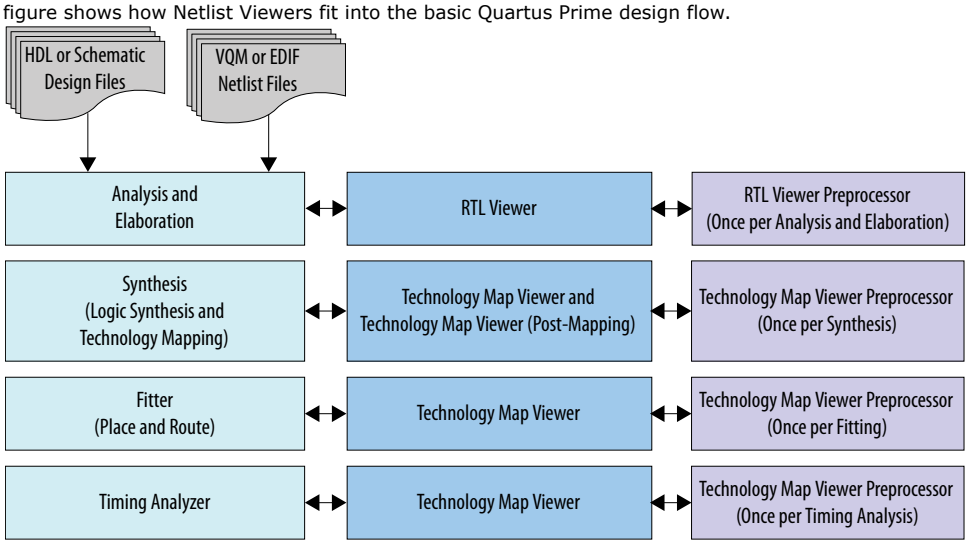
2.2. Quartus Prime Design Flow with the Netlist Viewers

When you first open one of the Netlist Viewers after compiling the design, a preprocessor stage runs automatically before the Netlist Viewer opens.

Click the link in the preprocessor process box to go to the **Settings ➤ Compilation Process Settings** page where you can turn on the **Run Netlist Viewers preprocessing during compilation** option. If you turn this option on, the preprocessing becomes part of the full project compilation flow and the Netlist Viewer opens immediately without displaying the preprocessing dialog box.

Figure 7. Quartus Prime Design Flow Including the RTL Viewer and Technology Map Viewer

This figure shows how Netlist Viewers fit into the basic Quartus Prime design flow.



Before the Netlist Viewer can run the preprocessor stage, you must compile your design:

- To open the RTL Viewer first perform Analysis and Elaboration.
- To open the Technology Map Viewer (Post-Fitting) or the Technology Map Viewer (Post-Mapping), first perform Analysis and Synthesis.

The Netlist Viewers display the results of the last successful compilation.

- Therefore, if you make a design change that causes an error during Analysis and Elaboration, you cannot view the netlist for the new design files, but you can still see the results from the last successfully compiled version of the design files.
- If you receive an error during compilation and you have not yet successfully run the appropriate compilation stage for your project, the Netlist Viewer cannot be displayed; in this case, the Quartus Prime software issues an error message when you try to open the Netlist Viewer.

Note: If the Netlist Viewer is open when you start a new compilation, the Netlist Viewer closes automatically. You must open the Netlist Viewer again to view the new design netlist after compilation completes successfully.

2.3. RTL Viewer Overview

The RTL Viewer allows you to view a register transfer level (RTL) graphical representation of Quartus Prime Pro Edition synthesis results or third-party netlist files in the Quartus Prime software.

You can view results after Analysis and Elaboration for designs that use any supported Quartus Prime design entry method, including Verilog HDL Design Files (`.v`), SystemVerilog Design Files (`.sv`), VHDL Design Files (`.vhd`), AHDL Text Design Files (`.tdf`), or schematic Block Design Files (`.bdf`).

You can also view the hierarchy of atom primitives (such as device logic cells and I/O ports) for designs that generate Verilog Quartus Mapping File (`.vqm`) or Electronic Design Interchange Format (`.edf`) files through a synthesis tool.

The RTL Viewer displays a schematic view of the design netlist after Analysis and Elaboration or after the Quartus Prime software performs netlist extraction, but before technology mapping and synthesis or fitter optimizations. This view a preliminary pre-optimization design structure and closely represents the original source design.

- For designs synthesized with the Quartus Prime Pro Edition synthesis, this view shows how the Quartus Prime software interprets the design files.
- For designs synthesized with a third-party synthesis tool, this view shows the netlist that the synthesis tool generates.

To run the RTL Viewer for an Quartus Prime project, first analyze the design to generate an RTL netlist. To analyze the design and generate an RTL netlist, click **Processing ➤ Start ➤ Start Analysis & Elaboration**. You can also perform a full compilation on any process that includes the initial Analysis and Elaboration stage of the Quartus Prime compilation flow.

To open the RTL Viewer, click **Tools ➤ Netlist Viewers ➤ RTL Viewer**.

2.3.1. Maximizing Readability in RTL Viewer

While displaying a design, the RTL Viewer optimizes the netlist to maximize readability:

- Removes logic with no fan-out (unconnected output) or fan-in (unconnected inputs) from the display.
- Hides default connections such as V_{CC} and GND.
- Groups pins, nets, wires, module ports, and certain logic into buses where appropriate.
- Groups constant bus connections.
- Displays values in hexadecimal format.
- Converts NOT gates into bubble inversion symbols in the schematic.
- Merges chains of equivalent combinational gates into a single gate; for example, a 2-input AND gate feeding a 2-input AND gate is converted to a single 3-input AND gate.

2.3.2. Running the RTL Viewer

To run the RTL Viewer for an Quartus Prime project:

1. Analyze the design to generate an RTL netlist by clicking **Processing ➤ Start ➤ Start Analysis & Elaboration**.

   You can also perform a full compilation on any process that includes the initial Analysis and Elaboration stage of the Quartus Prime compilation flow.
2. Open the RTL Viewer by clicking **Tools ➤ Netlist Viewers ➤ RTL Viewer**.

2.4. Technology Map Viewer Overview

The Quartus Prime Technology Map Viewer provides a technology-specific, graphical representation of FPGA designs after Analysis and Synthesis or after the Fitter maps the design into the target device.

The Technology Map Viewer shows the hierarchy of atom primitives (such as device logic cells and I/O ports) in the design. For supported device families, you can also view internal registers and look-up tables (LUTs) inside logic cells (LCELLs), and registers in I/O atom primitives.

Where possible, the Quartus Prime software maintains the port names of each hierarchy throughout synthesis. However, the software may change or remove port names from the design. For example, the software removes ports that are unconnected or driven by GND or V_{CC} during synthesis. If a port name changes, the software assigns a related user logic name in the design or a generic port name such as `IN1` or `OUT1`.

You can view Quartus Prime technology-mapped results after synthesis, fitting, or timing analysis. To run the Technology Map Viewer for an Quartus Prime project, on the **Processing** menu, point to **Start** and click **Start Analysis & Synthesis** to synthesize and map the design to the target technology. At this stage, the Technology Map Viewer shows the same post-mapping netlist as the Technology Map Viewer (Post-Mapping). You can also perform a full compilation, or any process that includes the synthesis stage in the compilation flow.

For designs that completed the Fitter stage, the Technology Map Viewer shows how the Fitter changed the netlist through physical synthesis optimizations, while the Technology Map Viewer (Post-Mapping) shows the post-mapping netlist. If you have completed the Timing Analysis stage, you can locate timing paths from the Timing Analyzer report in the Technology Map Viewer.

To open the Technology Map Viewer, click **Tools ➤ Netlist Viewers ➤ Technology Map Viewer (Post-Fitting)** or **Technology Map Viewer (Post Mapping)**.

Related Information

- Viewing a Timing Path on page 37
- View Contents of Nodes in the Schematic View on page 30

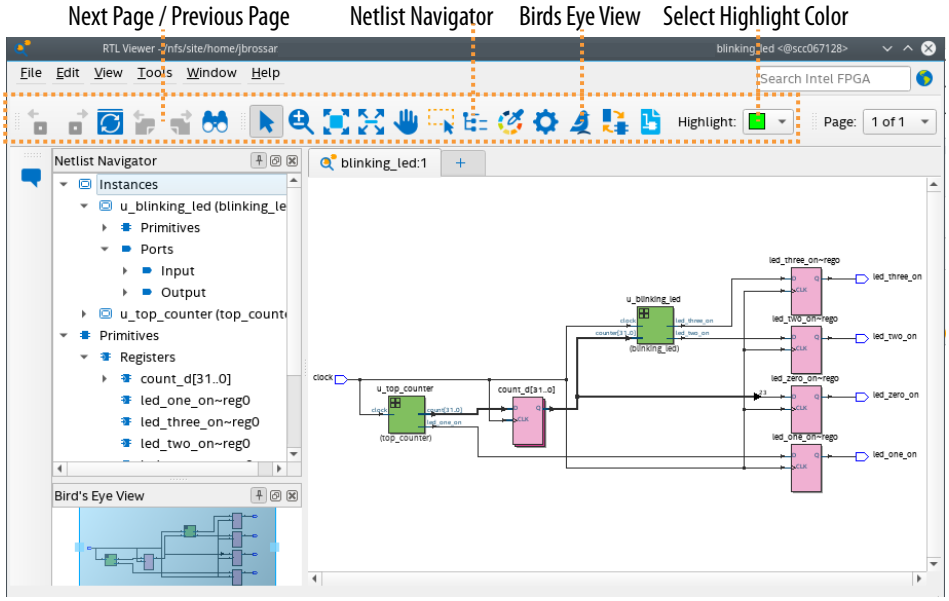
2.5. Netlist Viewer User Interface

The Netlist Viewer is a graphical user-interface for viewing and manipulating nodes and nets in the netlist.

The RTL Viewer and Technology Map Viewer each consist of these main parts:

- The **Netlist Navigator** pane—displays a representation of the project hierarchy.
- The **Find** pane—allows you to find and locate specific design elements in the schematic view.
- The **Properties** pane—displays the properties of the selected block when you select **Properties** from the shortcut menu.
- The schematic view—displays a graphical representation of the internal structure of the design.

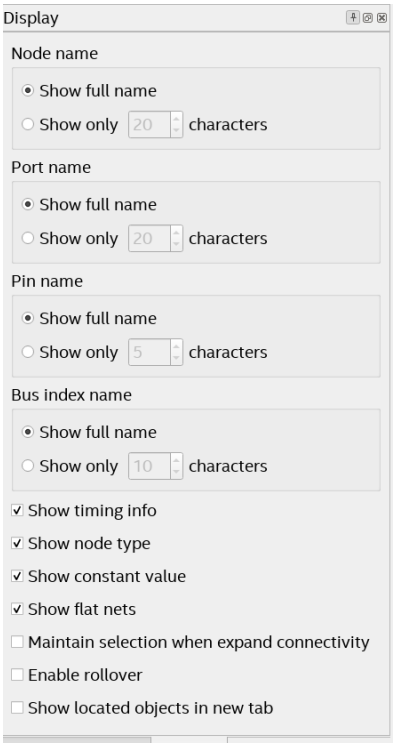
Figure 8. RTL Viewer



Netlist Viewers also contain a toolbar that provides tools to use in the schematic view.

- Use the **Back** and **Forward** buttons to switch between schematic views.
- Click the **Next Page** or **Previous Page** buttons to navigate directly to the next or previous page, respectively. These buttons are helpful when a long schematic partitions to multiple pages.
- The **Refresh** button to restore the schematic view and optimizes the layout. **Refresh** does not reload the database if you change the design and recompile.
- Click the **Find** button opens and closes the **Find** pane.
- Click the **Selection Tool** and **Zoom Tool** buttons to alternate between the selection mode and zoom mode.
- Click the **Fit in Window** button resets the schematic view to encompass the entire design.
- Click the **Fit Selection in Window** button resets the schematic view to encompass the entire selection.
- Use the **Hand Tool** to change the focus of the viewer without changing the perspective.
- Click the **Area Selection Tool** to drag a selection box around ports, pins, and nodes in an area.
- Click the **Netlist Navigator** button to open or close the **Netlist Navigator** pane.
- Click the **Color Settings** button to open the **Colors** pane where you can customize the Netlist Viewer color scheme.

Figure 9. Display Settings



- Click the **Display Settings** button to open the **Display** pane where you can specify the following settings:
  - **Show full name** or **Show only <*n*> characters**. You can specify this separately for **Node name**, **Port name**, **Pin name**, or **Bus index name**.
  - Turn **Show timing info** on or off.
  - Turn **Show node type** on or off.
  - Turn **Show constant value** on or off.
  - Turn **Show flat nets** on or off.
  - Turn **Maintain selection when expand hierarchy** on or off.
  - Turn **Enable rollover** on or off.
  - Turn **Show located objects in new tab** on or off.
- The **Bird's Eye View** button opens the **Bird's Eye View** window which displays a miniature version of the design and allows you to navigate within the design and adjust the magnification in the schematic view quickly.
- The **Show/Hide Instance Pins** button can alternate the display of instance pins not displayed by functions such as cross-probing between a Netlist Viewer and Timing Analyzer. You can also use this button to hide unconnected instance pins when filtering a node results in large numbers of unconnected or unused pins. The Netlist Viewer hides Instance pins by default.
- If the Netlist Viewer display encompasses several pages, the **Show Netlist on One Page** button resizes the netlist view to a single page. This action can make netlist tracing easier.
- Click the **Highlight** list to apply a highlight color to the objects that you select in the schematic. Unhighlight objects with **Unhighlight** or **Unhighlight All** from the right-click menu.

You can have only one RTL Viewer, one Technology Map Viewer (Post-Fitting), and one Technology Map Viewer (Post-Mapping) window open at the same time, although each window can show multiple pages, each with multiple tabs. For example, you cannot have two RTL Viewer windows open at the same time.

Related Information

- Netlist Navigator Pane on page 21
- Netlist Viewers Find Pane on page 23
- Properties Pane on page 22

2.5.1. Netlist Navigator Pane

The **Netlist Navigator** pane displays the entire netlist in a tree format based on the hierarchical levels of the design. Each level groups similar elements into subcategories.

The **Netlist Navigator** pane allows you to traverse through the design hierarchy to view the logic schematic for each level. You can also select an element in the **Netlist Navigator** to highlight in the schematic view.

Note: The **Netlist Navigator** pane does not list nodes inside atom primitives.

For each module in the design hierarchy, the **Netlist Navigator** pane displays the applicable elements listed in the following table. Click the "**+**" icon to expand an element.

Table 4. Netlist Navigator Pane Elements

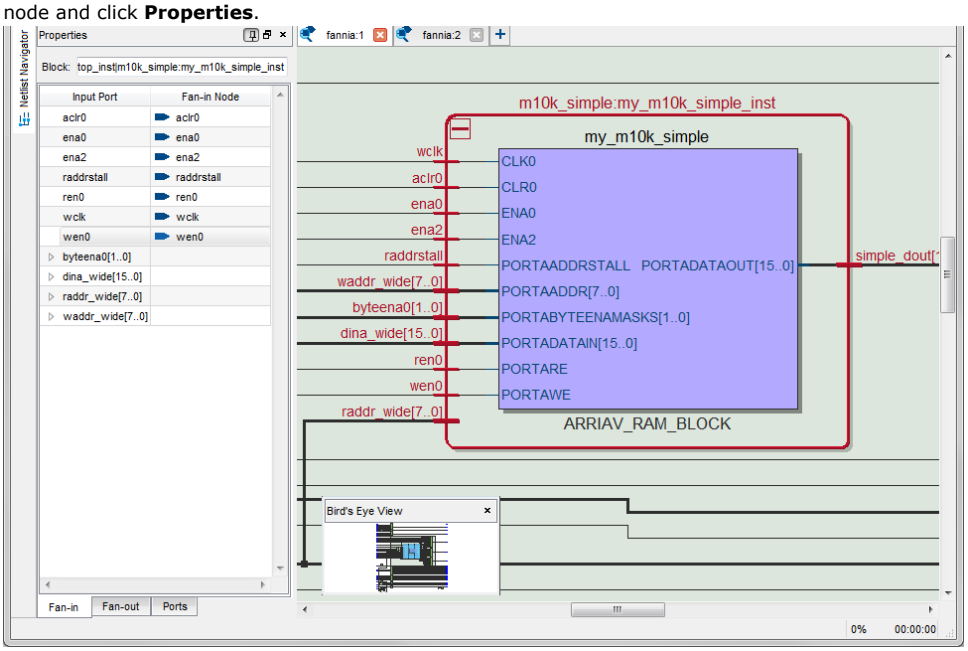
Elements	Description
Instances	Modules or instances in the design that can be expanded to lower hierarchy levels.
Primitives	Low-level nodes that cannot be expanded to any lower hierarchy level. These primitives include: • Registers and gates that you can view in the RTL Viewer when using Quartus Prime Pro Edition synthesis. • Logic cell atoms in the Technology Map Viewer or in the RTL Viewer when using a VQM or EDIF from third-party synthesis software In the Technology Map Viewer, you can view the internal implementation of certain atom primitives, but you cannot traverse into a lower-level of hierarchy.
Ports	The I/O ports in the current level of hierarchy. • Pins are device I/O pins when viewing the top hierarchy level and are I/O ports of the design when viewing the lower-levels. • When a pin represents a bus or an array of pins, expand the pin entry in the list view to see individual pin names.

2.5.2. Properties Pane

You can view the properties of an instance or primitive with the **Properties** pane.

Figure 10. Properties Pane

To view the properties of an instance or primitive in the RTL Viewer or Technology Map Viewer, right-click the node and click **Properties**.



The **Properties** pane contains tabs with the following information about the selected node:

- The **Fan-in** tab displays the **Input port** and **Fan-in Node**.
- The **Fan-out** tab displays the **Output port** and **Fan-out Node**.
- The **Parameters** tab displays the **Parameter Name** and **Values** of an instance.
- The **Ports** tab displays the **Port Name** and **Constant** value (for example, V_{CC} or GND). The following table lists the possible values of a port:

Table 5. Possible Port Values

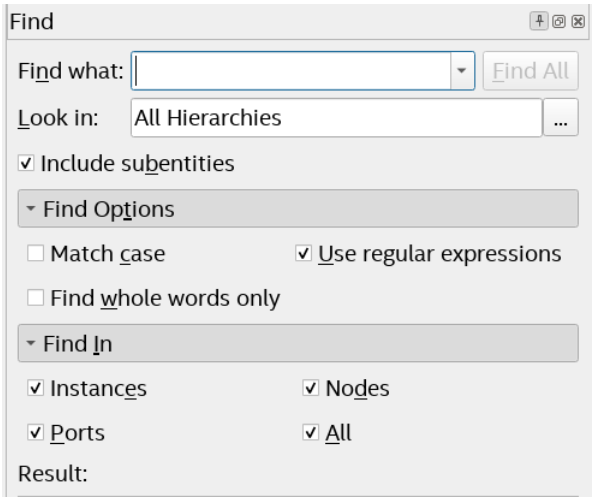
Value	Description
V_{CC}	The port is not connected and has V_{CC} value (tied to V_{CC})
GND	The port is not connected and has GND value (tied to GND)
--	The port is connected and has value (other than V_{CC} or GND)
Unconnected	The port is not connected and has no value (hanging)

If the selected node is an atom primitive, the **Properties** pane displays a schematic of the internal logic.

2.5.3. Netlist Viewers Find Pane

You can narrow the range of the search process by setting the following options in the **Find** pane:

Figure 11. Find Options



- Click **Browse** (…) next to **Look in** to specify the hierarchy level of the search. In the **Select Hierarchy Level** dialog box, you can select a particular instance you want to search.
- Turn on the **Include subentities** option to include child hierarchies of the parent instance during the search.
- Under **Find Options**, turn on or off **Match case**, **Use regular expressions**, **Find whole words only**, or any combination of the three option, to further refine the parameters of the search.
- Under **Find In**, turn on or off **Instances**, **Ports**, **Nodes**, **All** or any combination of options, to further refine the parameters of the search.

When you click the **Find All** button, a progress bar appears below the **Find** box.

All results that match the criteria you set are listed in a table. When you double-click an item in the table, the related node is highlighted in red in the schematic view.

2.6. Schematic View

The schematic view is shown on the right side of the RTL Viewer and Technology Map Viewer. The schematic view contains a schematic representing the design logic in the netlist. This view is the main screen for viewing your gate-level netlist in the RTL Viewer and your technology-mapped netlist in the Technology Map Viewer.

The RTL Viewer and Technology Map Viewer attempt to display schematic in a single page view by default. If the schematic crosses over to several pages, you can highlight a net and use connectors to trace the signal in a single page.

2.6.1. Display Schematics in Multiple Tabbed View

The RTL Viewer and Technology Map Viewer support multiple tabbed views.

With multiple tabbed view, schematics can be displayed in different tabs. Selection is independent between tabbed views, but selection in the tab in focus is synchronous with the Netlist Navigator pane.

To create a new blank tab, click the **New Tab** button at the end of the tab row . You can now drag a node from the **Netlist Navigator** pane into the schematic view.

Right-click in a tab to see a shortcut menu to perform the following actions:

- Create a blank view with **New Tab**
- Create a **Duplicate Tab** of the tab in focus
- Choose to **Cascade Tabs**
- Choose to **Tile Tabs**
- Choose **Close Tab** to close the tab in focus
- Choose **Close Other Tabs** to close all tabs except the tab in focus

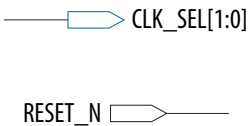
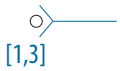
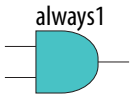
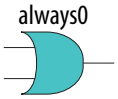
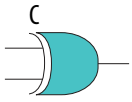
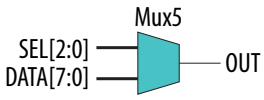
2.6.2. Schematic Symbols

The symbols for nodes in the schematic represent elements of your design netlist. These elements include input and output ports, registers, logic gates, Intel primitives, high-level operators, and hierarchical instances.

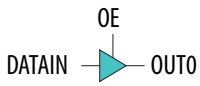
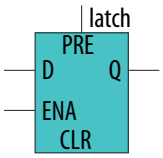
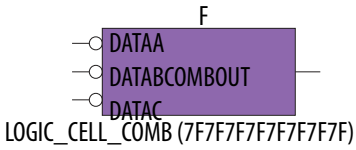
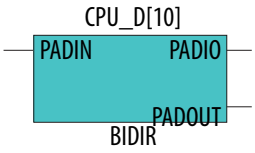
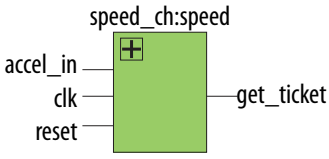
Note: The logic gates and operator primitives appear only in the RTL Viewer. Logic in the Technology Map Viewer is represented by atom primitives, such as registers and LCELLs.

Table 6. Symbols in the Schematic View

This table lists and describes the primitives and basic symbols that you can display in the schematic view of the RTL Viewer and Technology Map Viewer.

Symbol	Description
Wire indicator and net ripper 	Indicates the net signal flow direction into or out of the pin or port. There can be symbols in both directions due to automatic net bundling from the connectivity. You can click the net to highlight the signal flow details in the schematic.
I/O Ports  	An input, output, or bidirectional port in the current level of hierarchy. A device input, output, or bidirectional pin when viewing the top-level hierarchy. The symbol can also represent a bus. Only one wire is shown connected to the bidirectional symbol, representing the input and output paths. Input symbols appear on the left-most side of the schematic. Output and bidirectional symbols appear on the right-most side of the schematic.
I/O Connectors  	An input or output connector, representing a net that comes from another page of the same hierarchy. To go to the page that contains the source or the destination, double-click the connector to jump to the appropriate page.
OR, AND, XOR Gates   	An OR, AND, or XOR gate primitive (the number of ports can vary). A small circle (bubble symbol) on an input or output port indicates the port is inverted.
MULTIPLEXER 	A multiplexer primitive with a selector port that selects between port `0` and port `1`. A multiplexer with more than two inputs is displayed as an operator.

continued...

Symbol	Description
BUFFER 	A buffer primitive. The figure shows the tri-state buffer, with an inverted output enable port. Other buffers without an enable port include LCELL, SOFT, and GLOBAL. The NOT gate and EXP expander buffers use this symbol without an enable port and with an inverted output port.
LATCH 	A latch/DFF (data flipflop) primitive. A DFF has the same ports as a latch and a clock trigger. The other flipflop primitives are similar: • DFFEA (data flipflop with enable and asynchronous load) primitive with additional ALOAD asynchronous load and ADATA data signals • DFFEAS (data flipflop with enable and synchronous and asynchronous load), which has ASDATA as the secondary data port
Atom Primitive 	An atom primitive. The symbol displays the atom name, the port names, and the atom type. The blue shading indicates an atom primitive for which you can view the internal details.
Other Primitive 	Any primitive that does not fall into the previous categories. Primitives are low-level nodes that cannot be expanded to any lower hierarchy. The symbol displays the port names, the primitive or operator type, and its name.
Instance 	An instance in the design that does not correspond to a primitive or operator (a user-defined hierarchy block). The symbol displays the port name and the instance name.
Encrypted Instance	A user-defined encrypted instance in the design. The symbol displays the instance name. You cannot open the schematic for the lower-level hierarchy, because the source design is encrypted.

continued...

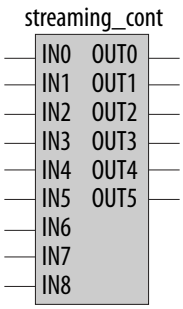
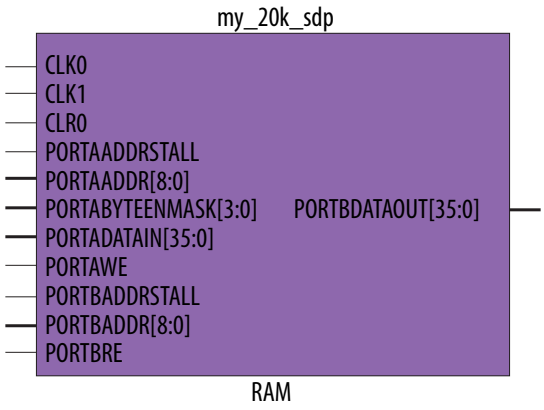
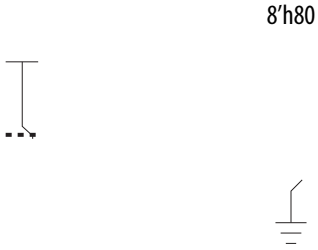
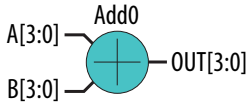
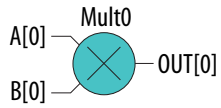
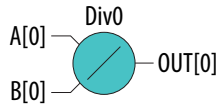
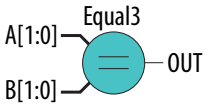
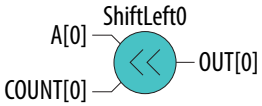
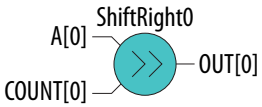
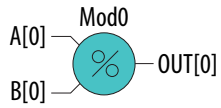
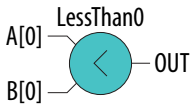
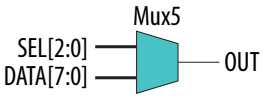
Symbol	Description
	
RAM 	A synchronous memory instance with registered inputs and optionally registered outputs. The symbol shows the device family and the type of memory block. This figure shows a true dual-port memory block in a Stratix M-RAM block.
Constant 	A constant signal value that is highlighted in gray and displayed in hexadecimal format by default throughout the schematic.

Table 7. Operator Symbols in the RTL Viewer Schematic View

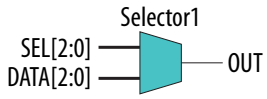
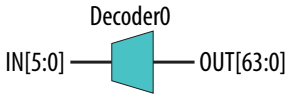
The following lists and describes the additional higher level operator symbols in the RTL Viewer schematic view.

Symbol	Description
	An adder operator: `OUT = A + B`

continued...

Symbol	Description
	A multiplier operator: `OUT = A ¥ B`
	A divider operator: `OUT = A / B`
	Equals
	A left shift operator: `OUT = (A << COUNT)`
	A right shift operator: `OUT = (A >> COUNT)`
	A modulo operator: `OUT = (A%B)`
	A less than comparator: `OUT = (A<:B:A>B)`
	A multiplexer: `OUT = DATA [SEL]` The data range size is $2^{\text{sel range size}}$

continued...

Symbol	Description
	A selector: A multiplexer with one-hot select input and more than two input signals
	A binary number decoder: `OUT = (binary_number (IN) == x)` for x = 0 to $x = 2^{(n+1)} - 1$

Related Information

- Partition the Schematic into Pages on page 35
- Follow Nets Across Schematic Pages on page 35

2.6.3. Select Items in the Schematic View

To select an item in the schematic view, ensure that the **Selection Tool** is enabled in the Netlist Viewer toolbar. Click an item in the schematic view to highlight in red.

Select multiple items by pressing the Shift key while selecting with the mouse.

Items selected in the schematic view are automatically selected in the **Netlist Navigator** pane. The folder then expands automatically if it is required to show the selected entry; however, the folder does not collapse automatically when you deselected the entries.

When you select a hierarchy box, node, or port in the schematic view, the Schematic View highlights the item in red, but not the connecting nets. When you select a net (wire or bus) in the schematic view, the Schematic View highlights all connected nets in red.

Once you select an item, you can perform different actions on it based on the contents of the shortcut menu which appears when you right-click your selection.

Related Information

Netlist Navigator Pane on page 21

2.6.4. Shortcut Menu Commands in the Schematic View

When you right-click a selected instance or primitive in the schematic view, the Netlist Viewer displays a shortcut menu.

If the selected item is a node, you see the following options:

- Click **Expand to Upper Hierarchy** to displays the parent hierarchy of the node in focus.
- Click **Copy ToolTip** to copy the selected item name to the clipboard. This command does not work on nets.
- Click **Hide Selection** to remove the selected item from the schematic view. This command does not delete the item from the design, merely masks it in the current view.
- Click **Filtering** to display a sub-menu with options for filtering your selection.

2.6.5. Filtering in the Schematic View

Filtering allows you to filter out nodes and nets in a netlist to view only the logic elements of interest to you.

You can filter a netlist by selecting hierarchy boxes, nodes, or ports of a node, that are part of the path you want to see. The following filter commands are available:

- **Sources**—displays the sources of the selection.
- **Destinations**—displays the destinations of the selection.
- **Sources & Destinations**—displays the sources and destinations of the selection.
- **Selected Nodes**—displays only the selected nodes.
- **Between Selected Nodes**—displays nodes and connections in the path between the selected nodes.
- **Bus Index**—Displays the sources or destinations for one or more indexes of an output or input bus port.
- **Filtering Options**—Displays the **Filtering Options** dialog box:
  - **Stop filtering at register**—Turning on this option directs the Netlist Viewer to filter out to the nearest register boundary.
  - **Filter across hierarchies**—Turning on this option directs the Netlist Viewer to filter across hierarchies.
  - **Maximum number of hierarchy levels**—Sets the maximum number of hierarchy levels that the schematic view can display.

To filter a netlist, select a hierarchy box, node, port, net, or state node, right-click in the window, point to **Filter** and click the appropriate filter command. The Netlist Viewer generates a new page showing the netlist that remains after filtering.

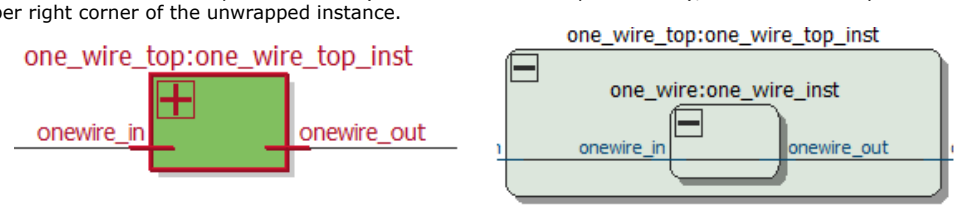
2.6.6. View Contents of Nodes in the Schematic View

In the RTL Viewer and the Technology Map Viewer, you can view the contents of nodes to see their underlying implementation details.

You can view LUTs, registers, and logic gates. You can also view the implementation of RAM and DSP blocks in certain devices in the RTL Viewer or Technology Map Viewer. In the Technology Map Viewer, you can view the contents of primitives to see their underlying implementation details.

Figure 12. Wrapping and Unwrapping Objects

If you can unwrap the contents of an instance, a plus symbol appears in the upper right corner of the object in the schematic view. To wrap the contents (and revert to the compact format), click the minus symbol in the upper right corner of the unwrapped instance.



Note: In the schematic view, the internal details in an atom instance cannot be selected as individual nodes. Any mouse action on any of the internal details is treated as a mouse action on the atom instance.

Figure 13. Nodes with Connections Outside the Hierarchy

In some cases, the selected instance connects to something outside the visible level of the hierarchy in the schematic view. In this case, the net appears as a dotted line. Double-click the dotted line to expand the view to display the destination of the connection .

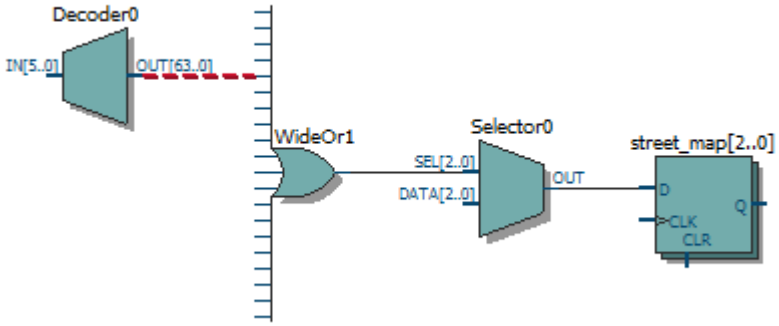


Figure 14. Display Nets Across Hierarchies

In cases where the net connects to an instance outside the hierarchy, you can select the net, and unwrap the node to see the destination ports.

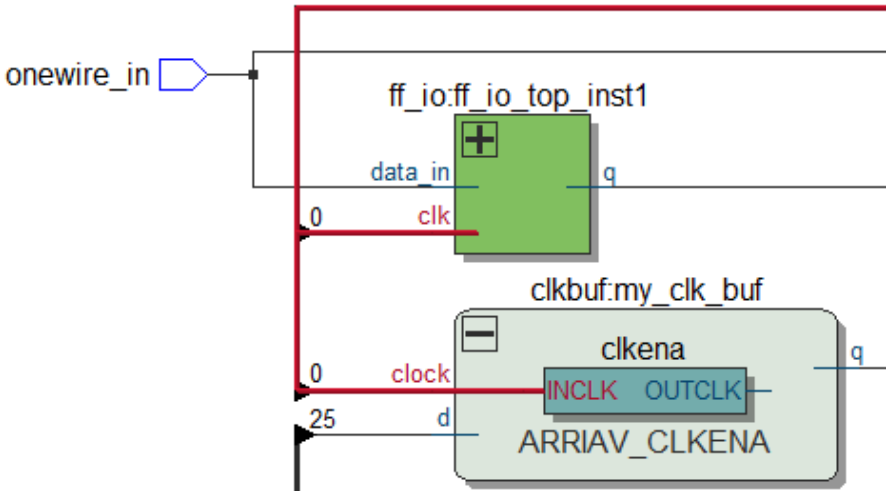
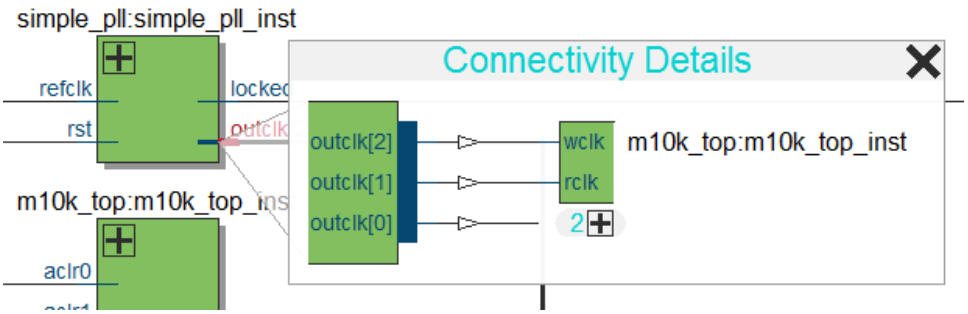


Figure 15. Show Connectivity Details

You can select a port, pin, bus port or bus pin and click **Connectivity Details** in the context menu for that object.



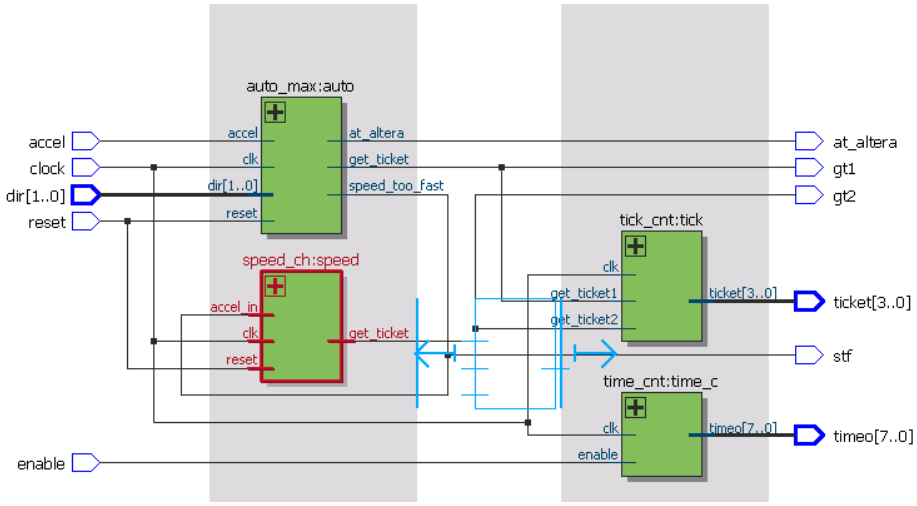
You can double-click objects in the **Connectivity Details** window to navigate to them quickly. If the plus symbol appears, you can further unwrap objects in the view. This can be very useful when tracing a signal in a complex netlist.

2.6.7. Moving Nodes in the Schematic View

Rearrange items in the schematic view by dragging to destination.

To move a node from one area of the netlist to another, select the node and hold down the Shift key. Legal placements appear as shaded areas within the hierarchy. Click to drop the selected node.

Figure 16. Legal Placement when Moving Nodes



To restore the schematic view to its default arrangement, right-click and click **Refresh**.

2.6.8. View LUT Representations in the Technology Map Viewer

You can view different representations of a LUT by right-clicking the selected LUT and clicking **Properties**.

You can view the LUT representations in the following three tabs in the **Properties** dialog box:

- The **Schematic** tab—the equivalent gate representations of the LUT.
- The **Truth Table** tab—the truth table representations.

Note: LUT representations in the Tech Map Viewer are only available for Arria® 10 devices and Cyclone® 10 GX devices in the Quartus Prime Pro Edition software.

Related Information

Properties Pane on page 22

2.6.9. Zoom Controls

Use the Zoom Tool in the toolbar, or mouse gestures, to control the magnification of your schematic on the View menu.

By default, the Netlist Viewer displays most pages sized to fit in the window. If the schematic page is very large, the schematic is displayed at the minimum zoom level, and the view is centered on the first node. Click **Zoom In** to view the image at a larger size, and click **Zoom Out** to view the image (when the entire image is not displayed) at a smaller size. The **Zoom** command allows you to specify a magnification percentage (100% is considered the normal size for the schematic symbols).

You can use the Zoom Tool on the Netlist Viewer toolbar to control magnification in the schematic view. When you select the Zoom Tool in the toolbar, clicking in the schematic zooms in and centers the view on the location you clicked. Right-click in the schematic to zoom out and center the view on the location you clicked. When you select the Zoom Tool, you can also zoom into a certain portion of the schematic by selecting a rectangular box area with your mouse cursor. The schematic is enlarged to show the selected area.

Within the schematic view, you can also use the following mouse gestures to zoom in on a specific section:

- **zoom in**—Dragging a box around an area starting in the upper-left and dragging to the lower right zooms in on that area.
- **zoom -0.5**—Dragging a line from lower-left to upper-right zooms out 0.5 levels of magnification.
- **zoom 0.5**—Dragging a line from lower-right to upper-left zooms in 0.5 levels of magnification.
- **zoom fit**—Dragging a line from upper-right to lower-left fits the schematic view in the page.

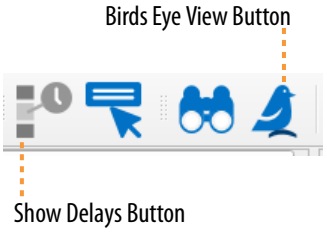
Related Information

Filtering in the Schematic View on page 30

2.6.10. Navigating with the Bird's Eye View

To open the Bird's Eye View, on the View menu, click **Bird's Eye View**, or click the **Bird's Eye View** icon in the toolbar.

Figure 17. Birds Eye View Button on Chip Planner Toolbar



Viewing the entire schematic can be useful when debugging and tracing through a large netlist. The Quartus Prime software allows you to quickly navigate to a specific section of the schematic using the Bird's Eye View feature, which is available in the RTL Viewer and Technology Map Viewer.

The Bird's Eye View shows the current area of interest:

- Select an area by clicking and dragging the indicator or right-clicking to form a rectangular box around an area.
- Click and drag the rectangular box to move around the schematic.
- Resize the rectangular box to zoom-in or zoom-out in the schematic view.

2.6.11. Partition the Schematic into Pages

For large design hierarchies, the RTL Viewer and Technology Map Viewer partition your netlist into multiple pages in the schematic view.

When a hierarchy level is partitioned into multiple pages, the title bar for the schematic window indicates which page is displayed and how many total pages exist for this level of hierarchy. The schematic view displays this as **Page** <*current page number*> **of** <*total number of pages*>.

Related Information

Netlist Viewer User Interface on page 19

2.6.12. Follow Nets Across Schematic Pages

Input and output connector symbols indicate nodes that connect across pages of the same hierarchy. Double-click a connector to trace the net to the next page of the hierarchy.

Note: After you double-click to follow a connector port, the Netlist Viewer opens a new page, which centers the view on the particular source or destination net using the same zoom factor as the previous page. To trace a specific net to the new page of the hierarchy, Intel recommends that you first select the necessary net, which highlights it in red, before you double-click to navigate across pages.

Related Information

Schematic Symbols on page 24

2.7. Cross-Probing to a Source Design File and Other Quartus Prime Windows

The RTL Viewer and Technology Map Viewer allow you to cross-probe to the source design file and to various other windows in the Quartus Prime software.

You can select one or more hierarchy boxes, nodes, state nodes, or state transition arcs that interest you in the Netlist Viewer and locate the corresponding items in another applicable Quartus Prime software window. You can then view and make changes or assignments in the appropriate editor or floorplan.

To locate an item from the Netlist Viewer in another window, right-click the items of interest in the schematic or state diagram, point to **Locate**, and click the appropriate command. The following commands are available:

- **Locate in Assignment Editor**
- **Locate in Pin Planner**
- **Locate in Chip Planner**
- **Locate in Resource Property Editor**
- **Locate in Technology Map Viewer**
- **Locate in RTL Viewer**
- **Locate in Design File**

The options available for locating an item depend on the type of node and whether it exists after placement and routing. If a command is enabled in the menu, it is available for the selected node. You can use the **Locate in Assignment Editor** command for all nodes, but assignments might be ignored during placement and routing if they are applied to nodes that do not exist after synthesis.

The Netlist Viewer automatically opens another window for the appropriate editor or floorplan and highlights the selected node or net in the newly opened window. You can switch back to the Netlist Viewer by selecting it in the Window menu or by closing, minimizing, or moving the new window.

2.8. Cross-Probing to the Netlist Viewers from Other Quartus Prime Windows

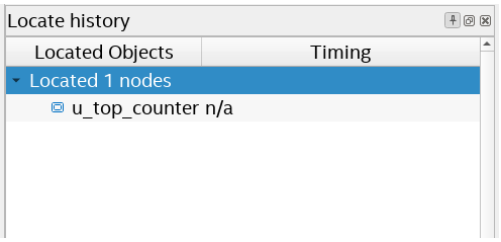
You can cross-probe to the RTL Viewer and Technology Map Viewer from other windows in the Quartus Prime software. You can select one or more nodes or nets in another window and locate them in one of the Netlist Viewers.

You can locate nodes between the RTL Viewer and Technology Map Viewer, and you can locate nodes in the RTL Viewer and Technology Map Viewer from the following Quartus Prime software windows:

- Project Navigator
- Chip Planner
- Resource Property Editor
- Node Finder
- Assignment Editor
- Messages Window
- Compilation Report
- Timing Analyzer (supports the Technology Map Viewer only)

To locate elements in the Netlist Viewer from another Quartus Prime window, select the node or nodes in the appropriate window; for example, select an entity in the **Entity** list on the **Hierarchy** tab in the Project Navigator, or select nodes in the Timing Closure Floorplan, or select node names in the **From** or **To** column in the Assignment Editor. Next, right-click the selected object, point to **Locate**, and click **Locate in RTL Viewer** or **Locate in Technology Map Viewer**. After you click this command, the Netlist Viewer opens, or is brought to the foreground if the Netlist Viewer is open. In addition, the **Locate history** pane displays a list of the items that you locate. Use the **Locate history** pane to easily rerun cross-probing directly within the RTL Viewer or Technology Map Viewer. The **Show located objects in new tab** option always displays the found items in a new tab, as Netlist Viewer User Interface on page 19 describes.

Figure 18. Locate History Pane



Note: The first time the window opens after a compilation, the preprocessor stage runs before the Netlist Viewer opens.

The Netlist Viewer shows the selected nodes and, if applicable, the connections between the nodes. The display is similar to what you see if you right-click the object, then click **Filter ➤ Selected Nodes** using **Filter across hierarchy**. If the nodes cannot be found in the Netlist Viewer, a message box displays the message: **Can't find requested location**.

2.9. Viewing a Timing Path

After completing a full design compilation, including the timing analyzer stage, you can see a visual representation of a timing path cross-probe from a timing report. For details about generating the timing report, refer to the *Quartus Prime Pro Edition User Guide: Timing Analyzer*.

When you locate the timing path from the Timing Analyzer to the Technology Map Viewer, the interconnect and cell delay associated with each node appears on top of the schematic symbols. The total slack of the selected timing path appears in the Page Title section of the schematic.

1. To open the report from the **Compilation Report** Table of Contents, click **Timing Analyzer GUI ➤ Report Timing**, and double-click the timing corner.
2. To open the report from the **Timing Analyzer**, open the **Report Timing** folder in the **Report** pane, and double-click the timing corner.
3. In the **Summary of Paths** tab, right-click a row in the table and select **Locate Path ➤ Locate in Technology Map Viewer**. In the Technology Map Viewer, the schematic page displays the nodes along the timing path with a summary of the total delay.

Related Information

Quartus Prime Pro Edition User Guide: Timing Analyzer

2.10. Optimizing the Design Netlist Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout. • Added device support note to *View LUT Representations in the Technology Map Viewer* topic.
2023.08.01	21.3	• Replaced missing graphics in *Navigating with the Bird's Eye View*.
2021.10.04	21.3	• Updated *Netlist Viewer User Interface* topic to add Fix Selection in Window, Unhighlight, and other new GUI controls. • Updated *Netlist Viewers Find Pane* topic for new Find Options and Find In controls. • Updated *Schematic Symbols* topic for wire indicator symbol. • Updated *Show Connectivity Details* figure for port or pin. • Updated *Cross-Probing to the Netlist Viewers from Other Intel Quartus Prime Windows* topic for Locate History panel.
2019.07.01	19.1	Added *Maintaining Selection in the Resource Property Viewer* topic explaining how the iterm-oterm dependency is maintained in the schematic view.
2018.09.24	18.1.0	• Added link to *Viewing a Timing Path*. • Removed reference to unsupported CARRY buffer from "Schematic Symbols" topic.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	Removed Schematic Viewer topic.
2015.11.02	15.1.0	Added Schematic Viewer topic for viewing stage snapshots. Added information for the following new features and feature updates: • Nets visible across hierarchies • Connection Details • Display Settings • Hand Tool • Area Selection Tool • New default behavior for Show/Hide Instance Pins (default is now off)
2014.06.30	14.0.0	Added Show Netlist on One Page and show/Hide Instance Pins commands.
November 2013	13.1.0	Removed HardCopy device information. Reorganized and migrated to new template. Added support for new Netlist viewer.
November 2012	12.1.0	Added sections to support Global Net Routing feature.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Changed to new document template.
July 2010	10.0.0	• Updated screenshots • Updated chapter for the Quartus Prime software version 10.0, including major user interface changes
November 2009	9.1.0	• Updated devices • Minor text edits

continued...

Document Version	Quartus Prime Version	Changes
March 2009	9.0.0	• Chapter 13 was formerly Chapter 12 in version 8.1.0 • Updated Figure 13–2, Figure 13–3, Figure 13–4, Figure 13–14, and Figure 13–30 • Added "Enable or Disable the Auto Hierarchy List" on page 13–15 • Updated "Find Command" on page 13–44
November 2008	8.1.0	Changed page size to 8.5" × 11"
May 2008	8.0.0	• Added Arria GX support • Updated operator symbols • Updated information about the radial menu feature • Updated zooming feature • Updated information about probing from schematic to Signal Tap Analyzer • Updated constant signal information • Added .png and .gif to the list of supported image file formats • Updated several figures and tables • Added new sections "Enabling and Disabling the Radial Menu", "Changing the Time Interval", "Changing the Constant Signal Value Formatting", "Logic Clouds in the RTL Viewer", "Logic Clouds in the Technology Map Viewer", "Manually Group and Ungroup Logic Clouds", "Customizing the Shortcut Commands" • Renamed several sections • Removed section "Customizing the Radial Menu" • Moved section "Grouping Combinational Logic into Logic Clouds" • Updated document content based on the Quartus Prime software version 8.0

3. Netlist Optimizations and Physical Synthesis

The Quartus Prime software offers netlist optimizations during synthesis, and physical synthesis optimization during fitting, that can improve the performance of your design. Synthesis netlist optimizations operate with the atom netlist of your design, which describes a design in terms of specific primitives. This chapter provides guidelines for applying synthesis and physical synthesis optimization settings.

You can access a range of global synthesis and physical synthesis optimization settings from the **Compiler Settings** page:

Table 8. Synthesis Netlist Optimization and Physical Synthesis Options

Options	Location/Description
Enable synthesis netlist optimization settings	Enable synthesis optimization options (for example, **Synthesis Effort**) in the **Advanced Synthesis Settings** dialog box. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** to access these options.
Enable physical synthesis options	Enable physical synthesis options (for example, **Advanced Physical Synthesis**) in the **Advanced Fitter Settings** dialog box. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** to access these settings.

Note: Because the node names for primitives in the design can change when you use physical synthesis optimizations, you should evaluate whether your design depends on fixed node names. If you use a verification flow that might require fixed node names, such as the Signal Tap Logic Analyzer, formal verification, or the Logic Lock based optimization flow (for legacy devices), disable physical synthesis options.

3.1. Physical Synthesis Optimizations

The Quartus Prime Fitter places and routes the logic cells to ensure critical portions of logic are close together and use the fastest possible routing resources. However, routing delays are often a significant part of the typical critical path delay. Physical synthesis optimizations take into consideration placement information, routing delays, and timing information to determine the optimal placement. The Fitter then focuses timing-driven optimizations at those critical parts of the design. The tight integration of the synthesis and fitting processes is known as physical synthesis.

The following sections describe the physical synthesis optimizations available in the Quartus Prime software, and how they can help improve performance and fitting for the selected device.

Related Information

Compiler Settings Page (Settings Dialog Box)
In *Quartus Prime Help*

3.1.1. Disabling or Enabling Physical Synthesis Optimization

Physical synthesis optimization improves circuit performance by performing combinational and sequential optimization and register duplication.

The Compiler performs physical synthesis optimization by default during place and route. You can disable or enable physical synthesis optimization and related options by following these steps:

To disable or enable physical synthesis optimization:

1. Click **Assignments ➤ Settings ➤ Compiler Settings**.
2. To enable retiming, combinational optimization, and register duplication, click **Advanced Settings (Fitter)**.
3. Enable **Advanced Physical Synthesis**.
4. View physical synthesis results in the **Netlist Optimizations** report under **Compilation Report ➤ Fitter** section.

3.1.2. Physical Synthesis Options

The Quartus Prime software provides physical synthesis optimization options to improve fitting results. To access these options, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

Note: To disable global physical synthesis optimizations for specific elements of your design, assign the **Netlist Optimizations** logic option to **Never Allow** to the specific nodes or entities.

Table 9. Physical Synthesis Options

Option	Description
Advanced Physical Synthesis	Uses the physical synthesis engine to perform combinational and sequential optimization during fitting to improve circuit performance.
Netlist Optimizations	You can use the Assignment Editor to apply the **Netlist Optimizations** logic option. Use this option to disable physical synthesis optimizations for parts of your design. This option is available only for Arria 10 and Cyclone 10 GX devices.
Allow Register Duplication	Allows the Compiler to duplicate registers to improve design performance. When you enable this option, the Compiler copies registers and moves some fan-out to this new node. This optimization improves routability and can reduce the total routing wire in nets with many fan-outs. If you disable this option, this disables optimizations that retime registers. This setting affects Analysis & Synthesis and the Fitter. This option is available only for Arria 10 and Cyclone 10 GX devices.
Allow Register Merging	Allows the Compiler to remove registers that are identical to other registers in the design. When you enable this option, in cases where two registers generate the same logic, the Compiler deletes one register, and the remaining registers fan-out to the deleted register's destinations. This option is useful if you want to prevent the Compiler from removing intentional use of duplicate registers. If you disable register merging, the Compiler disables optimizations that retime registers. This setting affects Analysis & Synthesis and the Fitter.

3.2. Applying Netlist Optimizations

The improvement in performance when using netlist optimizations is design dependent. If you have restructured your design to balance critical path delays, netlist optimizations might yield minimal improvement in performance.

You may have to experiment with available options to see which combination of settings works best for a particular design. Refer to the messages in the compilation report to see the magnitude of improvement with each option, and to help you decide whether you should turn on a given option or specific effort level.

Turning on more netlist optimization options can result in more changes to the node names in the design; bear this in mind if you are using a verification flow, such as the Signal Tap Logic Analyzer or formal verification that requires fixed or known node names.

To find the best results, you can use the Quartus Prime Design Space Explorer II (DSE) to apply various sets of netlist optimization options.

Related Information

Optimize Settings with Design Space Explorer II on page 102

3.2.1. WYSIWYG Primitive Resynthesis

For designs synthesized with a third-party tool, the **Perform WYSIWYG primitive resynthesis** option allows you to apply optimizations to the synthesized netlist.

The **Perform WYSIWYG primitive resynthesis** option directs the Quartus Prime software to un-map the logic elements (LEs) in an atom netlist to logic gates, and then re-map the gates back to Intel-specific primitives. Third-party synthesis tools generate either an `.edf` or `.vqm` atom netlist file using Intel-specific primitives. When you turn on the **Perform WYSIWYG primitive resynthesis** option, the Quartus Prime software uses device-specific techniques during the re-mapping process. This feature re-maps the design using the **Optimization Technique** specified for your project (**Speed**, **Area**, or **Balanced**).

The **Perform WYSIWYG primitive resynthesis** option unmaps and remaps only logic cells, also referred to as LCELL or LE primitives, and regular I/O primitives (which may contain registers). Double data rate (DDR) I/O primitives, memory primitives, digital signal processing (DSP) primitives, and logic cells in carry chains are not remapped. This process does not process logic specified in an encrypted `.vqm` file or an `.edf` file, such as third-party intellectual property (IP).

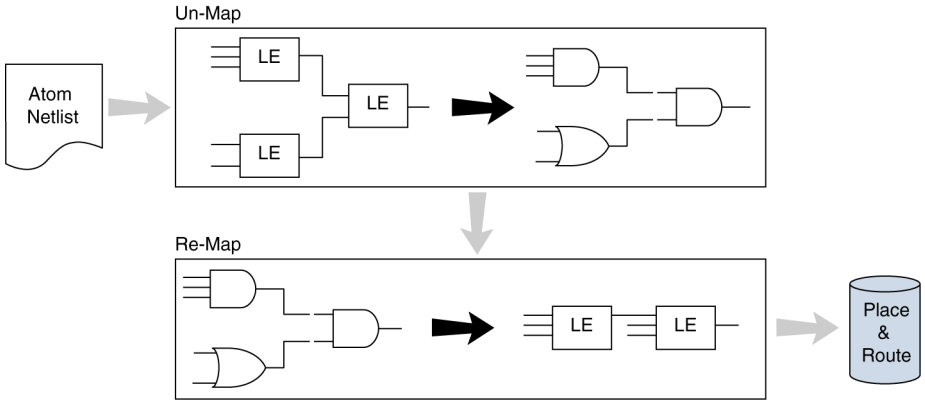
The **Perform WYSIWYG primitive resynthesis** option can change node names in the `.vqm` file or `.edf` file from your third-party synthesis tool, because the primitives in the atom netlist are broken apart and then re-mapped by the Quartus Prime software. The re-mapping process removes duplicate registers. Registers that are not removed retain the same name after re-mapping.

Any nodes or entities that have the **Netlist Optimizations** logic option set to **Never Allow** are not affected during WYSIWYG primitive resynthesis. You can use the Assignment Editor to apply the **Netlist Optimizations** logic option. This option disables WYSIWYG resynthesis for parts of your design.

Note: Primitive node names are specified during synthesis. When netlist optimizations are applied, node names might change because primitives are created and removed. HDL attributes applied to preserve logic in third-party synthesis tools cannot be maintained because those attributes are not written into the atom netlist, which the Quartus Prime software reads.

If you use the Quartus Prime software to synthesize your design, you can use the **Preserve Register (preserve)** and **Keep Combinational Logic (keep)** attributes to maintain certain nodes in the design.

Figure 19. Quartus Prime Flow for WYSIWYG Primitive Resynthesis



3.3. Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime Command-Line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp
```

You can specify many of the options described in this section on either an instance or global level, or both.

Use the following Tcl command to make a global assignment:

```
set_global_assignment -name <QSF variable name> <value>
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <QSF variable name> <value> -to <instance name>
```

Related Information

Quartus Prime Pro Edition User Guide: Scripting

3.3.1. Synthesis Netlist Optimizations

The project `.qsf` file preserves the settings that you specify in the GUI. Alternatively, you can edit the `.qsf` directly. The `.qsf` file supports the following synthesis netlist optimization commands. The **Type** column indicates whether the setting is supported as a global setting, an instance setting, or both.

Table 10. Synthesis Netlist Optimizations and Associated Settings

Setting Name	Quartus Prime Settings File Variable Name	Values	Type
Perform WYSIWYG Primitive Resynthesis	ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP	ON, OFF	Global, Instance
Optimization Mode	OPTIMIZATION_MODE	BALANCED, HIGH PERFORMANCE EFFORT, HIGH PERFORMANCE EFFORT WITH MAXIMUM PLACEMENT EFFORT, HIGH PERFORMANCE WITH AGGRESSIVE POWER EFFORT, SUPERIOR PERFORMANCE, SUPERIOR PERFORMANCE WITH MAXIMUM PLACEMENT EFFORT, AGGRESSIVE AREA, HIGH PLACEMENT ROUTABILITY EFFORT, HIGH PACKING ROUTABILITY EFFORT, OPTIMIZE NETLIST FOR ROUTABILITY, AGGRESSIVE POWER,	Global, Instance
Power-Up Don't Care	ALLOW_POWER_UP_DONT_CARE	ON, OFF	Global

3.3.2. Physical Synthesis Optimizations

The project `.qsf` file preserves the settings that you specify in the GUI. Alternatively, you can edit the `.qsf` directly. The `.qsf` file supports the following synthesis netlist optimization commands. The **Type** column indicates whether the setting is supported as a global setting, an instance setting, or both.

Table 11. Physical Synthesis Optimizations and Associated Settings

Setting Name	Quartus Prime Settings File Variable Name	Values	Type
Advanced Physical Synthesis	ADVANCED_PHYSICAL_SYNTHESIS	ON, OFF	Global

3.4. Netlist Optimizations and Physical Synthesis Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2022.01.07	21.4	• Revised *Disabling or Enabling Physical Synthesis Optimization* topic for default state. • Revised *Physical Synthesis Options* for device support limitations. • Corrected syntax error in *Scripting Support* topic. • Revised *Synthesis Netlist Optimizations and Associated Settings* topic for latest options.
2019.04.24	18.1	Updated example in "Netlist Optimizations and Physical Synthesis" topic.
2019.04.18	18.1	Clarified wording in "Netlist Optimizations and Physical Synthesis" topic.
2018.09.24	18.1	Removed reference to unsupported CASCADE buffer from "Optimize IOC Register Placement for Timing Logic Option" topic.
2018.05.07	18.0	Removed topic: *Isolating a Partition Netlist*.
2017.11.06	17.1	• Removed reference to .vqm files • Added topic: *Isolating a Partition Netlist*.
2016.10.31	16.1	• Implemented Intel rebranding. • Updated physical synthesis options and procedure.
2016.05.02	16.0	• Removed information about deprecated physical synthesis options.
2015.11.02	15.1	• Changed instances of *Quartus II* to *Quartus Prime*. • Added Physical Synthesis.
2014.12.15	14.1	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations Settings to Compiler Settings. • Updated DSE II content.
June 2014	14.0	Updated format.
November 2013	13.1	Removed HardCopy device information.
June 2012	12.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0	Template update.
July 2010	10.0	• Added links to Quartus Prime Help in several sections. • Removed Referenced Documents section. • Reformatted Document Revision History
November 2009	9.1	• Added information to “Physical Synthesis for Registers—Register Retiming” • Added information to “Applying Netlist Optimization Options” • Made minor editorial updates

continued...

Document Version	Quartus Prime Version	Changes
March 2009	9.0	• Was chapter 11 in the 8.1.0 release. • Updated the "Physical Synthesis for Registers—Register Retiming" and "Physical Synthesis Options for Fitting" • Updated "Performing Physical Synthesis Optimizations" • Deleted Gate-Level Register Retiming section. • Updated the referenced documents
November 2008	8.1	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0	• Updated "Physical Synthesis Optimizations for Performance on page 11-9 • Added Physical Synthesis Options for Fitting on page 11-16

4. Area Optimization

This chapter describes techniques for efficient use of device resources.

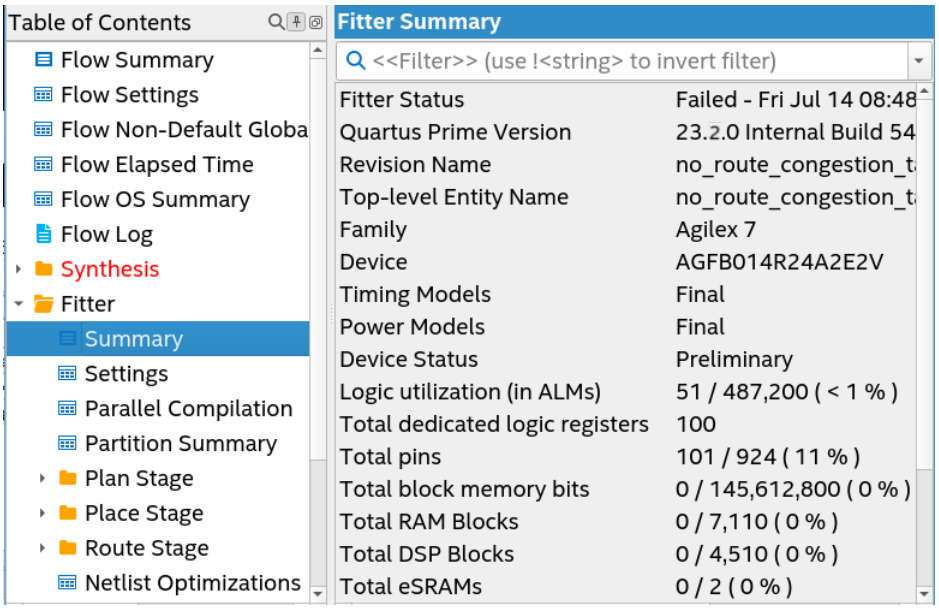
4.1. Resource Utilization Information

Determining device utilization provides useful information regardless of whether the design achieved a successful fit. If the compilation results in a no-fit error, resource utilization information helps to analyze the fitting problems in the design. If the fitting is successful, this information allows you to determine if design changes introduce fitting difficulties. Additionally, you can determine the impact of the resource utilization in the timing performance. The Compilation Report provides information about resource usage.

4.1.1. Flow Summary Report

The **Flow Summary** section of the compilation report indicates whether the design exceeds the available device resources, and reports resource utilization, including pins, memory bits, DSP blocks, and PLLs.

Figure 20. Flow Summary Report



The Fitter can spread logic throughout the device, which may lead to higher overall utilization. As the device fills up, the Fitter automatically searches for logic functions with common inputs to place in one ALM. The number of packed registers also

increases. Therefore, a design that has high overall utilization might still have space for extra logic if the logic and registers can be packed together more tightly. In those cases, you can benefit by a report that provides more details.

4.1.2. Fitter Reports

The Fitter generates detailed reports for each stage of place and route. The **Fitter** section of the Compilation Report includes reports detailing the Fitter's use of device resources.

The **Fitter Resource Usage Summary** report summarizes how the Fitter utilizes logic resources of the target device when implementing your design, such as the number of bits in each type of memory block. This report also summarizes the usage of global clocks, PLLs, DSP blocks, and other device-specific resources.

Related Information

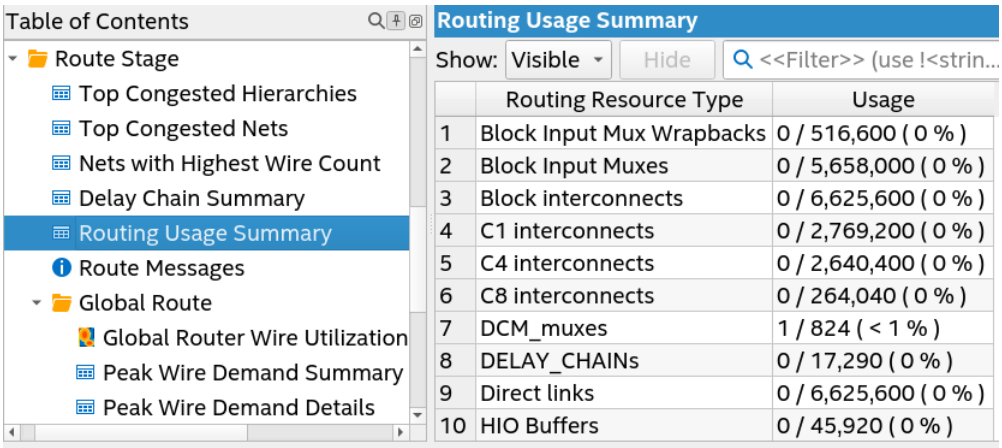
Quartus Prime Pro Edition User Guide: Design Compilation

4.1.2.1. Route Stage Reports

The Route stage reports (**Compilation Report ➤ Fitter ➤ Route Stage**) provide details about the various types of device resources that the Fitter allocates during routing. These details include reports on the following types of routing information:

- The type, number, and overall use percentage of each device resource
- Nets with the highest wire count
- Delay chain summary information
- Global wire utilization information
- The top congested hierarchies and nets

Figure 21. Example Routing Usage Summary

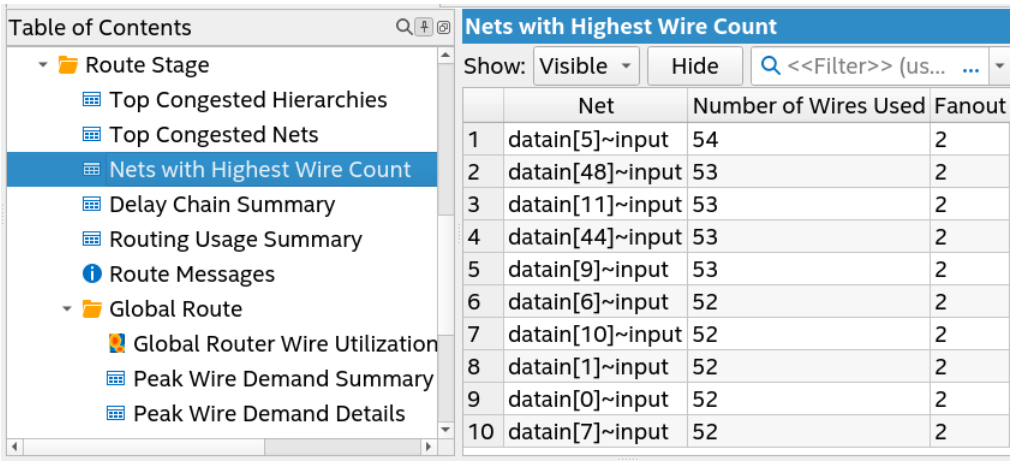


	Routing Resource Type	Usage
1	Block Input Mux Wrapbacks	0 / 516,600 (0 %)
2	Block Input Muxes	0 / 5,658,000 (0 %)
3	Block interconnects	0 / 6,625,600 (0 %)
4	C1 interconnects	0 / 2,769,200 (0 %)
5	C4 interconnects	0 / 2,640,400 (0 %)
6	C8 interconnects	0 / 264,040 (0 %)
7	DCM_muxes	1 / 824 (< 1 %)
8	DELAY_CHAINs	0 / 17,290 (0 %)
9	Direct links	0 / 6,625,600 (0 %)
10	HIO Buffers	0 / 45,920 (0 %)

4.1.2.1.1. Nets with Highest Wire Count Report

The Nets with Highest Wire Count report (**Route Stage ➤ Nets with Highest Wire Count**) lists in descending order the nets that use the highest number of wires in the design and their fan-out count. You can use this report to identify and evaluate the high fan-out nets in the design. The Nets with Highest Wire Count report generates whether or not the design routes successfully.

Figure 22. Example Nets with Highest Wire Count Report

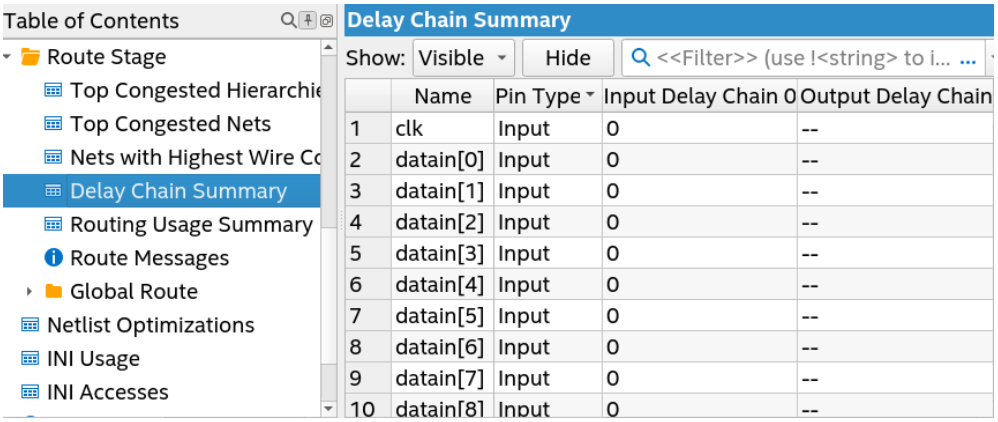


4.1.2.1.2. Delay Chain Summary Report

A delay chain is a series of LCELL or EXP primitives or I/O delay chains in the I/O block that you use to create an intentional delay or asynchronous pulse. A delay chain is generally unreliable because the best-case delay of an LCELL or EXP cannot be guaranteed. This delay chain configuration also increases the sensitivity of the design to operating conditions.

The Delay Chain Summary report (**Route Stage ➤ Delay Chain Summary**) summarizes information about the delay chains in your design. This report lists the node name and pin type in the chain. Delay chains appear in terms of their delay chain fan-out setting and actual delay in ps.

Figure 23. Example Delay Chain Summary Report (Truncated)

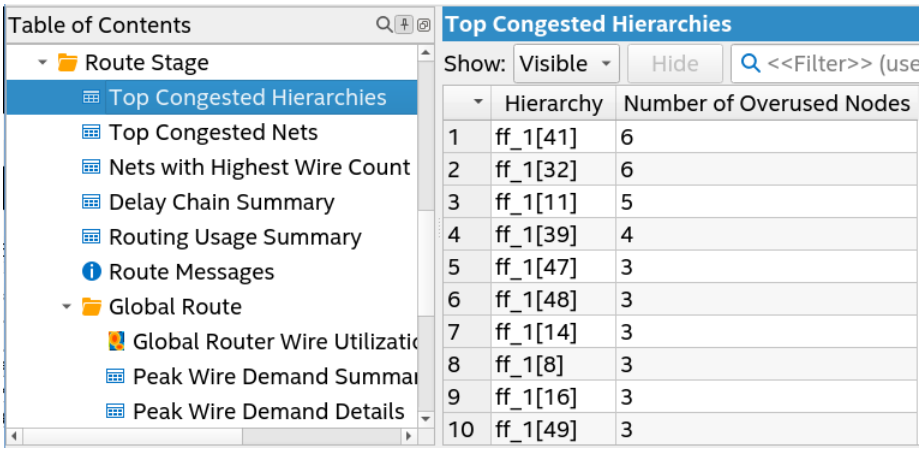


4.1.2.1.3. Top Congested Hierarchies and Nets Reports

If your design fails to route, you can use the Top Congested Hierarchies and Top Congested Nets reports to determine the most congested hierarchies and nets in the design. View these reports under **Compilation Report ➤ Fitter ➤ Route Stage**.

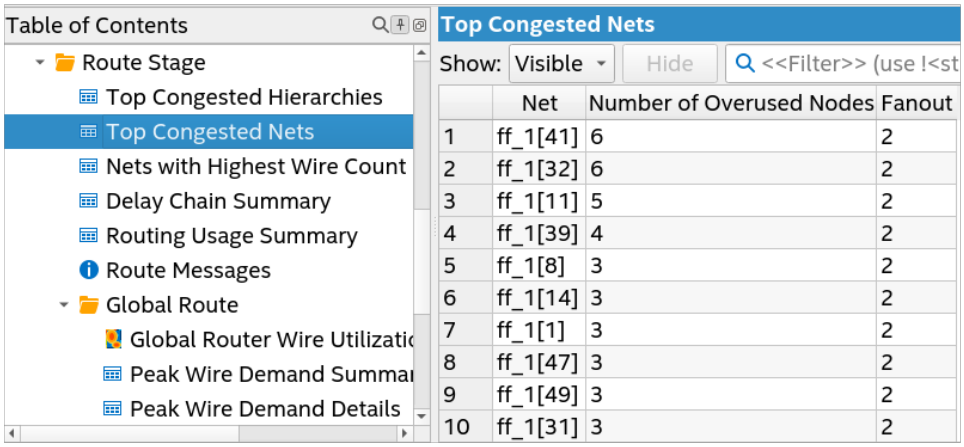
Use context menu commands to locate directly to the reported hierarchies and nets in the Text Editor, Assignment Editor, Pin Planner, Chip Planner, and other editors. Optimize your design in the reported areas to reduce the congestion and successfully route the design.

Figure 24. Example Top Congested Hierarchies Report (Truncated)



	Hierarchy	Number of Overused Nodes
1	ff_1[41]	6
2	ff_1[32]	6
3	ff_1[11]	5
4	ff_1[39]	4
5	ff_1[47]	3
6	ff_1[48]	3
7	ff_1[14]	3
8	ff_1[8]	3
9	ff_1[16]	3
10	ff_1[49]	3

Figure 25. Example Top Congested Nets Report (Truncated)

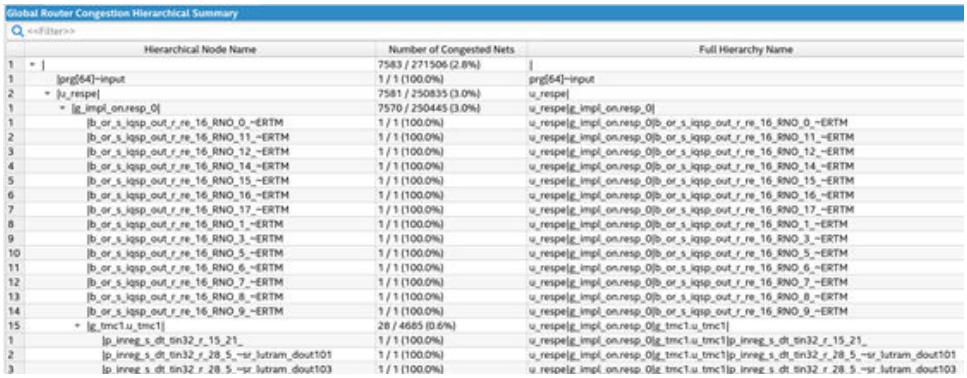


	Net	Number of Overused Nodes	Fanout
1	ff_1[41]	6	2
2	ff_1[32]	6	2
3	ff_1[11]	5	2
4	ff_1[39]	4	2
5	ff_1[8]	3	2
6	ff_1[14]	3	2
7	ff_1[1]	3	2
8	ff_1[47]	3	2
9	ff_1[49]	3	2
10	ff_1[31]	3	2

4.1.2.1.4. Global Route Reports

A global routing congested region is an area of the FPGA where short wire usage in a particular direction exceeds the capacity of that region. A congested net is a net that passes through a global routing congested region. The Global Route reports (**Compilation Report ➤ Fitter ➤ Route Stage ➤ Global Route**) show information about the congested nets in your design.

Figure 26. Example of the Global Router Congestion Hotspot Summary Report



Global Router Congestion Hierarchical Summary

	Hierarchical Node Name	Number of Congested Nets	Full Hierarchy Name
1	|	7583 / 271506 (2.8%)	|
1	|prg[64]~input	1 / 1 (100.0%)	prg[64]~input
2	|u_respe|	7581 / 250835 (3.0%)	u_respe|
1	|g_impl_on.resp_0|	7570 / 250445 (3.0%)	u_respe|g_impl_on.resp_0|
1	|b_or_s_iqsp_out_r_re_16_RNO_0_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_0_~ERTM
2	|b_or_s_iqsp_out_r_re_16_RNO_11_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_11_~ERTM
3	|b_or_s_iqsp_out_r_re_16_RNO_12_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_12_~ERTM
4	|b_or_s_iqsp_out_r_re_16_RNO_14_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_14_~ERTM
5	|b_or_s_iqsp_out_r_re_16_RNO_15_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_15_~ERTM
6	|b_or_s_iqsp_out_r_re_16_RNO_16_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_16_~ERTM
7	|b_or_s_iqsp_out_r_re_16_RNO_17_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_17_~ERTM
8	|b_or_s_iqsp_out_r_re_16_RNO_1_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_1_~ERTM
9	|b_or_s_iqsp_out_r_re_16_RNO_3_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_3_~ERTM
10	|b_or_s_iqsp_out_r_re_16_RNO_5_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_5_~ERTM
11	|b_or_s_iqsp_out_r_re_16_RNO_6_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_6_~ERTM
12	|b_or_s_iqsp_out_r_re_16_RNO_7_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_7_~ERTM
13	|b_or_s_iqsp_out_r_re_16_RNO_8_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_8_~ERTM
14	|b_or_s_iqsp_out_r_re_16_RNO_9_~ERTM	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|b_or_s_iqsp_out_r_re_16_RNO_9_~ERTM
15	|g_tmc1:u_tmc1|	28 / 4685 (0.6%)	u_respe|g_impl_on.resp_0|g_tmc1:u_tmc1|
1	|p_inreg_s_dt_tin32_r_15_21_	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|g_tmc1:u_tmc1|p_inreg_s_dt_tin32_r_15_21_
2	|p_inreg_s_dt_tin32_r_28_5_~sr_lutram_dout101	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|g_tmc1:u_tmc1|p_inreg_s_dt_tin32_r_28_5_~sr_lutram_dout101
3	|p_inreg_s_dt_tin32_r_28_5_~sr_lutram_dout103	1 / 1 (100.0%)	u_respe|g_impl_on.resp_0|g_tmc1:u_tmc1|p_inreg_s_dt_tin32_r_28_5_~sr_lutram_dout103

For example, the Global Router Congestion Hotspot Summary report shows congested net hotspots in your design organized by hierarchical node name. If the congested area is small, the Router may be successful in detouring around overused regions. However, if the congested area is large, routing may be unsuccessful. The exact threshold where the Fitter cannot route the design varies greatly by design characteristics.

Use the Global Router Congestion Hotspot Summary report to identify parts of your RTL code that are associated with routing congestion. The Global Router Wire Utilization Map shows the threshold and size of the largest congested region if the size of adjacent congested regions is below a threshold.

4.1.3. Design Assistant Recommendations

You can run the Design Assistant at various stages throughout the compilation process. Correcting Design Assistant rule violations improves the reliability, timing performance, and logic utilization of the design.

4.1.4. Analysis and Synthesis Reports

For designs synthesized with the Quartus Prime synthesis engine, you can see reports describing optimizations that occurred during compilation.

For example, in the **Analysis & Synthesis** section, **Optimization Results** folder, you can find a list of registers removed during synthesis. With this report you can estimate resource utilization for partial designs so you make sure that registers were not removed due to missing connections with other parts of the design.

Related Information

Quartus Prime Pro Edition User Guide: Design Recommendations

4.1.5. Compilation Messages

If the reports show resource usage lower than 100%, but the design does not fit, either resources are insufficient or the design contains invalid assignments. In either case, the Compiler generates a message in the **Processing** tab of the **Messages** window describing the problem. As resource utilization approaches 100%, the design becomes increasingly difficult to fit.

If the Fitter finishes unsuccessfully and runs much faster than on similar designs, a resource might be over-utilized, there might be an illegal location or timing assignment that is difficult or impossible to resolve.

If the Quartus Prime software takes too long to run when compared to similar designs, the Compiler may not be able to find a valid placement or route solution. In the Compilation Report, look for errors and warnings that indicate these types of problems.

Related Information

Viewing Messages

4.1.6. Chip Planner Visualization

The Chip Planner can help you find areas of the device that have routing congestion for specific types of routing resources. If you find areas with very high congestion, analyze the cause of the congestion. Issues such as high fan-out nets not using global resources, an improperly chosen optimization goal (speed versus area), very restrictive floorplan assignments, or the coding style can cause routing congestion. After you identify the cause, modify the source or settings to reduce routing congestion.

Related Information

Viewing Routing Congestion in Chip Planner on page 151

4.2. Optimizing Resource Utilization

The following lists the stages after design analysis:

1. Optimize resource utilization—Ensure that you have already set the basic constraints
2. I/O timing optimization—Optimize I/O timing after you optimize resource utilization and your design fits in the desired target device
3. Register-to-register timing optimization

Related Information

- Design Optimization Overview on page 6
- Timing Closure and Optimization on page 65

4.2.1. Resource Utilization Issues Overview

Resource utilization issues can be divided into three categories:

- Issues relating to *I/O pin utilization or placement*, including dedicated I/O blocks such as PLLs or LVDS transceivers.
- Issues relating to *logic utilization or placement*, including logic cells containing registers and LUTs as well as dedicated logic, such as memory blocks and DSP blocks.
- Issues relating to *routing*.

4.2.2. I/O Pin Utilization or Placement

Resolve I/O resource problems with these guidelines.

4.2.2.1. Guideline: Modify Pin Assignments or Choose a Larger Package

If a design that has pin assignments fails to fit, compile the design without the pin assignments to determine whether a fit is possible for the design in the specified device and package. You can use this approach if an Quartus Prime error message indicates fitting problems due to pin assignments.

If the design fits when all pin assignments are ignored or when several pin assignments are ignored or moved, you might have to modify the pin assignments for the design or select a larger package.

If the design fails to fit because insufficient I/Os pins are available, a larger device package (which can be the same device density) that has more available user I/O pins can result in a successful fit.

Related Information

Quartus Prime Pro Edition User Guide: Design Constraints

4.2.3. Logic Utilization or Placement

Resolve logic resource problems, including logic cells containing registers and LUTs, as well as dedicated logic such as memory blocks and DSP blocks, with these guidelines.

4.2.3.1. Guideline: Optimize Source Code

If your design does not fit because of logic utilization, then evaluate and modify the design at the source. The Design Assistant reports can help you to identify source code optimizations.

You can often improve logic significantly by making design-specific changes to your source code. This is typically the most effective technique for improving the quality of your results.

If your design does not fit into available logic elements (LEs) or ALMs, but you have unused memory or DSP blocks, check if you have code blocks in your design that describe memory or DSP functions that are not being inferred and placed in dedicated logic. You might be able to modify your source code to allow these functions to be placed into dedicated memory or DSP resources in the target device.

Ensure that your state machines are recognized as state machine logic and optimized appropriately in your synthesis tool. State machines that are recognized are generally optimized better than if the synthesis tool treats them as generic logic. In the Quartus Prime software, you can check for the State Machine report under **Analysis & Synthesis** in the Compilation Report. This report provides details, including the state encoding for each state machine that was recognized during compilation. If your state machine is not being recognized, you might have to change your source code to enable it to be recognized.

Related Information

- AN 584: Timing Closure Methodology for Advanced FPGA Designs
- Quartus Prime Pro Edition User Guide: Design Recommendations

4.2.3.2. Guideline: Optimize Synthesis for Area, Not Speed

If the Fitter cannot resolve a design due to limitations in logic resources, resynthesize the design to improve the area utilization.

First, ensure that the device and timing constraints are set correctly in the synthesis tool. Particularly when area utilization of the design is a concern, ensure that you do not over-constrain the timing requirements for the design. Synthesis tools try to meet the specified requirements, which can result in higher device resource usage if the constraints are too aggressive.

If resource utilization is an important concern, you can optimize for area instead of speed.

- If you are using Quartus Prime synthesis, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** and select **Balanced** or **Area** for the **Optimization Technique**.
- The **Aggressive Area Optimization Mode** optimizes for area at the cost of performance.
- If you want to reduce area for specific modules in the design using the **Area** or **Speed** setting while leaving the default **Optimization Technique** setting at **Balanced**, use the Assignment Editor.
- In some synthesis tools, not specifying an f_{MAX} requirement can result in less resource utilization.

Optimizing for area or speed can affect the register-to-register timing performance.

Note: In the Quartus Prime software, the **Balanced** setting typically produces utilization results that are very similar to those produced by the **Area** setting, with better performance results. The **Area** setting can give better results in some cases.

The Quartus Prime software provides additional attributes and options that can help improve the quality of the synthesis results.

Related Information

Optimization Mode

4.2.3.3. Guideline: Restructure Multiplexers

Multiplexers form a large portion of the logic utilization in many FPGA designs. By optimizing your multiplexed logic, you can achieve a more efficient implementation in your Intel device.

Related Information

Restructure Multiplexers logic option
For more information about the Restructure Multiplexers option

4.2.3.4. Guideline: Perform WYSIWYG Primitive Resynthesis with Balanced or Area Setting

The **Perform WYSIWYG Primitive Resynthesis** logic option specifies whether to perform WYSIWYG primitive resynthesis during synthesis. This option uses the setting specified in the **Optimization Technique** logic option. The **Perform WYSIWYG Primitive Resynthesis** logic option is useful for resynthesizing some or all of the

WYSIWYG primitives in your design for better area or performance. However, WYSIWYG primitive resynthesis can be done only when you use third-party synthesis tools.

Note: The **Balanced** setting typically produces utilization results that are very similar to the **Area** setting with better performance results. The **Area** setting can give better results in some cases. Performing WYSIWYG resynthesis for area in this way typically reduces register-to-register timing performance.

Related Information

Perform WYSIWYG Primitive Resynthesis logic option
For information about this logic option

4.2.3.5. Guideline: Use Register Packing

The **Auto Packed Registers** option implements the functions of two elements into one logic element by combining the register of one element, in which only the register is used with the LUT of another element, in which only the LUT is used.

Remember: DSP register packing is not always possible. For a list of conditions that prevent register packing, refer to the "Fixed Point DSP Register Packing Summary and Fixed Point DSP Register Packing Details" section of *Fitter Feature Specific Report* in the Quartus Prime Pro Edition help.

DSP Register Packing Entity Assignment

In addition, you can control DSP register packing at the entity level by specifying the **DSP Register Packing** entity assignment in the Assignment Editor, or with the corresponding `DSP_REGISTER_PACKING` assignment in the project `.qsf`. **DSP Register Packing** specifies how aggressively the Fitter optimizes DSP performance by automatically packing registers into the internal registers of the specified DSP blocks. With the default **Balanced** setting, the Fitter packs registers into the specified DSP blocks that improve timing. With **Always** enabled, the Fitter aggressively pack registers into the specified DSP blocks, unless your constraints or other legality restrictions prevent packing. With **Disable** enabled, registers do not pack into the specified DSP blocks. The following is the equivalent `.qsf` assignment:

```
set_instance_assignment -name DSP_REGISTER_PACKING -to \
<to> -entity <name> <value>
```

DSP_REGISTER_PACKING_LEVEL Entity Assignment

In addition, you can enter the `DSP_REGISTER_PACKING_LEVEL` entity assignment directly in the project `.qsf` to specify the maximum number of register stages desired for a specific DSP. `DSP_REGISTER_PACKING_LEVEL` specifies the maximum number of registers that you want to pack in the specified DSP instance. The following are `DSP_REGISTER_PACKING_LEVEL` setting values:

- `0`—equivalent to disabling DSP register packing operation of the DSP.
- `1`—the Fitter tries to pack one layer of registers from the DSP's input side.
- `2`—the Fitter tries to add an additional layer of registers from the DSP's output side.
- `3` or `4`—the Fitter tries to add one or two layers of the pipeline registers from the input side.

If the Compiler cannot implement the packing level you specify, the Compiler issues a warning message indicating that the assignment is not respected. the Fixed Point DSP Register Packing Details report also describes the reasons the packing level cannot be met. The following is the equivalent `.qsf` assignment:

```
set_instance_assignment -name DSP_REGISTER_PACKING_LEVEL -to \
<to> -entity <entity name> <value>
```

Fixed Point DSP Register Packing Summary Report and Fixed Point DSP Register Packing Details Report

After running the Compiler's **Plan** stage, the Compilation Report includes the Fixed Point DSP Register Packing Summary report, and the Fixed Point DSP Register Packing Details report. These reports provide information about the use of DSP blocks in your design, including DSP register packing data. The summary report lists how many DSP blocks are fully registered, partially registered, or unregistered.

The Fixed Point DSP Register Packing Details report also indicates the names of the registers packed into register banks, the register usage (fully registered, partially registered, or unregistered), and the reasons preventing any register packing. Viewing these register name details in the report allows you to readily identify registers for packing assignments.

Related Information

- DSP_REGISTER_PACKING Assignment, Quartus Prime Pro Edition Settings File Reference Manual
  For complete command syntax and options
- DSP_REGISTER_PACKING_LEVEL Assignment, Quartus Prime Pro Edition Settings File Reference Manual
  For complete command syntax and options
- Fixed Point DSP Register Packing Summary and Fixed Point DSP Register Packing Details, Quartus Prime Pro Edition Help
  For details about reason preventing register packing

4.2.3.6. Guideline: Remove Fitter Constraints

A design with conflicting constraints or constraints that are difficult to meet may not fit in the targeted device. For example, a design might fail to fit if the location or Logic Lock assignments are too strict and not enough routing resources are available on the device.

To resolve routing congestion caused by restrictive location constraints or Logic Lock region assignments, use the **Routing Congestion** task in the Chip Planner to locate routing problems in the floorplan, then remove any internal location or Logic Lock region assignments in that area. If your design still does not fit, the design is over-constrained. To correct the problem, remove all location and Logic Lock assignments and run successive compilations, incrementally constraining the design before each compilation. You can delete specific location assignments in the Assignment Editor or the Chip Planner. To remove Logic Lock assignments in the Chip Planner, in the Logic Lock Regions Window, or on the Assignments menu, click **Remove Assignments.** Turn on the assignment categories you want to remove from the design in the **Available assignment categories** list.

Related Information

Analyzing and Optimizing the Design Floorplan on page 142

4.2.3.7. Guideline: Flatten the Hierarchy During Synthesis

Synthesis tools typically provide the option of preserving hierarchical boundaries, which can be useful for verification or other purposes. However, the Quartus Prime software optimizes across hierarchical boundaries so as to perform the most logic minimization, which can reduce area in a design with no design partitions.

4.2.3.8. Guideline: Re-target Memory Blocks

If the Fitter cannot resolve a design due to memory resource limitations, the design may require a type of memory that the device does not have.

For memory blocks created with the Parameter Editor, edit the RAM block type to target a new memory block size.

The Compiler can also infer ROM and RAM memory blocks from the HDL code, and the synthesis engine can place large shift registers into memory blocks by inferring the Shift register (RAM-based) IP core. When you turn off this inference in the synthesis tool, the synthesis engine places the memory or shift registers in logic instead of memory blocks. Also, turning off this inference prevents registers from being moved into RAM, improving timing performance,

Depending on the synthesis tool, you can also set the RAM block type for inferred memory blocks. In Quartus Prime synthesis, set the **ramstyle** attribute to the desired memory type for the inferred RAM blocks. Alternatively, set the option to **logic** to implement the memory block in standard logic instead of a memory block.

Review the Resource Utilization by Entity report in the report file to determine whether there is an unusually high register count in any of the modules corresponding with an unexpectedly low RAM block count. Some coding styles prevent the Quartus Prime software from inferring RAM blocks from the source code because of the blocks' architectural implementation, forcing the software to implement the logic in flip-flops.

For example, an asynchronous reset on a register bank might make the register bank incompatible with the RAM blocks in the device architecture, so Compiler implements the register bank in flip-flops. It is often possible to move a large register bank into RAM by slight modification of associated logic.

Using the appropriate memory can also help reduce resource use. For example, a shallow but wider memory may be more suitable for MLABs, rather than for M20K memory blocks.

Related Information

- Quartus Prime Pro Edition User Guide: Design Recommendations
- Agilex 7 Embedded Memory User Guide
- Stratix 10 Embedded Memory User Guide
- Arria 10 Embedded Memory User Guide

4.2.3.9. Guideline: Use Physical Synthesis Options to Reduce Area

The physical synthesis options available at **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)** help you decrease resource usage. When you enable physical synthesis, the Quartus Prime software makes placement-specific changes to the netlist that reduce resource utilization for a specific Intel device.

Note: Physical synthesis increases compilation time. To reduce the impact on compilation time, you can apply physical synthesis options to specific instances.

Related Information

Advanced Fitter Settings Dialog Box

4.2.3.10. Guideline: Retarget or Balance DSP Blocks

A design might not fit because it requires more DSP blocks than the target FPGA device has available.

You can implement all DSP block functions with logic cells, so you can retarget some of the DSP blocks to logic to obtain a fit.

If the DSP function was created with the parameter editor, open the parameter editor and edit the function so it targets logic cells instead of DSP blocks. The Quartus Prime software uses the `DEDICATED_MULTIPLIER_CIRCUITRY` IP core parameter to control the implementation.

DSP blocks also can be inferred from your HDL code for multipliers, multiply-adders, and multiply-accumulators. You can turn off this inference in your synthesis tool. When you are using Quartus Prime synthesis, you can disable inference by turning off the **Auto DSP Block Replacement** logic option for your entire project. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. Turn off **Auto DSP Block Replacement**. Alternatively, you can disable the option for a specific block with the Assignment Editor.

The Quartus Prime software also offers the **DSP Block Balancing** logic option, which implements DSP block elements in logic cells or in different DSP block modes. The default **Auto** setting allows DSP block balancing to convert the DSP block slices automatically as appropriate to minimize the area and maximize the speed of the design. You can use other settings for a specific node or entity, or on a project-wide basis, to control how the Quartus Prime software converts DSP functions into logic cells and DSP blocks. Using any value other than **Auto** or **Off** overrides the `DEDICATED_MULTIPLIER_CIRCUITRY` parameter used in IP core variations.

For designs with large number of low-precision arithmetic operations, such as additions and multiplications, you can enable fractal synthesis optimizations. Fractal synthesis optimizations are useful for high-throughput, arithmetic-intensive designs that exceed all available DSP resources. These optimizations are beneficial in designs with large numbers of low-precision arithmetic operations, such as additions and multiplications.

Related Information

Fractal Synthesis Optimizations, Quartus Prime Pro Edition User Guide: Design Compilation

4.2.3.11. Guideline: Use a Larger Device

If a successful fit cannot be achieved because of a shortage of routing resources, you might require a larger device.

4.2.3.12. Guideline: Reduce Global Signal Congestion

For Stratix 10 and Arria 10 devices, you can refer to the generated Global Signal Visualization report to see global signal routing and clock sector utilization in an interactive heat map.

The presence of too many signals in a clock sector can result in congestion and routing failures in the Fitter's place stage. Use the Global Signal Visualization Report to debug global signal routing congestion and global signal placement and routing failures.

The interactive heat map shows the number of signals in use for a given clock sector. If these global signals are clocks, use clock region assignments to move clocks away from the affected clock sectors.

4.2.3.13. Guideline: Report Pipelining Information

Pipelining the design can be useful for providing resources for retiming and improving performance. However, excessive pipelining can unnecessarily consume area. Use the report_pipeline function to identify areas in the design with excess pipelining.

Related Information

Quartus Prime Pro Edition User Guide: Design Optimization

4.2.4. Routing

Resolve routing resource problems with these guidelines.

4.2.4.1. Guideline: Set Auto Packed Registers to Sparse or Sparse Auto

The **Auto Packed Registers** option reduces LE or ALM count in a design. You can set this option by clicking **Assignment ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

Related Information

Auto Packed Registers logic option

4.2.4.2. Guideline: Set Fitter Aggressive Routability Optimizations to Always

The **Fitter Aggressive Routability Optimization** option is useful if your design does not fit due to excessive routing wire utilization.

If there is a significant imbalance between placement and routing time (during the first fitting attempt), it might be because of high wire utilization. Turning on the **Fitter Aggressive Routability Optimizations** option can reduce your compilation time.

On average, this option can save up to 6% wire utilization, but can also reduce performance by up to 4%, depending on the device.

Related Information

Fitter Aggressive Routability Optimizations logic option

4.2.4.3. Guideline: Increase Router Effort Multiplier

The Router Effort Multiplier controls how quickly the router tries to find a valid solution. The default value is 1.0 and legal values must be greater than 0.

- Numbers higher than 1 help designs that are difficult to route by increasing the routing effort.
- Numbers closer to 0 (for example, 0.1) can reduce router runtime, but usually reduce routing quality slightly.

Experimental evidence shows that a multiplier of 3.0 reduces overall wire usage by approximately 2%. Using a Router Effort Multiplier higher than the default value can benefit designs with complex datapaths with more than five levels of logic. However, congestion in a design is primarily due to placement, and increasing the Router Effort Multiplier does not necessarily reduce congestion.

Note: Any Router Effort Multiplier value greater than 4 only increases by 10% for every additional 1. For example, a value of 10 is actually 4.6.

4.2.4.4. Guideline: Remove Fitter Constraints

A design with conflicting constraints or constraints that are difficult to meet may not fit in the targeted device. For example, a design might fail to fit if the location or Logic Lock assignments are too strict and not enough routing resources are available on the device.

To resolve routing congestion caused by restrictive location constraints or Logic Lock region assignments, use the **Routing Congestion** task in the Chip Planner to locate routing problems in the floorplan, then remove any internal location or Logic Lock region assignments in that area. If your design still does not fit, the design is over-constrained. To correct the problem, remove all location and Logic Lock assignments and run successive compilations, incrementally constraining the design before each compilation. You can delete specific location assignments in the Assignment Editor or the Chip Planner. To remove Logic Lock assignments in the Chip Planner, in the Logic Lock Regions Window, or on the Assignments menu, click **Remove Assignments.** Turn on the assignment categories you want to remove from the design in the **Available assignment categories** list.

Related Information

Analyzing and Optimizing the Design Floorplan on page 142

4.2.4.5. Guideline: Optimize Synthesis for Routability

You can specify Compiler optimization modes that optimize for routability over speed.

If resource utilization is an important concern, you can optimize for routability rather than speed.

The High-Placement Routability Effort, **High Packing Routability Effort**, and **Optimize Netlist for Routability Optimization** modes apply optimizations that improve routability of the design.

The Quartus Prime software provides additional attributes and options that can help improve the quality of your synthesis results.

Related Information

Optimization Mode

4.2.4.6. Guideline: Optimize Source Code

If your design does not fit because of routing problems and the methods described in the preceding sections do not sufficiently improve the routability of the design, you can modify the design at the source to achieve the desired results.

You can often improve results significantly by making design-specific changes to your source code, such as duplicating logic or changing the connections between blocks that require significant routing resources.

You can use the Design Assistant to help you identify areas in the design that can benefit from optimization. For example, the following Design Assistant rules identify registers in the design with high tension span, for which register duplication simplifies place and route.

You can also view the Global Router Wire Utilization Map report to identify instances and nets that utilize many global wire resources.

Related Information

- Design Assistant Rules List
- Design Assistant Design Rule Checking, Quartus Prime Pro Edition User Guide: Design Recommendations
- Global Router Wire Visualization Map, Quartus Prime Pro Edition User Guide: Design Recommendations

4.2.4.7. Guideline: Use a Larger Device

If a successful fit cannot be achieved because of a shortage of routing resources, you might require a larger device.

4.3. Scripting Support

You can run procedures and assign settings described in this chapter in a Tcl script. You can also run procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime command-line and Tcl API Help browser.

1. To run the Help browser, type the following command at the command prompt:

   ```
   quartus_sh --qhelp
   ```

   You can specify many of the options described in this section either in an instance, or at a global level, or both.
2. Use the following Tcl command to make a global assignment:

   ```
   set_global_assignment -name <QSF variable name> <value>
   ```
3. Use the following Tcl command to make an instance assignment:

   ```
   set_instance_assignment -name <QSF variable name> <value> -to <instance name>
   ```

Note: If the <*value*> field includes spaces (for example, 'Standard Fit'), you must enclose the value in straight double quotation marks.

Related Information

- Quartus Prime Pro Edition Settings File Reference Manual
  For information about all settings and constraints in the Quartus Prime software.
- Quartus Prime Pro Edition User Guide: Scripting

4.3.1. Initial Compilation Settings

Use the Quartus Prime Settings File (`.qsf`) variable name in the Tcl assignment to make the setting along with the appropriate value. The **Type** column indicates whether the setting is supported as a global setting, an instance setting, or both.

Table 12. Advanced Compilation Settings

Setting Name	.qsf File Variable Name	Values	Type
Placement Effort Multiplier	PLACEMENT_EFFORT_MULTIPLIER	Any positive, non-zero value	Global
Router Effort Multiplier	ROUTER_EFFORT_MULTIPLIER	Any positive, non-zero value	Global
Router Timing Optimization level	ROUTER_TIMING_OPTIMIZATION_LEVEL	NORMAL, MINIMUM, MAXIMUM	Global
Final Placement Optimization	FINAL_PLACEMENT_OPTIMIZATION	ALWAYS, AUTOMATICALLY, NEVER	Global

4.3.2. Resource Utilization Optimization Techniques

This table lists QSF assignments and applicable values for Resource Utilization Optimization settings:

Table 13. Resource Utilization Optimization Settings

Setting Name	.qsf File Variable Name	Values	Type
Auto Packed Registers	QII_AUTO_PACKED_REGISTERS	AUTO, OFF, NORMAL, MINIMIZE AREA, MINIMIZE AREA WITH CHAINS,SPARSE, SPARSE AUTO	Global, Instance
Perform WYSIWYG Primitive Resynthesis	ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP	ON, OFF	Global, Instance
Optimization Technique	OPTIMIZATION_TECHNIQUE	AREA, SPEED, BALANCED	Global, Instance
Speed Optimization Technique for Clock Domains	SYNTH_CRITICAL_CLOCK	ON, OFF	Instance
State Machine Encoding	STATE_MACHINE_PROCESSING	AUTO, ONE-HOT, GRAY, JOHNSON, MINIMAL BITS, ONE-HOT, SEQUENTIAL, USER-ENCODE	Global, Instance
Auto RAM Replacement	AUTO_RAM_RECOGNITION	ON, OFF	Global, Instance
Auto ROM Replacement	AUTO_ROM_RECOGNITION	ON, OFF	Global, Instance

continued...

Setting Name	.qsf File Variable Name	Values	Type
Auto Shift Register Replacement	AUTO_SHIFT_REGISTER_RECOGNITION	ON, OFF	Global, Instance
Auto Block Replacement	AUTO_DSP_RECOGNITION	ON, OFF	Global, Instance
Number of Processors for Parallel Compilation	NUM_PARALLEL_PROCESSORS	Integer between 1 and 16 inclusive, or ALL	Global

4.4. Area Optimization Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2023.10.02	23.3	• Updated *Guideline: Use Register Packing* to include link to help topic with list of reasons that prevent register packing.
2023.07.24	23.2	• Added new *Route Stage Reports* section to describe latest routing reports to alleviate routing congestion.
2023.04.03	23.1	• Revised *Guideline: Use Register Packing* to describe latest changes for DSP Register Packing and DSP_REGISTER_PACKING_LEVEL entity assignments and reporting.
2022.01.07	21.4	• Corrected syntax error in *Scripting Support* topic. • Added link to *Fitter Reports* topic. • Added new *Design Assistant Recommendations* topic. • Revised *Compilation Messages* topic. • Added *Chip Planner Visualization* topic. • Added reference to Design Assistant to *Guideline: Optimize Source Code* topic. • Revised *Guideline: Optimize Synthesis for Area, Not Speed* topic to mention Aggressive Area Optimization Mode. • Revised *Guideline: Optimize Synthesis for Area, Not Speed* topic to mention Aggressive Area Optimization Mode and remove Speed Optimization Technique for Clock Domains reference. • Revised *Guideline: Retarget Memory Blocks* topic to mention use of appropriate embedded memory IP. • Revised *Guideline: Retarget or Balance DSP Blocks* topic to mention use of fractal synthesis. • Added *Guideline: Report Pipelining Information* topic. • Added reference to Global Router Wire Utilization Map report to *Guideline: Optimize Source Code* topic. • Removed references to obsolete Timing Optimization Advisor.
2018.10.18	18.1	• Corrected broken link to Optimization Modes Help topic.
2018.09.24	18.1	• Divided topic: *Resource Utilization* into topics: *Resource Utilization Information*, *Flow Summary Report*, *Fitter Reports*, *Analysis and Synthesis Reports*, and *Compilation Messages*.
2018.07.03	18.0	Fixed typo and added links in topic *Guideline: Retarget Memory Blocks*.
2017.05.08	17.0	• Removed information about deprecated Integrated Synthesis • Revised topics: *Resolving Resource Utilization Issues*, *Guideline: Optimize Synthesis for Area, Not Speed*
2016.10.31	16.1	• Implemented Intel rebranding.

continued...

Document Version	Quartus Prime Version	Changes
2016.05.02	16.0	• Removed information about deprecated physical synthesis options.
2015.11.02	15.1	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2014.12.15	14.1	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings.
June 2014	14.0	• Removed Cyclone III and Stratix III devices references. • Removed Macrocell-Based CPLDs related information. • Updated template.
May 2013	13.0	Initial release.

5. Timing Closure and Optimization

This chapter describes techniques to improve timing performance when designing for Intel FPGA devices. The application of techniques varies between designs and target FPGA device. Applying each technique does not improve results in all cases.

The default settings and options in the Quartus Prime software provide the most balanced trade-off between compilation time, resource utilization, and timing performance. You can then adjust these settings to determine whether a different mix of settings might provide better results for your design.

5.1. Optimize Multi Corner Timing

Process variations and changes in operating conditions can result in path delays that are significantly smaller than those in the slow corner timing model. As a consequence, the design can present hold time violations on those paths, and in rare cases, additional setup time violations.

In addition, designs targeting newer device families (with smaller process geometry) do not always present the slowest circuit performance at the highest operating temperature. The temperature at which the circuit is slowest depends on the selected device, the design, and the compilation results. The Quartus Prime software manages this new dependency by providing newer device families with three different timing corners—Slow 85°C corner, Slow 0°C corner, and Fast 0°C corner. For other device families, two timing corners are available—Fast 0°C and Slow 85°C corner.

The **Optimize multi-corner timing** option directs the Fitter to meet timing requirements at all process corners and operating conditions. The resulting design implementation is more robust across process, temperature, and voltage variations. This option is on by default, and increases compilation time by approximately 10%.

When this option is off, the Fitter optimizes designs considering only slow-corner delays from the slow-corner timing model (slowest manufactured device for a given speed grade, operating in low-voltage conditions).

5.2. Optimize Critical Paths

Critical paths are timing paths in your design that have a negative slack and may require optimization. These timing paths can span from device I/Os to internal registers, registers to registers, or from registers to device I/Os.

The slack of a path determines its criticality; slack appears in the timing analysis report, which you can generate using the Timing Analyzer.

Design analysis for timing closure is a fundamental requirement for optimal performance in highly complex designs. The analytical capability of the Chip Planner helps you close timing on complex designs.

Related Information

Critical Path Delay Reduction Trade-Offs on page 11

5.2.1. Viewing Critical Paths

Viewing critical paths in the Chip Planner shows why a specific path is failing. You can see if any modification in the placement can reduce the negative slack. To display paths in the floorplan, perform a timing analysis and display results on the Timing Analyzer.

5.3. Optimize Critical Chains

Critical chains are design paths that limit further register retiming optimization. You can use the Hyper-Aware design flow to shorten design cycles and optimize critical chain performance for Stratix 10 and Agilex 7 devices. The Hyper-Aware design flow maximizes use of Hyper-Registers by combining automated register retiming with implementation of targeted timing closure recommendations (Fast Forward compilation). This sum of techniques drive the highest performance for Hyperflex® architecture designs.

A critical chain reports the design paths that limit further register retiming optimization. The Quartus Prime Pro Edition software provides the Hyper-Retimer critical chain reports to help you improve design performance. You can focus on higher level optimization, because the Hyper-Retimer uses Hyper-Registers to evenly balance slacks on all the registers in a critical chain.

Related Information

Hyperflex Architecture High-Performance Design Handbook

5.3.1. Viewing Critical Chains

Looking at the critical chain shows the exact logic that limits retiming operations in your design. For example, you can see if the retiming is limited by your RTL code, or by the constraints you applied on the design. Quartus Prime Pro Edition reports one critical chain per clock domain and clock domain crossing.

The critical chain is available at two different stages in the Hyper Aware Design Flow:

- In the Retiming Limit Details Report—this report for the retiming stage in the Hyper Aware Design Flow, and is enabled by default.
- In the Fast Forward Compilation Report—Click **Fast Forward Timing Closure Recommendations** on the Compilation Dashboard to run..
- You can also graphically visualize the critical chains in the Technology Map Viewer.

Related Information

- Hyperflex Architecture High-Performance Design Handbook
- Stratix 10 HyperFlex Design: Analyzing Critical Chains (OS10CRCHNS) Online Course

5.4. Design Evaluation for Timing Closure

As you move towards completion of your design, you can begin evaluating your design for timing closure. Follow the techniques in this section to evaluate whether your design is likely to close timing. If your design requires further steps for timing closure, you can use the analysis techniques in this section to determine whether RTL changes can help you to close timing.

5.4.1. Review Messages

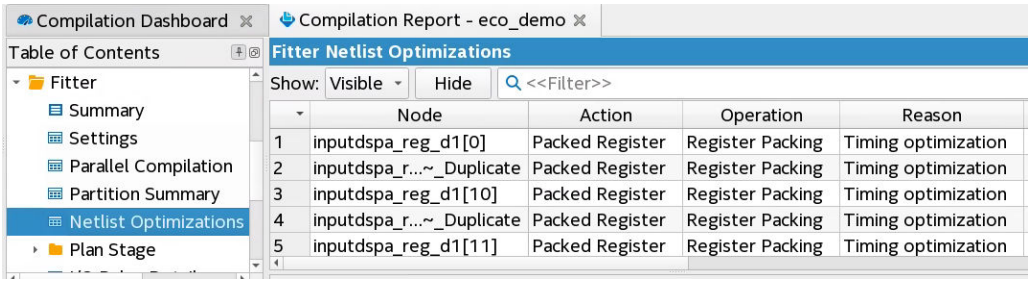
After compiling your design, review the messages in each section of the compilation report. Most designs that fail timing start out with other problems that the Fitter reports as warning messages during compilation. Determine what causes a warning message, and whether to fix or ignore the warning.

After reviewing the warning messages, review the informational messages. Take note of anything unexpected, for example, unconnected ports, ignored constraints, missing files, or other unexpected conditions.

5.4.2. Evaluate Fitter Netlist Optimizations

You can specify options that direct the Fitter to perform optimizations to the design netlist. Specify global options, such as register packing, duplicating or deleting logic cells, or inverting signals by clicking **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

Figure 27. Netlist Optimizations Report



5.4.3. Evaluate Optimization Results

After checking what optimizations were done and how they improved performance, evaluate the runtime it took to get the extra performance. To reduce compilation time, review the physical synthesis and netlist optimizations over a couple of compilations, and edit the RTL to reflect the changes that physical synthesis performed. If a particular set of registers consistently get retimed, edit the RTL to retime the registers the same way. If the changes are made to match what the physical synthesis algorithms did, the physical synthesis options can be turned off to save compile time while getting the same type of performance improvement.

5.4.4. Evaluate Resource Usage

Evaluate the device resources that the design consumes, including global and non-global signal usage, routing utilization, and clustering difficulty. Determine whether you approach capacity for any type of resource that might limit performance.

5.4.4.1. Evaluate Global and Non-Global Usage

For Arria 10 and Cyclone 10 GX designs that contain many clocks, evaluate global and non-global signals to determine whether global resources are used effectively, and if not, consider making changes. After running the Fitter, refer to the Global and Other Fast Signals report to review data on these signals.

Note: Stratix 10 and Cyclone 10 GX devices do not contain regional clocks, but use local routing.

The figure shows an example of inefficient use of a global clock.

Figure 28. Inefficient Use of a Global Clock in Arria 10 Design—Single Fan-Out from Global Clock

Global & Other Fast Signals

Location	Fan-Out	Global Resource Used	Global Line Name
FRACTIONALPLL_X98_Y2_N0	1	Global Clock	--
PLLOUTPUTCOUNTER_X98_Y2_N1	29044	Global Clock	GCLK7
PLLOUTPUTCOUNTER_X98_Y13_N1	253103	Global Clock	GCLK6
FF_X185_Y66_N13	280349	Global Clock	GCLK8
PIN_AE17	4887	Global Clock	GCLK4
FRACTIONALPLL_X98_Y11_N0	1	Global Clock	--
PLLOUTPUTCOUNTER_X98_Y3_N1	1	Global Clock	GCLK5
PLLOUTPUTCOUNTER_X98_Y1_N1	1691	Regional Clock	RCLK29
PLLOUTPUTCOUNTER_X98_Y8_N1	302	Regional Clock	RCLK23
PLLOUTPUTCOUNTER_X98_Y11_N1	141	Regional Clock	RCLK25
PLLOUTPUTCOUNTER_X98_Y10_N1	17	Regional Clock	RCLK22

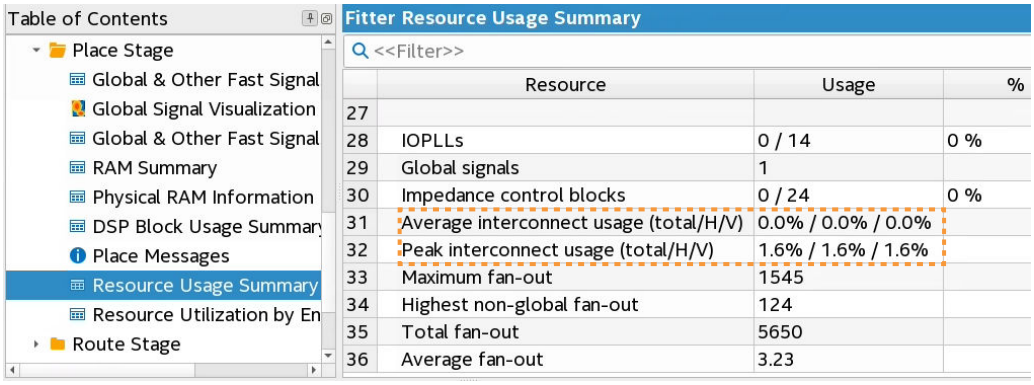
If you assign these resources to a Regional Clock, the Global Clock becomes available for another signal. You can ignore signals with an empty value in the **Global Line Name** column as the signal uses dedicated routing, and not a clock buffer. The Non-Global High Fan-Out Signals report lists the highest fan-out nodes not routed on global signals. `Reset` and `enable` signals appear at the top of the list.

If there is routing congestion in the design, and there are high fan-out non-global nodes in the congested area, consider using global or regional signals to fan-out the nodes, or duplicate the high fan-out registers so that each of the duplicates can have fewer fan-outs. Use the Chip Planner to locate high fan-out nodes, to report routing congestion, and to determine whether the alternatives are viable.

5.4.4.2. Evaluate Routing Usage

Review routing usage reported in the **Fitter Resource Usage Summary** report.

Figure 29. Fitter Resource Usage Summary Report



Average interconnect usage reports the average amount of interconnect that is used, out of what is available on the device. **Peak interconnect usage** reports the largest amount of interconnect used in the most congested areas.

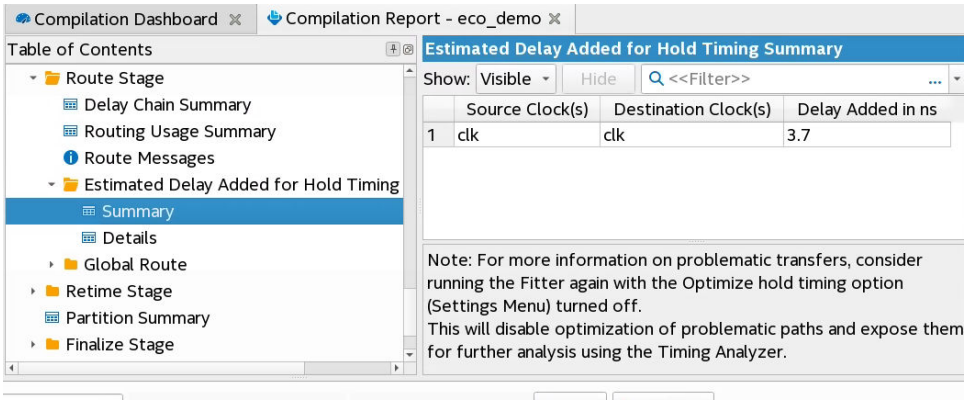
Designs with an average value below 50% typically do not have any problems with routing. Designs with an average between 50-65% may have difficulty routing. Designs with an average over 65% typically have difficulty meeting timing unless the RTL tolerates a highly utilized chip. Peak values at or above 90% are likely to have problems with timing closure; a 100% peak value indicates that all routing in an area of the device has been used, so there is a high possibility of degradation in timing performance.

5.4.4.3. Evaluate Wires Added for Hold

During routing the Fitter may add wire between register paths to increase delay to meet hold time requirements. The Fitter reports how much routing delay was added in the **Estimated Delay Added for Hold Timing** report. Excessive additional wire can indicate an error with the constraint. The cause of such errors is typically incorrect multicycle transfers between multi-rate clocks, and between different clock networks.

Review the specific register paths in the **Estimated Delay Added for Hold Timing** report to determine whether the Fitter adds excessive wire to meet hold timing.

Figure 30. Estimated Delay Added for Hold Timing Report

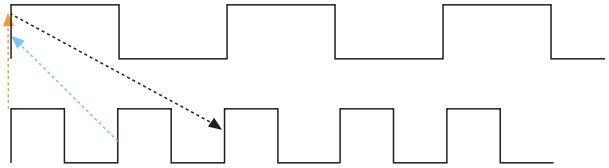


An example of an incorrect constraint which can cause the router to add wire for hold requirements is when there is data transfer from 1x to 2x clocks. Assume the design intent is to allow two cycles per transfer. Data can arrive any time in the two destination clock cycles by adding a multicycle setup constraint as shown in the example:

```
set_multicycle_path -from 1x -to 2x -setup -end 2
```

The timing requirement is relaxed by one 2x clock cycle, as shown in the black line in the waveform in the figure.

Figure 31. Timing Requirement Relaxed Waveform



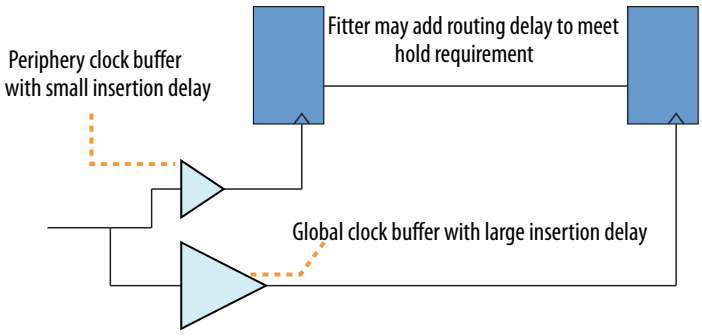
The default hold requirement, shown with the dashed blue line, can force the router to add wire to guarantee that data is delayed by one cycle. To correct the hold requirement, add a multicycle constraint with a hold option.

```
set_multicycle_path -from 1x -to 2x -setup -end 2
set_multicycle_path -from 1x -to 2x -hold -end 1
```

The orange dashed line in the figure above represents the hold relationship, and no extra wire is required to delay the data.

The router can also add wire for hold timing requirements when data transfers in the same clock domain, but between clock branches that use different buffering. Transferring between clock network types happens more often between the periphery and the core. The following figure shows data is coming into a device, a periphery clock drives the source register, and a global clock drives the destination register. A global clock buffer has larger insertion delay than a periphery clock buffer. The clock delay to the destination register is much larger than to the source register, hence extra delay is necessary on the data path to ensure that it meets its hold requirement.

Figure 32. Clock Delay



To identify cases where a path has different clock network types, review the path in the Timing Analyzer, and check nodes along the source and destination clock paths. Also, check the source and destination clock frequencies to see whether they are the

same, or multiples, and whether there are multicycle exceptions on the paths. Finally, ensure that all cross-domain paths that are false by intent have an associated false path exception.

If you suspect that routing is added to fix real hold problems, you can disable the **Optimize hold timing** advanced Fitter setting (**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter) ➤ Optimize hold timing**). Recompile the design with **Optimize hold timing** disabled, and then rerun timing analysis to identify and correct any paths that fail hold time requirements.

Note: Disable the **Optimize hold timing** option only when debugging your design. Ensure to enable the option (default state) during normal compiles. Wire added for hold is a normal part of timing optimization during routing and is not always a problem.

5.4.5. Evaluate Other Reports and Adjust Settings Accordingly

5.4.5.1. Difficulty Packing Design

In the Fitter Resource Section, under the **Resource Usage Summary**, review the **Difficulty Packing Design** report. The **Difficulty Packing Design** report details the effort level (low, medium, or high) of the Fitter to fit the design into the device, partition, and Logic Lock region.

As the effort level of **Difficulty Packing Design** increases, timing closure gets harder. Going from medium to high can result in significant drop in performance or increase in compile time. Consider reducing logic to reduce packing difficulty.

5.4.5.2. Review Ignored Assignments

The Compilation Report includes details of any assignments that the Fitter ignores. The Fitter may ignore assignments if they refer to nodes names that change, but assignments are not updated accordingly. Make sure that the Fitter is not ignoring any valid assignments.

5.4.5.3. Review Non-Default Settings

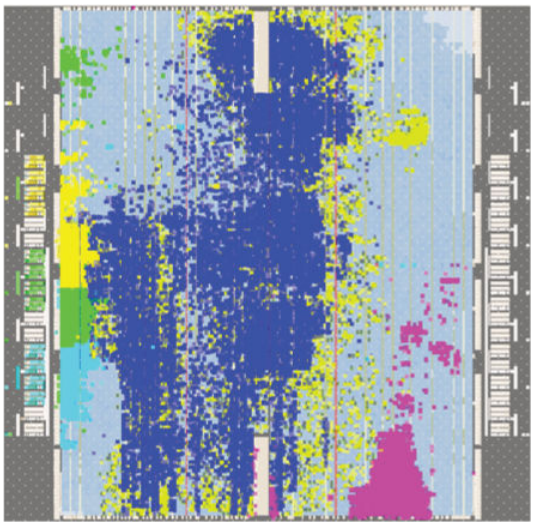
The Synthesis and Fitter reports list all settings set to a non-default value during the compilation. Review the non-default settings to ensure benefit.

5.4.5.4. Review the Design Floorplan

Use the Chip Planner for reviewing placement. You can use the Chip Planner to locate hierarchical entities, using colors for each located entity in the floorplan. Look for logic that seems out of place, based on where you expect it to be

For example, logic that interfaces with I/Os should be close to the I/Os, and logic that interfaces with an IP or memory should be close to the IP or memory.

Figure 33. Floorplan with Color-Coded Entities



The following notes describe use of the visualization in *Floorplan with Color-Coded Entities*:

- The green block is spread apart. Check to see if those paths are failing timing, and if so, what connects to that module that could affect placement.
- The blue and aqua blocks are spread out and mixed together. Check if connections between the two modules contribute to this.
- The pink logic at the bottom must interface with I/Os at the bottom edge. Check fan-in and fan-out of a highlighted module by using the buttons on the task bar. Look for signals that go a long way across the chip and see if they are contributing to timing failures.
- Check global signal usage for signals that affect logic placement, and verify if the Fitter placed logic feeding a global buffer close to the buffer and away from related logic. Use settings like high fan-out on non-global resource to pull logic together.
- Check for routing congestion. The Fitter spreads out logic in highly congested areas, making the design harder to route.

5.4.5.5. Adjust Placement Effort

You can increase the **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter) ➤ Placement Effort Multiplier** value to spend additional compilation time and effort in Place stage of the Fitter.

Adjust the multiplier after reviewing and optimizing other settings and RTL. Try an increased value, up to 4, and reset to default if performance or compile time does not improve.

5.4.5.6. Adjust Fitter Effort

Fitter **Optimization mode** settings allow you to specify whether the Compiler focuses optimization efforts for performance, resource utilization, power, or compile times.

By default, the Fitter **Optimization mode** is set to **Balanced (Normal flow)** mode, which reduces Fitter effort and compilation time as soon as timing requirements are met. You can optionally select another **Optimization mode** to target performance, area, routability, power, or compile time.

To increase Fitter effort further, you can also enable the **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter) ➤ Fitter Effort** option. The default **Auto Fit** setting reduces Fitter effort once timing requirements are met. **Standard Fit (highest effort)** setting uses maximum effort regardless of the design's requirements, leading to higher compilation time and more timing margin.

5.4.5.7. Review Timing Constraints

Ensure that you constrain all clocks with the correct frequency requirements.

To confirm the proper application of timing constraints, run the Design Assistant and then review and correct any timing constraint rule violations. In addition, you can review the **Ignored Constraints** report to locate any constraints assigned to invalid node names in the design. These invalid node names are most commonly caused by changes that you make in the design hierarchy that are not yet reflected in the constraint. Similarly, review the **Report Unconstrained Paths** report to locate unconstrained paths. Add constraints as necessary so that the Compiler can fully optimize the design.

5.4.6. Evaluate Clustering Difficulty

You can evaluate clustering difficulty to help reach timing closure. You can monitor clustering difficulty whenever you add logic and recompile. Use the clustering information to gauge how much timing closure difficulty is inherent in your design.

- If your design is full but clustering difficulty is low or medium, your design itself, rather than clustering, is likely the main cause of congestion.
- Conversely, congestion occurring after adding a small amount of logic to the design, can be due to clustering. If clustering difficulty is high, this contributes to congestion regardless of design size.

5.4.7. Revise and Recompile

Look for obvious problems that you can fix with minimal effort. To identify where the Compiler had trouble meeting timing, perform seed sweeping with about five compiles. Doing so shows consistently failing paths. Consider recoding or redesigning that part of the design.

To reach timing closure, a well written RTL can be more effective than changing your compilation settings. Seed sweeping can also be useful if the timing failure is very small, and the design has already been optimized for performance improvements and is close to final release. Additionally, seed sweeping can be used for evaluating changes to compilation settings. Compilation results vary due to the random nature of fitter algorithms. If a compilation setting change produces lower average performance, undo the change.

Sometimes, settings or constraints can cause more problems than they fix. When significant changes to the RTL or design architecture have been made, compile periodically with default settings and without Logic Lock regions, and re-evaluate paths that fail timing.

5.5. Timing Optimization

You can use the techniques and tools in this section to optimize timing when your design does not meet its timing requirements. Also, refer to the design recommendations in *Optimizing for Timing Closure, Quartus Prime Pro Edition User Guide: Design Recommendations*.

5.5.1. Correct Design Assistant Rule Violations

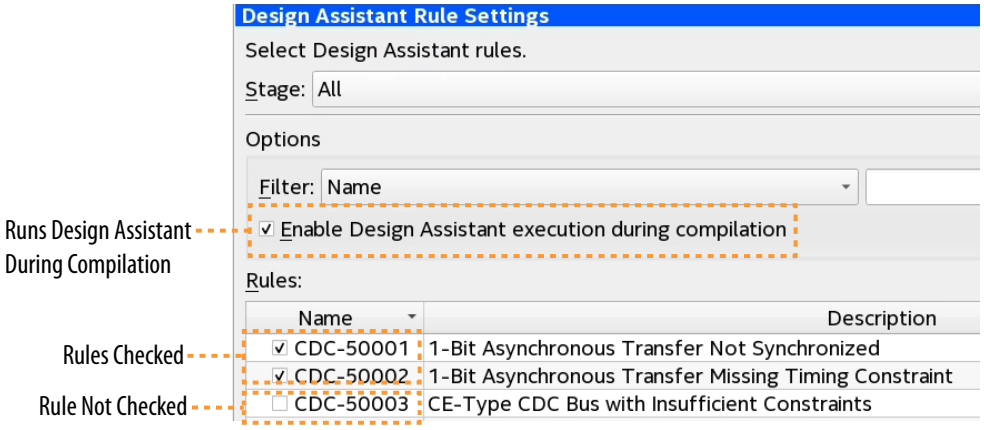
After running any stage of the Compiler, review the Design Assistant reports to analyze any design rule violations and view recommendations to correct failing paths. When enabled, the Quartus Prime Design Assistant automatically runs during compilation and reports any violations against a set of Intel FPGA-recommended design guidelines. Design Assistant rules include Timing Closure, Clocking, CDC, reset, and floorplanning.

You can customize the Design Assistant for your design characteristics and reporting requirements. Run Design Assistant in Compilation Flow mode to view the violations relevant for Compiler stages. Run in analysis mode from tools like the Timing Analyzer and Chip Planner to cross-probe from an individual rule violation to more information.

Follow these steps to enable and run Design Assistant and view results following compilation:

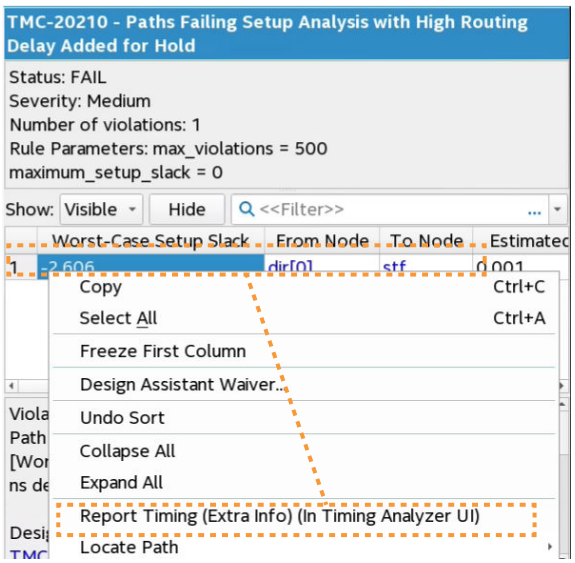
1. Click **Assignments ➤ Settings ➤ Design Assistant Rules Settings**.

Figure 34. Design Assistant Rules Settings



2. To enable Design Assistant checking during compilation, turn on **Enable Design Assistant execution during compilation**.
3. To run Design Assistant during compilation, run one or more modules of the Compiler. Design Assistant reports results for each stage in the Compilation Report.
4. To view the results for each rule, click the rule in the **Rules** list. A description of the rule and design recommendations for correction appear.
5. For timing path-related rule violations, right-click the node or path, and then click **Report Timing (Extra Info)** or **Report Path (Extra Info)**. The Timing Analyzer loads and automatically displays the **Report Timing** or **Report Path** data related to the rule violation, allowing you to probe every aspect of the violation. **Report Path** can report timing even for paths that are cut.

Figure 35. Cross Probing From Design Assistant Rule Violations to Timing Analyzer



Related Information

Quartus Prime Pro Edition User Guide: Design Recommendations

5.5.2. Implement Fast Forward Timing Closure Recommendations

In traditional FPGA timing closure flows, the starting point for most design analysis is the critical path. Due to the nature of Hyperflex architecture and the availability of the Hyper Retimer, it is best to start you timing closure activities from the Retiming Limit Report. Provide the Hyper-Retimer as many optimization opportunities as possible, before having to look into more time intensive and potentially manual timing closure techniques.

Related Information

Hyperflex Architecture High-Performance Design Handbook

5.5.2.1. Retiming Limit Details Report

Use the Retiming Limit Details report to get specific information on what is currently limiting the Hyper Retimer from performing more optimizations.

The Retiming Limit Details report specifies:

- Clock Transfer: Clock domain, or the clock domain transfer for which the critical chain applies
- Limiting Reason: Design conditions which prevent further optimizations from happening.
- Critical Chain Details: Timing paths associated with the timing restrictions.

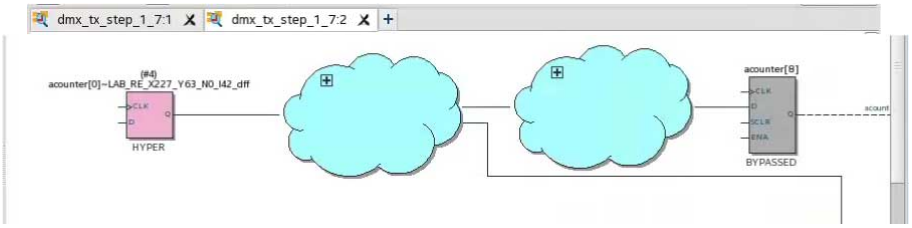
5.5.2.1.1. Using the Retiming Limit Details Report

To access the Retiming Limit Details report:

1. In the **Reports** tab, double-click **Retiming Limit Details** under **Fitter ➤ Retime Stage**.
2. To locate the critical chain in the Technology Map Viewer, right-click any path and click **Locate Critical Chain in Technology Map Viewer**.

   The Technology Map Viewer displays a schematic representation of the complete critical chain after place, route and register retiming.

Figure 36. Critical Chain in Technology Map Viewer



5.5.2.2. Fast Forward Timing Closure Recommendations

When running Fast Forward compilation, the Compiler removes signals from registers to allow mobility within the netlist for subsequent retiming. Fast Forward compilation generates design-specific timing closure recommendations, and predicts maximum performance with removal of all timing restrictions.

After you complete Fast Forward explorations, you can determine which recommendations to implement to provide the most benefit. Implement appropriate recommendations in your RTL, and recompile the design to achieve the performance levels that Fast Forward reports.

The Fast Forward Details Report provides the following information:

Table 14. Fast Forward Details Report Information

Name	Description
Step	Displays the various Fast Forward optimization steps, starting from the pre-optimization base compilation. • Each step comes with its associated critical chain. • Each step corresponds to a new optimization cumulative to the previous step.
Fast Forward Optimization	Analyzed Summary of the optimizations necessary to implement each step.
Estimated f_{MAX}	Estimated f_{MAX} performance after you implement the recommendations for this step in your design. This is cumulative, and step *n* represents the potential f_{MAX} after implementing all previous steps.
Optimization Analyzed	(cumulative) List of all the consecutive optimization steps applied.
Recommendation for Critical Chain	Lists recommended changes to your designs. These recommendations are geared towards removing retiming limitations, and allowing register movement.

5.5.2.2.1. Generating Fast Forward Timing Closure Recommendations

To generate Fast Forward timing closure recommendations:

1. On the Compilation Dashboard, click **Fast Forward Timing Closure Recommendations**.
   The Compiler runs prerequisite synthesis or Fitter stages as needed, and generates timing closure recommendations in the Compilation Report.
2. View timing closure recommendations in the Compilation Report to evaluate design performance, and implement key RTL performance improvements.

The Quartus Prime Pro Edition software allows you to automate or refine Fast Forward analysis:

- To run Fast Forward compilation during each full compilation, click **Assignments ➤ Settings ➤ Compiler Settings ➤ HyperFlex**, and turn on **Run Fast Forward Timing Closure Recommendations during compilation**.
- To modify how Fast Forward compilation interprets specific I/O and block types, click **Assignments ➤ Settings ➤ Compiler Settings ➤ HyperFlex Advanced Settings**.

5.5.2.2.2. Implementing Fast Forward Recommendations

After implementing timing closure recommendations in your design, you can rerun the Retime stage to obtain the predictive performance gains.

You can continue exploring performance and implementing RTL changes to your code until you reach the desired performance target. Once you have completed all the modifications you want to do, continue your timing closure activities with the traditional techniques explained in this document.

For more information about implementing Fast Forward timing closure recommendations in your design, refer to the *Implement Fast Forward Recommendations* section of the *Hyperflex Architecture High Performance Design Handbook*

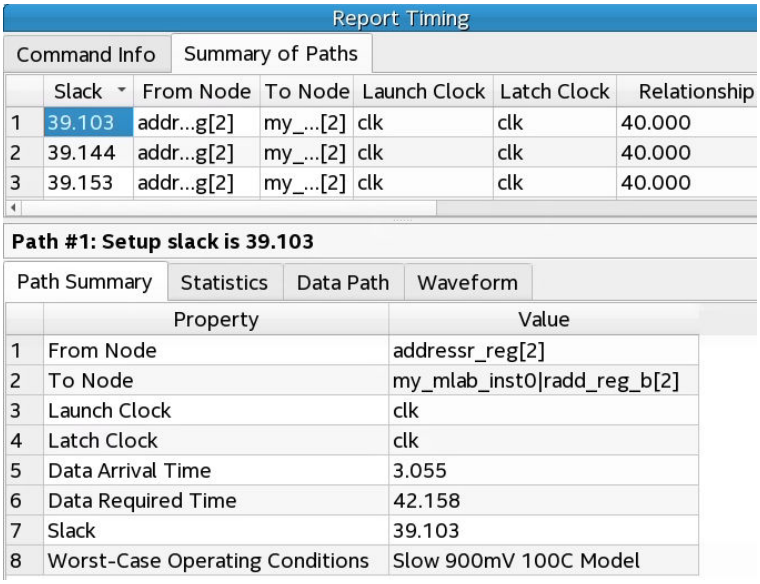
5.5.3. Review Timing Path Details

Reporting the timing paths and routing details can help uncover correctable timing and routing delays and other conditions that prevent retiming registers for higher performance.

5.5.3.1. Report Timing

The Timing Analyzer's **Reports ➤ Timing Slack ➤ Report Timing...** command allows you to specify settings to report the timing of any path or clock domain in the design. The equivalent scripting command is `report_timing`.

Figure 37. Report Timing Report



You can specify diverse options to customize the reporting. You can specify the **Clocks** and **Targets** that the report displays, the **Analysis Type** to run, whether to display **Extra Info** in the report, and the **Output** options for the report. For example, you can increase the number of paths to report, add a **Target** filter, and add a **From Clock**.

Figure 38. Report Timing Dialog Box (Top Section)

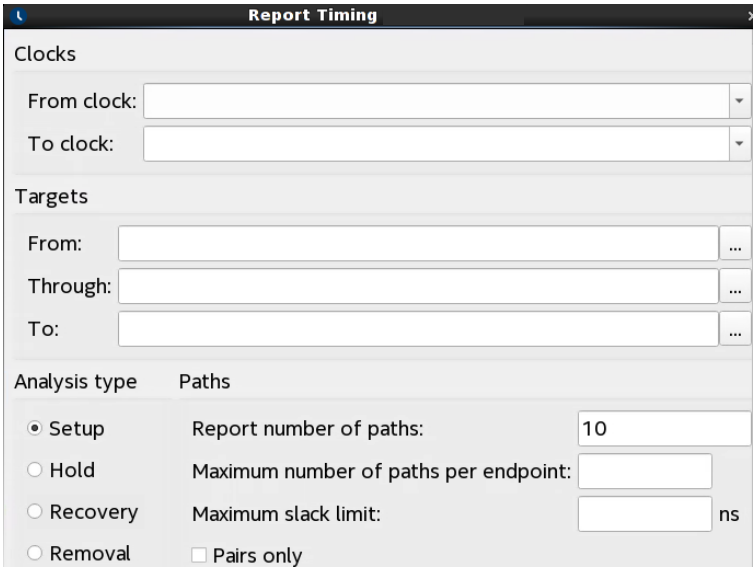


Figure 39. Report Timing Dialog Box (Bottom Section)

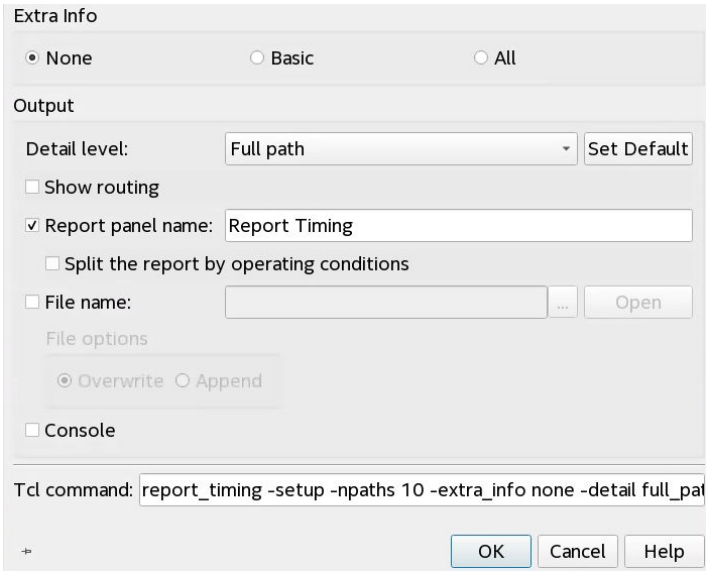


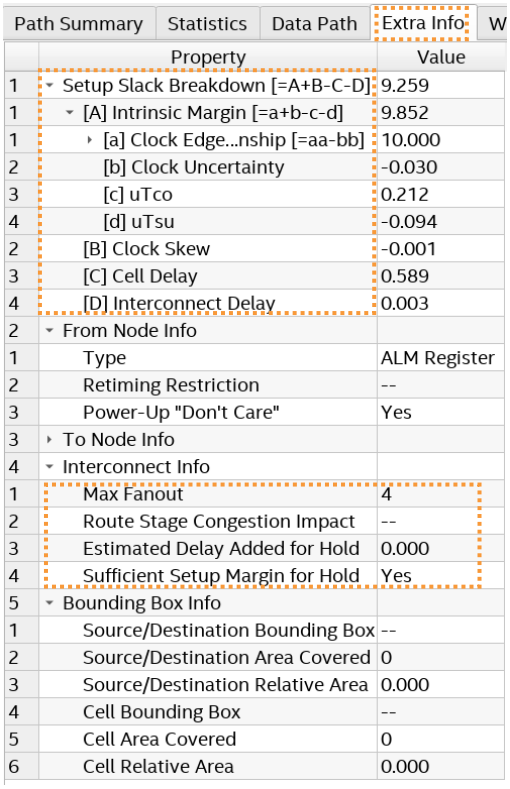
Table 15. Report Timing Settings

Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Targets	Specifies the target node for **From Clock** and **To Clock** to report paths with only those endpoints. Specify an I/O or register name or I/O port for this option. The field also supports wildcard characters. For example, to report only paths within a specific hierarchy: `report_timing -from *|egress:egress_inst|* \` `  -to *|egress:egress_inst|* -(other options)` When the **From**, **To**, or **Through** boxes are empty, the Timing Analyzer assumes all possible targets in the device. The **Through** option limits the report to paths that pass through combinatorial logic, or a particular pin on a cell.
Analysis type	The **Analysis type** options are **Setup**, **Hold**, **Recovery**, or **Removal**. The Timing Analyzer reports the results for the type of analysis you select.
Paths	Specifies the number of paths to display by endpoint and slack level. The default value for **Report number of paths** is 10, otherwise, the report can be very long. Enable **Pairs only** to list only one path for each pair of source and destination. Limit further with **Maximum number of paths per endpoints**. You can also filter paths by entering a value in the **Maximum slack limit** field.
Extra Info	Provides additional data that is relevant for diagnosing timing failure root cause, such as setup slack breakdown, and unexpected routing detours caused by congestion and hold time fix-up. Specify whether to include **None**, **Basic**, or **All** extra information in the report. The **Extra Info** tab data can help you identify potential, unnecessary routing detours, as well as placement or circuit issues that restrict the path f_{MAX} performance. Refer to Setup Slack Breakdown On the Extra Info Tab on page 81. • **All**—report includes **Extra Info** tab that reports extra information for source timing endpoints that pass through the unregistered output of a RAM or DSP block, or for destination timing endpoints that pass through the unregistered input of a DSP block. The **Data Path** tab includes Estimated Delay Added for Hold and Route Stage Congestion Impact data. • **Basic**—report includes the **Extra Info** tab but no extra information on the **Data Path** tab. • **None**—report includes no **Extra Info** tab or other extra information on the **Data Path** tab.
Output	Specify the path types the analysis includes in output for **Detail level**:

continued...

Option	Description
	• **Summary**—level includes basic summary reports. Review the **Clock Skew** column in the **Summary** report. If the skew is less than +/-150ps, the clock tree is well balanced between source and destination. • **Path only**—displays all the detailed information, except the **Data Path** tab displays the clock tree as one line item. • **Path and Clock**—displays the same as **Path only** with respect to the clock. • **Full path**—when higher clock skew is present, enable the **Full path** option. This option breaks the clock tree into greater detail, showing every cell, including the input buffer, PLL, global buffer (called CLKCTRL_), and any logic. Review this data to determine the cause of clock skew in your design. Use the **Full path** option for I/O analysis because only the source clock or destination clock is inside the FPGA, and therefore the delay is a critical factor to meet timing.
Show routing	Shows routing data in the report.
Split the report by operating conditions	For the operating condition timing corners, subdivides the data by each operating condition.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

Figure 40. Extra Info Tab

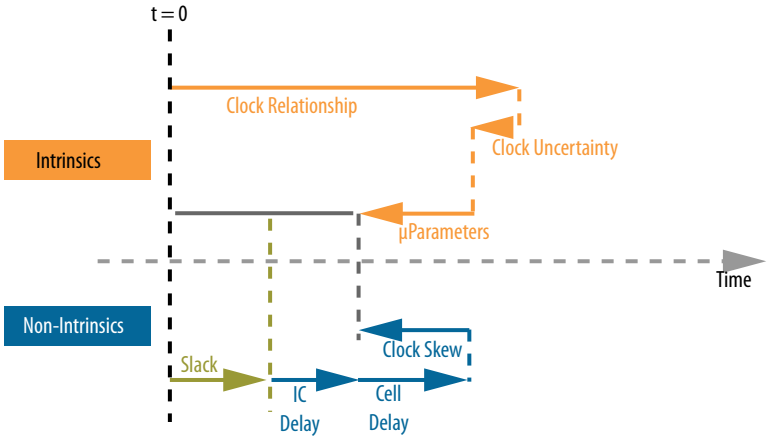


Setup Slack Breakdown On the Extra Info Tab

The **Extra Info** tab contains other timing metrics to help you diagnose timing closure issues, including **Setup Slack Breakdown** for the path.

The slack of a path specifies the margin by which the path meets its timing requirement. The setup slack breakdown is a numeric value that the Timing Analyzer calculates from the following timing requirements and path element delays:

Figure 41. Setup Slack Breakdown Calculations



A path can fail timing requirements for many varied reasons. For example, the clock relationship can be impossibly tight, or there can be excessive routing delays that alone cause failure for the timing path. Calculating the intrinsic margin of a timing path, and then comparing that margin to other delays of the path, can help identify the specific reasons why a path fails its timing requirement.

The **Extra Info** tab can help you identify potential significant or unexpected routing detours caused by congestion and hold time fix-up. The **Extra Info** tab can also report extra information for source timing endpoints that pass through the unregistered output of a RAM or DSP block, or for destination timing endpoints that pass through the unregistered input of a DSP block.

You can review the **Extra Info** data and **Locate Path** or **Locate Chip Area** in Chip Planner, Technology Map Viewer, or Resource Property Viewer to determine whether to make changes to improve placement and routing.

Some delay elements are more sensitive to a path's placement and routing than others. Intrinsic delays that are part of **Setup Slack Breakdown** are less sensitive to placement and routing, and are inherent in the RTL and timing requirements. Non-intrinsic delays are the other delays that are sensitive to placement and routing.

Table 16. Extra Info Tab Data

Extra Info Data	Description
Intrinsic Margin	Reports the intrinsic and non-intrinsic timing elements that comprise the timing path slack value. Intrinsic margin is a numeric value that the Timing Analyzer calculates from the timing requirements and path element delays. The Timing Analyzer also derives the slack of the path from the same requirements and delays, but with a different calculation. Intrinsic delays are less sensitive to placement and routing, and are inherent in the RTL and timing requirements. Non-intrinsic delays are the other delays that are sensitive to placement and routing.
From Node Info	Specifies the node Type, any Retiming Restriction, and any Power-Up "Don't Care" attributes for the From Node. Consider removing the retiming restriction to allow retiming and improve performance for timing closure.
To Node Info	Specifies the node Type, any Retiming Restriction, and any Power-Up "Don't Care" attributes for the To Node. Consider removing the retiming restriction to allow retiming and improve performance for timing closure.
Max Fanout	Reports the maximum fan-out of register and combinational nodes in the path.
Route Stage Congestion Impact	Reports whether routing has a **Low**, **Medium**, or **High** impact on congestion. A **Low** value suggests timing issues are not congestion related. A **High** value suggests competition for scarce routing resources plays a role in poor timing.
Estimated Delay Added for Hold	Reports the estimated amount of delay added on to the fastest delay route to satisfy hold. This value can help you determine whether delays are routing congestion or Hold related.
Sufficient Setup Margin for Hold	Reports whether the setup margin is suitable for the hold timing. **Yes**, indicates that the setup margin is sufficient. **No** indicates that the setup margin is insufficient for hold timing.
Source/Destination Bounding Box	Reports the lower-left and upper-right coordinates for the boundary box enclosing the source and destination registers. In an ideal case, the **Source/Destination Bounding Box**, **Cell Bounding Box**, and **Interconnect Bounding Box** values are roughly the same, and the relative areas are approximately 1.0. If the cell bounding box size grows relative to the **Source/Destination Bounding Box**, that can indicate a potential unnecessary routing detour on the path.
Source/Destination Area Covered	Reports the total area covered in terms of LABs.
Source/Destination Relative Area	Reports the area for the source and destination, relative to the **Source/Destination Bounding Box**. The value is always 1.0, which equals the same size.
Cell Bounding Box	Reports the lower-left and upper-right coordinates for the boundary box enclosing the source and destination registers, and any cells in the path.
Cell Area Covered	Reports the area for the cell, relative to the **Source/Destination Bounding Box**. A value of 1.0 equals the same size. A value greater than 1.0 can indicate a path has a cell outside of the space between the registers in the path.

The following describe the interpretation of timing conditions indicated by the **Setup Slack Breakdown**:

- **When the Setup Slack Breakdown is less than 0**—the path has such a tight timing relationship, a significant difference in microparameters, or such significant clock source uncertainty, that the path fails before the addition of any delay. Review the SDC constraints to verify that the timing relationship is correct. An incorrect relationship can exist between unrelated clocks that lack the proper timing cut. Ensure that parameterizable hard blocks (such as 20K RAM and DSP blocks) are fully registered. Investigate clock sources to verify that the clocks use global signals for routing.
- **When the clock skew exceeds the Setup Slack Breakdown**—address the clock transfer to meet timing on the path. You may need to create clock region assignments. You might also need to redesign cross-clock transfers to switch from synchronous to asynchronous implementation, such as with a FIFO or other handshake.

- **When the cell delay is greater than its intrinsic margin**—reduce the cell delay, as the path would fail timing even if the clocks are perfect and use no routing wires. Rewrite RTL to reduce the logic depth, restructure logic to allow the Compiler to use faster LUT inputs, or unblock retiming optimizations. The Compiler can automatically retime registers to reduce logic depth, but only in ways that maintain functionality and that the device architecture supports. To unblock the Hyper-Retimer, remove asynchronous resets and initial conditions.
- **When the interconnect delay is greater than its intrinsic margin**—the path would fail timing even if the clocks are perfect, and there is no logic. This occurs if registers are too far apart, or a timing path detours around a congested chip area. Review the fan-in and fan-out of registers that are far apart. Apply Logic Lock regions so the Fitter places the registers closer together. Use Logic Lock regions only after determining why placement is initially poor.

Related Information

Hyperflex Architecture High-Performance Design Handbook

5.5.3.2. Report Logic Depth

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Logic Depth...** command allows you to report the number of logic levels within a clock domain. This value typically corresponds to the number of look-up tables (LUTs) that a path passes through.

The equivalent scripting command is `report_design_metrics -logic_depth`. **Report Logic Depth** shows the distribution of logic depth among the critical paths, allowing you to identify areas where you can reduce logic levels in your RTL.

Figure 42. Report Logic Depth (Histogram)

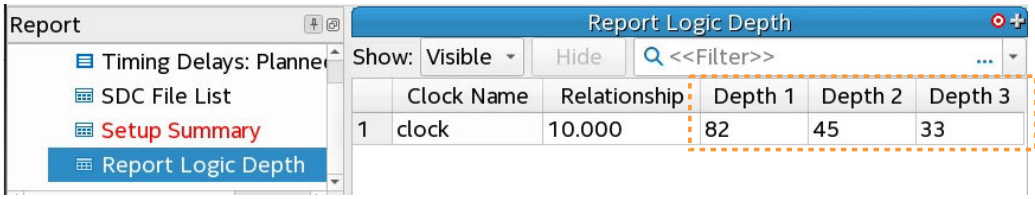


Figure 43. Report Paths of Depth 3

Call report logic depth by topology for each clock, intraclock only.

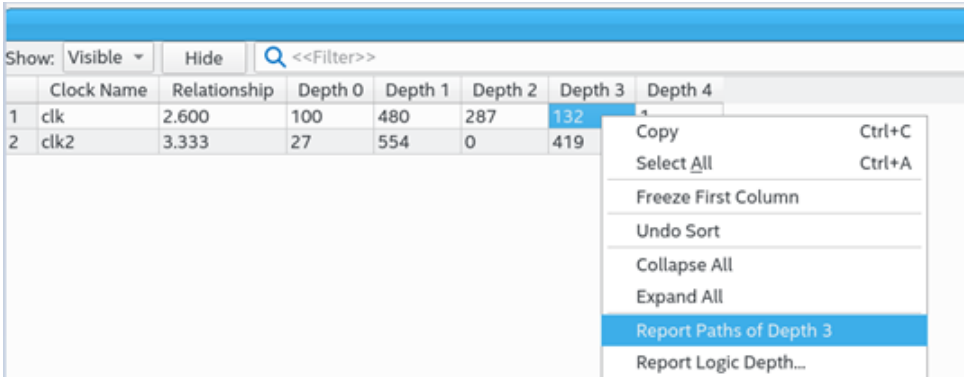
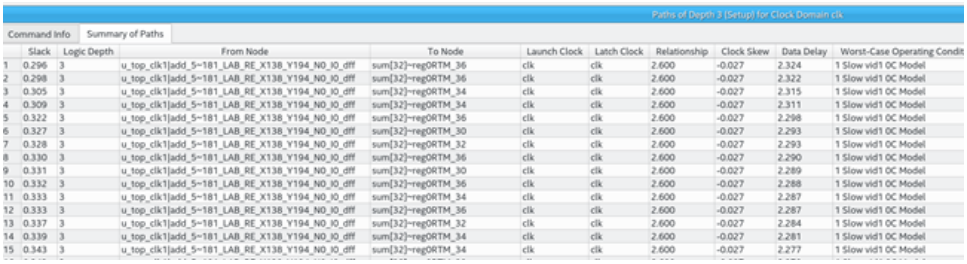


Figure 44. Summary of Paths

Close timing with accurate histogram cross probing.



	Slack	Logic Depth	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay	Worst-Case Operating Condit
1	0.296	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_36	clk	clk	2.600	-0.027	2.324	1 Slow vid1 0C Model
2	0.298	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_36	clk	clk	2.600	-0.027	2.322	1 Slow vid1 0C Model
3	0.305	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_34	clk	clk	2.600	-0.027	2.315	1 Slow vid1 0C Model
4	0.309	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_34	clk	clk	2.600	-0.027	2.311	1 Slow vid1 0C Model
5	0.322	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_36	clk	clk	2.600	-0.027	2.298	1 Slow vid1 0C Model
6	0.327	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_30	clk	clk	2.600	-0.027	2.293	1 Slow vid1 0C Model
7	0.328	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_32	clk	clk	2.600	-0.027	2.293	1 Slow vid1 0C Model
8	0.330	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_36	clk	clk	2.600	-0.027	2.290	1 Slow vid1 0C Model
9	0.331	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_30	clk	clk	2.600	-0.027	2.289	1 Slow vid1 0C Model
10	0.332	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_36	clk	clk	2.600	-0.027	2.288	1 Slow vid1 0C Model
11	0.333	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_34	clk	clk	2.600	-0.027	2.287	1 Slow vid1 0C Model
12	0.333	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_36	clk	clk	2.600	-0.027	2.287	1 Slow vid1 0C Model
13	0.337	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_32	clk	clk	2.600	-0.027	2.284	1 Slow vid1 0C Model
14	0.339	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_34	clk	clk	2.600	-0.027	2.281	1 Slow vid1 0C Model
15	0.343	3	u_top_clk1|add_5~181_LAB_RE_X138_Y194_N0_IO_dff	sum[32]~regORTM_34	clk	clk	2.600	-0.027	2.277	1 Slow vid1 0C Model

You can specify various options to customize the reporting.

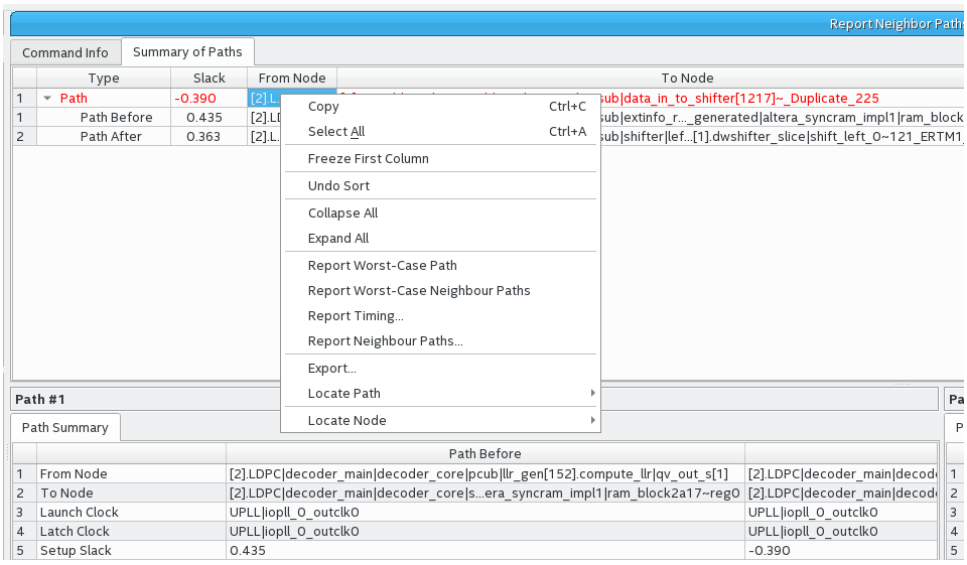
Table 17. Report Logic Depth Settings

Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Targets	Specifies the target node for **From Clock** and **To Clock** to report logic depth with only those endpoints. Specify an I/O or register name or I/O port for this option. The field also supports wildcard characters. When the **From**, **To**, or **Through** boxes are empty, the Timing Analyzer assumes all possible targets in the device. The **Through** option limits the report for paths that pass through combinatorial logic, or a particular pin on a cell.
Analysis type	The **Setup**, **Hold**, **Recovery**, and **Removal** analyses report the logic depths of the top *X* paths by slack. **Topology** analysis reports the logic depths of the top *X* paths by logic depth.
Paths	Specifies the number of paths to display by endpoint and slack level. The default value for **Report number of paths** is 10, otherwise, the report can be very long. Enable **Pairs only** to list only one path for each pair of source and destination. Limit further with **Maximum number of paths per endpoints**. You can also filter paths by entering a value in the **Maximum slack limit** field.
Detail	Specify whether to display on **Histogram** or full **Path** level of detail.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

5.5.3.3. Report Neighbor Paths

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Neighbor Paths...** command helps you to determine the root cause of critical paths (for example, high logic level, retiming limitation, sub-optimal placement, I/O column crossing, hold fix-up, time borrowing, or others). The equivalent scripting command is `report_design_metrics -neighbor_paths`.

Figure 45. Report Neighbor Paths Report



Report Neighbor Paths reports the most timing-critical paths in the design, including associated slack, additional path summary information, and path bounding boxes. **Report Neighbor Paths** shows the most timing-critical **Path Before** and **Path After** each critical **Path**. You can optionally view multiple before and after paths. Retiming or logic balancing of the **Path** can simplify timing closure if there is negative slack on the **Path**, but positive slack on the **Path Before** or **Path After**.

Table 18. Report Neighbor Path Dialog Box Settings

Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Targets	Specifies the target node for **From Clock** and **To Clock** to report neighbor paths with only those endpoints. Specify an I/O or register name or I/O port for this option. The field also supports wildcard characters. When the **From**, **To**, or **Through** boxes are empty, the Timing Analyzer assumes all possible targets in the device. The **Through** option limits the report for paths that pass through combinatorial logic, or a particular pin on a cell.
Analysis type	The **Analysis type** options are **Setup**, **Hold**, **Recovery**, or **Removal**. The Timing Analyzer reports the results for the type of analysis you select.
Paths	Specifies the number of paths to display by endpoint and slack level. The default value for **Report number of paths** is 10, otherwise, the report can be very long. Enable **Pairs only** to list only one path for each pair of source and destination. Limit further with **Maximum number of paths per endpoints**. You can also filter paths by entering a value in the **Maximum slack limit** field.
Report Number of Neighbor Paths	Specifies the number of neighbor paths to report, allowing you to view a number of the top adjacent paths entering the critical path, and the top paths exiting the critical path.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.
Extra Info	Specifies extra info.

5.5.3.4. Report Register Spread

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Register Spread...** command analyzes the final placement to identify registers with sinks pulling them in various directions. These registers are potential candidates for duplication. The equivalent scripting command is `report_register_spread`.

Registers that drive in opposite directions and connect to high fan-out can have placement-warping effects on the floorplan that impact f_{MAX}. The placement-warping may not cause timing failures. Therefore, you can view this report to identify such registers. Taking steps to address the registers listed in the report can make placement of the design easier and improve f_{MAX} performance.

You can automate duplication of registers with the `DUPLICATE_REGISTER` and `DUPLICATE_HIERARCHY_DEPTH` `.qsf` assignments, or you can manually modify RTL to duplicate registers or refactor logic. Refer to "Automatic Register Duplication: Hierarchical Proximity" in *Quartus Prime Pro Edition User Guide: Design Optimization*.

Figure 46. Report Register Spread Report

Report Register Spread

Show: Visible | Hide | <<Filter>>

Register Name	Register_Location	Number of Endpoints	Endpoint Centroid	Total Distance of Endpoints
counter_a[0]~ERTM	(69, 67)	78	(69, 67)	117.0
counter_a[1]~ERTM	(68, 67)	39	(68, 66)	21.4
counter_b[1]~ERTM	(70, 67)	39	(70, 67)	21.4
counter_b[2]~ERTM	(70, 67)	38	(70, 67)	20.9
counter_a[2]~ERTM	(68, 67)	38	(68, 66)	20.9
counter_b[3]~ERTM	(70, 67)	37	(70, 66)	20.4

You can specify various options to customize the report.

Table 19. Report Register Spread Settings

Option	Available Settings
Spread Type	Specifies the type of spread data in the report: • **Tension**—reports the sum over each sink of the distance from it to the centroid of all the sinks. • **Angle**—reports how far around the source register the fan-outs wrap, expressed from 0 to 360 degrees. This value corresponds to 360 minus the maximum angle between any two angularly adjacent sinks. This metric complements **Tension** by identifying registers which are surrounded by their sinks in all directions, and not those registers only being pulled in a few directions. • **Span**—reports the maximum 1-dimensional delta between the left bottom-most sink and the right top-most sink. • **Area**—reports the coverage of the sinks by number of LABs on the FPGA device. This option multiplies the span of the sinks in both X- and Y- dimensions. This metric complements **Span** by incorporating both dimensional spans of the sinks, and not only the maximum sink. • **Count**—reports registers with the largest sink counts.
Sink Type	Specifies the type of sink in the report: • **Endpoint**—the nodes (usually registers) that terminate timing paths from a register. • **Immediate Fanout**—the immediately connected nodes of the register. For example, lookup tables, other registers, RAM, or DSP blocks.
From Clock	Filters paths in the report to show only the launching clocks you specify.

continued...

Option	Available Settings
To Clock	Filters paths in the report to show only the latching clocks you specify, allowing you to debug one clock at a time.
Report number of registers	Specifies the number of registers to display in the report. The default value for **Report number of registers** is 10.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

Figure 47. Report Register Spread Types

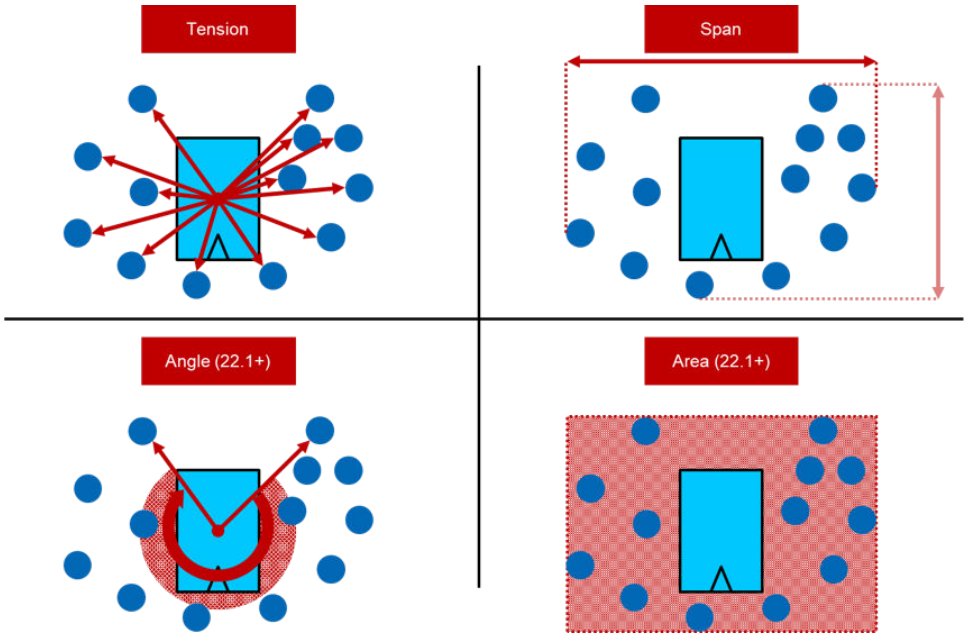
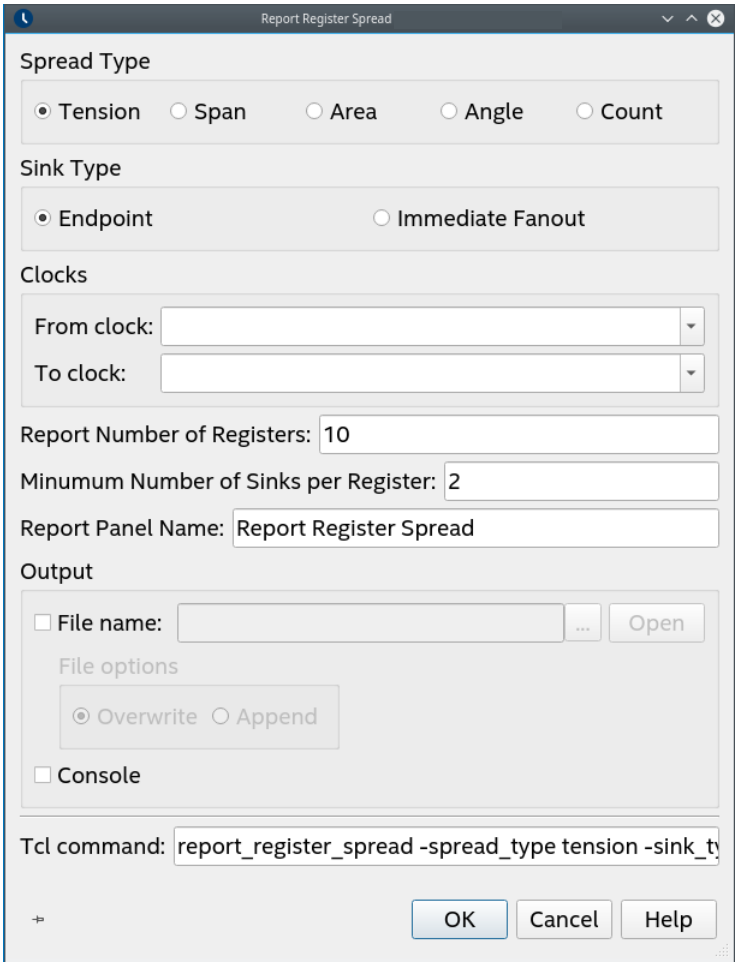


Figure 48. Report Register Spread Dialog Box



5.5.3.5. Report Route Net of Interest

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Route Net of Interest...** command allows you to report the nets that require the most effort from the router. The report shows the percentage of total router effort for the nets reported. The equivalent scripting command is `report_route_net_of_interest`.

This report allows you to identify nets that should not require significant router effort. For example, you might expect that low speed management interface nets are not timing critical, and therefore not require much router effort. However, if **Report Route Net of Interest** reports that some nets in the low speed management interface require significant effort from the router, you can investigate that further. The investigation can determine whether the timing constraints are correct, whether the fan-out is significant and can reduce through driver duplication, or whether the net passes through congested areas.

Figure 49. Report Route Net of Interest Report

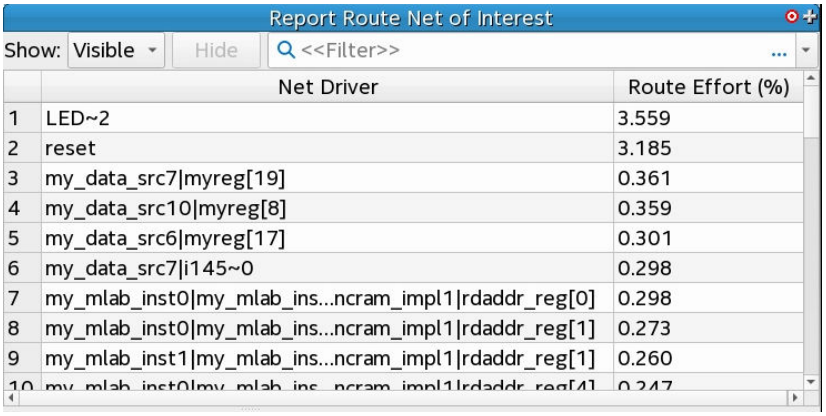
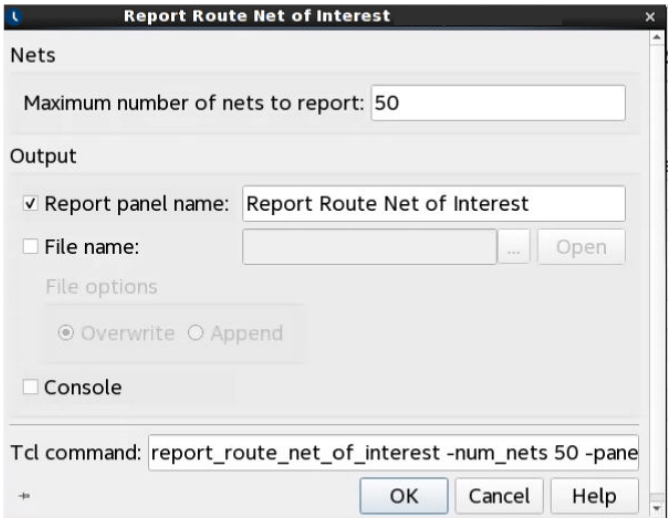


Figure 50. Report Route Net of Interest Dialog Box



From the Route Net of Interest Report in the Timing Analyzer GUI, you can right-click on any net and run Report Timing for more details about the net, its slack, and any of the net's paths.

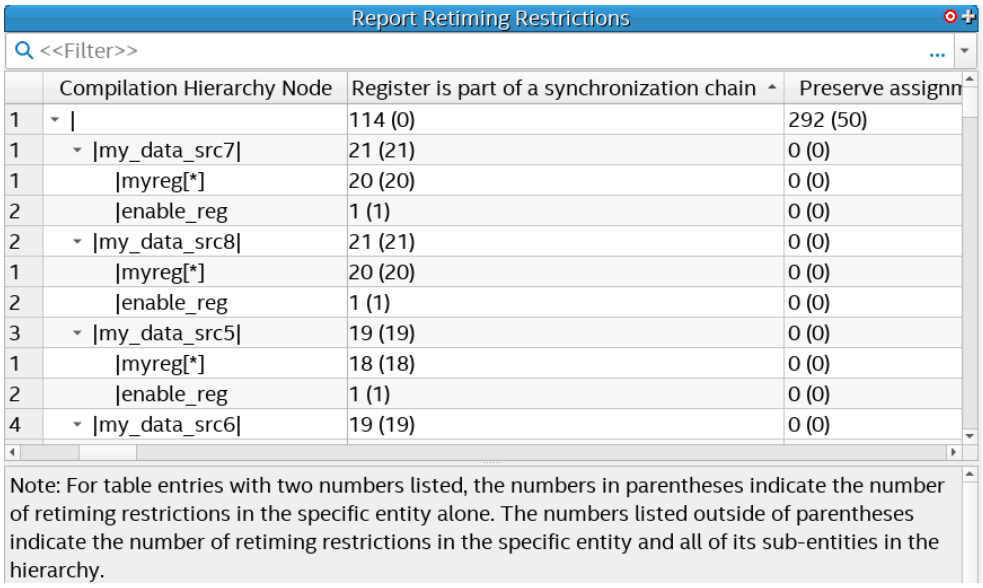
Table 20. Report Route Net of Interest Settings

Option	Available Settings
Nets	Specifies the **Maximum number of nets to report**. The default value is 50.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

5.5.3.6. Report Retiming Restrictions

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Retiming Restrictions...** command allows you to report the occurrences of design conditions that restrict Hyper-Retiming, such as Power-up "Care" restrictions, and don't touch or preserve attributes for each port. You can refer to this report to improve the circuit and remove retiming restrictions that limit circuit performance. `report_retiming_restrictions` is the equivalent scripting command.

Figure 51. Report Retiming Restrictions Report



For table entries with two number values, the number in parentheses indicates the number of retiming restrictions in the specific entity alone. The number listed outside of parentheses indicates the number of retiming restrictions in the specific entity and all of its sub-entities in the `hierarchy`.

Related Information

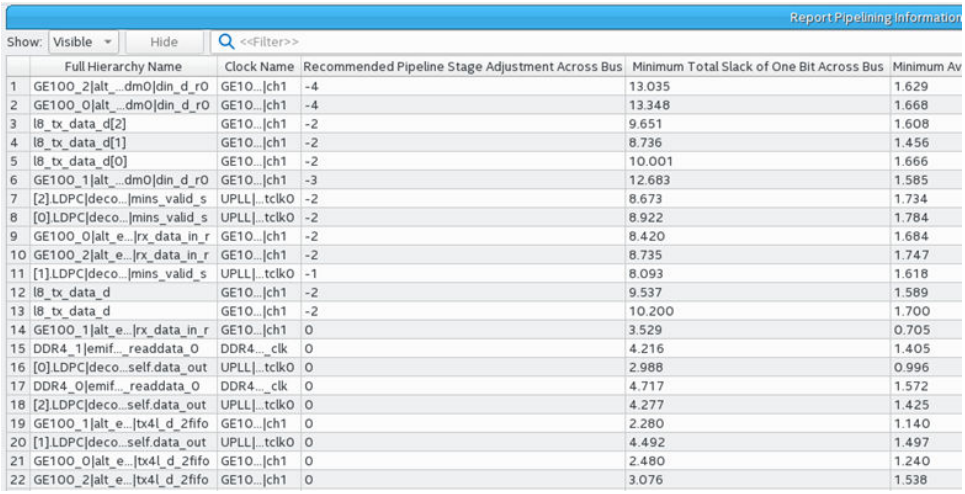
Retiming Restrictions and Workarounds, Hyperflex Architecture High-Performance Design Handbook

5.5.3.7. Report Pipelining Information

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Pipelining Information...** command allows you to generate a report that can help you to identify potential areas of over-pipelining in your design. Excessive pipelining unnecessarily consumes area. The equivalent scripting command is `report_pipelining_info`.

Report Pipelining Information... does not perform any functional analysis in making the recommended pipeline stage adjustment. You must be aware of any potential functional changes from removing pipeline stages. There may be circumstances when all the stages in a register pipeline are necessary for functional reasons. The report helps to identify location with more registers than necessary for covering distance.

Figure 52. Report Pipelining Information Report

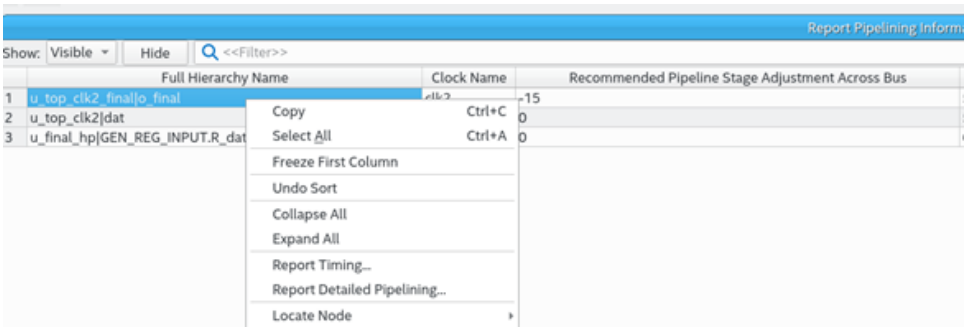


Report Pipelining Information

Show: Visible | Hide | <<Filter>>

	Full Hierarchy Name	Clock Name	Recommended Pipeline Stage Adjustment Across Bus	Minimum Total Slack of One Bit Across Bus	Minimum Av
1	GE10O_2|alt_...dm0|din_d_r0	GE10...|ch1	-4	13.035	1.629
2	GE10O_0|alt_...dm0|din_d_r0	GE10...|ch1	-4	13.348	1.668
3	l8_tx_data_d[2]	GE10...|ch1	-2	9.651	1.608
4	l8_tx_data_d[1]	GE10...|ch1	-2	8.736	1.456
5	l8_tx_data_d[0]	GE10...|ch1	-2	10.001	1.666
6	GE10O_1|alt_...dm0|din_d_r0	GE10...|ch1	-3	12.683	1.585
7	[2].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-2	8.673	1.734
8	[0].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-2	8.922	1.784
9	GE10O_0|alt_e...|rx_data_in_r	GE10...|ch1	-2	8.420	1.684
10	GE10O_2|alt_e...|rx_data_in_r	GE10...|ch1	-2	8.735	1.747
11	[1].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-1	8.093	1.618
12	l8_tx_data_d	GE10...|ch1	-2	9.537	1.589
13	l8_tx_data_d	GE10...|ch1	-2	10.200	1.700
14	GE10O_1|alt_e...|rx_data_in_r	GE10...|ch1	0	3.529	0.705
15	DDR4_1|emif..._readdata_0	DDR4..._clk	0	4.216	1.405
16	[0].LDPC|deco...self.data_out	UPLL|...tclk0	0	2.988	0.996
17	DDR4_0|emif..._readdata_0	DDR4..._clk	0	4.717	1.572
18	[2].LDPC|deco...self.data_out	UPLL|...tclk0	0	4.277	1.425
19	GE10O_1|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	2.280	1.140
20	[1].LDPC|deco...self.data_out	UPLL|...tclk0	0	4.492	1.497
21	GE10O_0|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	2.480	1.240
22	GE10O_2|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	3.076	1.538

Figure 53. Report Detailed Pipelining



Report Pipelining Inform

Show: Visible | Hide | <<Filter>>

	Full Hierarchy Name	Clock Name	Recommended Pipeline Stage Adjustment Across Bus
1	u_top_clk2_final|o_final		-15
2	u_top_clk2|dat		0
3	u_final_hp|GEN_REG_INPUT.R_dat		0

Copy Ctrl+C
Select All Ctrl+A
Freeze First Column
Undo Sort
Collapse All
Expand All
Report Timing...
Report Detailed Pipelining...
Locate Node

The detailed report shows every register in a tree structure. Over- or under-pipelining recommendations are in the main report. The following shows every single register inside the bus chain in a tree structure:

Figure 54. Detailed Pipelining Result

<<Filter>>

	Full Hierarchy Name	Slack	Manhattan Distance	Chain Depth
1	u_top_clk2_final|o_final[0]	58.579	46	21
1	u_final_dat_hp|GEN_REG_INPUT.R_data[0][0]	2.538	8	1
2	u_final_dat_hp|GEN_REG_INPUT.R_data[1][0]	2.632	7	1
3	u_final_dat_hp|GEN_REG_INPUT.R_data[2][0]	2.433	14	1
4	u_final_dat_hp|GEN_REG_INPUT.R_data[3][0]	2.878	0	1
5	u_final_dat_hp|GEN_REG_INPUT.R_data[4][0]	2.901	0	1
6	u_final_dat_hp|GEN_REG_INPUT.R_data[5][0]	2.872	0	1
7	u_final_dat_hp|GEN_REG_INPUT.R_data[6][0]	2.864	0	1
8	u_final_dat_hp|GEN_REG_INPUT.R_data[7][0]	2.867	0	1
9	u_final_dat_hp|GEN_REG_INPUT.R_data[8][0]	2.387	12	1
10	u_final_dat_hp|GEN_REG_INPUT.R_data[9][0]	2.735	3	1
11	u_final_dat_hp|GEN_REG_INPUT.R_data[10][0]	2.923	0	1
12	u_final_dat_hp|GEN_REG_INPUT.R_data[11][0]	2.919	0	1
13	u_final_dat_hp|GEN_REG_INPUT.R_data[12][0]	2.908	0	1
14	u_final_dat_hp|GEN_REG_INPUT.R_data[13][0]	2.887	0	1
15	u_final_dat_hp|GEN_REG_INPUT.R_data[14][0]	2.894	0	1
16	u_final_dat_hp|GEN_REG_INPUT.R_data[15][0]	2.832	0	1
17	u_final_dat_hp|GEN_REG_INPUT.R_data[16][0]	2.837	0	1
18	u_final_dat_hp|GEN_REG_INPUT.R_data[17][0]	2.809	0	1
19	u_final_dat_hp|GEN_REG_INPUT.R_data[18][0]	2.865	0	1
20	u_final_dat_hp|GEN_REG_INPUT.R_data[19][0]	2.743	2	1
21	final_dat[0]~reg0	2.855	0	1

To help identify potential over-pipelining, **Report Pipelining Information** reports:

- The recommended pipeline stage adjustment across bus
- The minimum total slack of one bit across bus
- The minimum average slack of one bit across bus
- The distance between the registers
- The width of buses in your design
- The number of sequential registers
- The number of registers on the bus

The Recommended Pipeline Stage Adjustment Across Bus reports the number of registers that you can remove from the bus for each bit. The Average Distance Per Stage, Max Distance Per Stage, and Min Distance Per Stage columns report the Manhattan distance measured in logic array blocks (LABs). The Bus Average Depth, Bus Max Depth, and Bus Min Depth columns report the number of sequential, single fan-out registers. For registers that have more than one clock source, the report lists the fastest one.

The **1+** sign under Recommended Pipeline Stage Adjustment Across Bus column means that the bus might need to add more registers to meet timing requirement. Refer to the Fast Forward Timing Closure Recommendations report.

If the report identifies a large register chain with multiple sequential registers, and the distance between registers is low, that condition can suggest over-pipelining. You may be able to remove some registers to recover some of the device area and reduce congestion.

The following options are available for this report:

Figure 55. Report Pipelining Information Dialog Box

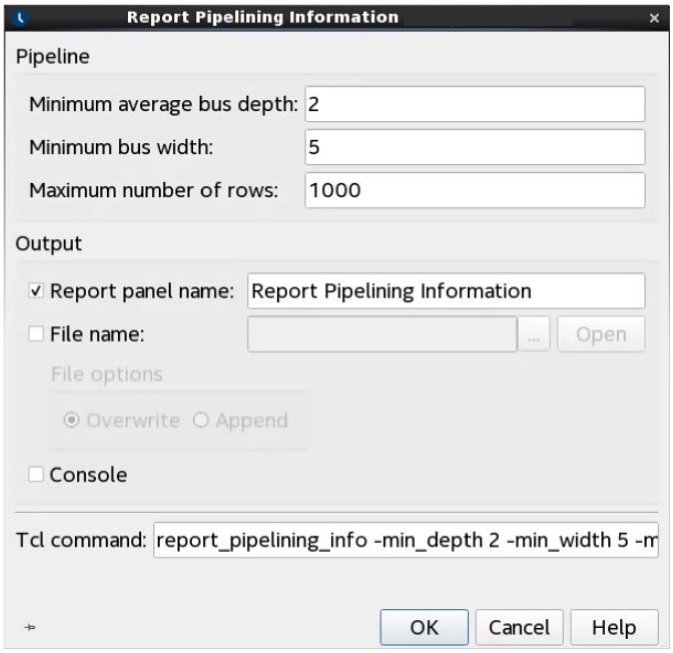


Table 21. Report Pipelining Information Settings

Option	Available Settings
Pipeline	Specifies the thresholds for reporting a register pipeline. You can define the **Minimum average bus depth**, the **Minimum bus width**, and the **Maximum number of rows** that the report includes.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

5.5.3.8. Report CDC Viewer

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report CDC Viewer...** command allows you to configure and display a custom clock domain crossing report and the Clock Domain Crossing (CDC) Viewer. The CDC Viewer graphically displays the setup, hold, recovery, or removal analysis of all clock transfers in your design. The equivalent scripting command is `report_cdc_viewer`.

Table 22. Report Clock Domain Crossing Viewer Settings

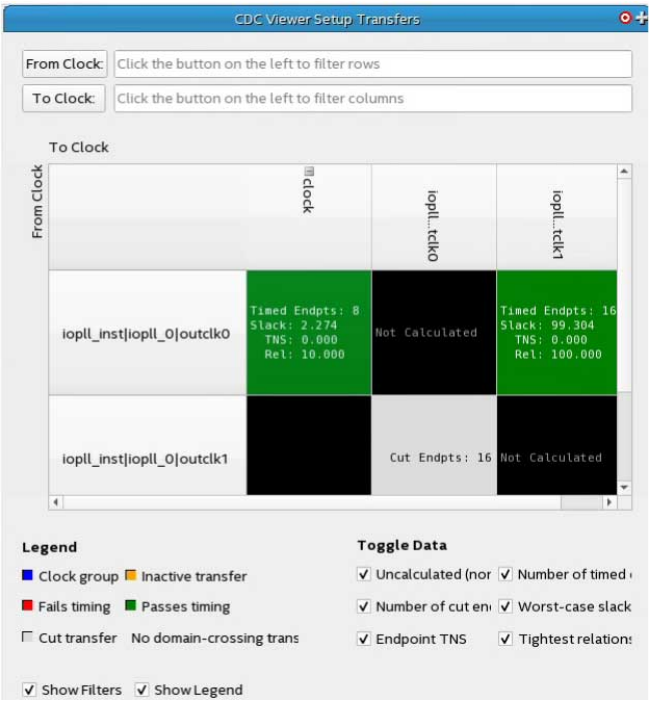
Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Analysis type	Options are **Setup**, **Hold**, **Recovery**, or **Removal**. The Timing Analyzer reports the results for the type of analysis you select.
Transfers	Specifies the type of clock transfers to include or exclude from the report, including **Timed transfers**, **Fully cut transfers**, **Clock groups, Inactive clocks**, and **Non-crossing transfers**. You can specify the **Maximum slack limit** and **Grid options** for the report.
Detail level	**Full** shows all details of the report and **Summary** filters the details and shows summary data.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data, and specify **Grid** or **List** format. *Note:* In grid format reports, clocks with non-crossing transfers always appear if they have transfers between other clocks.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

You can specify the following options to customize CDC Viewer reporting:

Table 23. CDC Viewer Report Controls

Control	Description
From Clock: and **To Clock:**	Filters the display according to the clock names you specify. Click **From Clock:** or **To Clock:** to search for specific clock names.
Legend	Defines the status colors. A color coded grid displays the clock transfer status. The clock headers list each clock with transfers in the design. The GUI truncates long clock names, but you can view the full name in a tool tip or by resizing the clock header cell. The GUI represents the generated clocks as children of the parent clock. A '+' icon next to a clock name indicates the presence of generated clocks. Clicking on the clock header displays the generated clocks associated with that clock.
Toggle Data	The text in each transfer cell contains data specific to each transfer. Turn on or off display of the following types of data: • **Number of timed endpoints** between clocks— the number of timed, endpoint-unique paths in the transfer. A path being "timed" means that analysis occurs on that path. Only paths with unique endpoints count towards this total. • **Number of cut endpoints** between clocks— the number of cut endpoint-unique paths, instead of timed paths. These paths are cut by either a false path or clock group assignment. Timing analysis skips such paths. • **Worst-case slack** between clocks— the worst-case slack among all endpoint-unique paths in the transfer. • **Total negative slack** between clocks— the sum of all negative slacks among all endpoint-unique paths in this transfer. • **Tightest relationship** between clocks— the lowest-value setup, hold, recovery, or removal relationship between the two clocks in this transfer.
Show Filters and **Show Legend**	Turns on or off Filters and **Legend**.

Figure 56. CDC Viewer Setup Transfers Report



Each block in the grid is a transfer cell. Each transfer cell uses color and text to display important details of the paths in the transfer. The color coding represents the following states:

Table 24. Transfer Cell Content

Cell Color	Color Legend
Black	Indicates no transfers. There are no paths crossing between the source and destination clock of this cell.
Green	Indicates passing timing. All timing paths in this transfer, that have not been cut, meet their timing requirements.
Red	Indicates failing timing. One or more of the timing paths in the transfer do not meet their timing requirements. If the transfer is between unrelated clocks, the paths likely require a synchronizer chain.
Blue	Indicates clock groups. The source and destination clocks of these transfers are cut by means of asynchronous clock groups.
Gray	Indicates a cut transfer. All paths in this transfer are cut by false paths. Therefore, timing analysis does not consider these paths.
Orange	Indicates inactive clocks. One of the clocks in the transfer is an inactive clock (with the `set_active_clocks` command). The Timing Analyzer ignores such transfers.

Right-click menus allow you to perform operations on transfer cells and clock headers. When the operation is a Timing Analyzer report or SDC command, a dialog box opens containing the contents of the transfer cell.

Table 25. Transfer Cell Right-Click Menus

Command	Description
Copy	Copies the contents of the transfer cell or clock header to the clipboard.
Report Timing	Reports timing. Not available for transfer cells with no valid paths (gray or black cells).
Report Endpoints	Reports endpoints. Not available for transfer cells with no cut paths (gray or black cells).
Report False Path	Reports false paths. Not available for transfer cells with no valid paths (black cells).
Report Exceptions	Reports exceptions. Only available for clock group transfers (blue cells).
Report Exceptions (with clock groups)	Reports exceptions with clock groups. Only available for clock group transfers (blue cells).
Set False Path	Sets a false path constraint.
Set Multicycle Path	Sets a multicycle path exception.
Set Min Delay	Sets a min delay constraint.
Set Max Delay	Sets a max delay constraint.
Set Clock Uncertainty	Sets a clock uncertainty constraint.

Table 26. Clock Header Right-Click Menus

Command	Description
Copy (include children)	Copies the name of the clock header, and the names of each of its derived clocks. This option only appears for clock headers with generated clocks.
Expand/Collapse All Rows/Columns	Shows or hides all derived clocks in the grid.
Create Slack Histogram	Generates a slack histogram report for the clock you select.
Report Timing From/To Clock	Generates a timing report for the clock you select. If you do not expand the clock to display derived clocks, the timing report includes all clocks that derive from the clock. To prevent this, expand the clock before right-clicking it.
Remove Clock(s)	Removes the clock you select from the design. If you do not expand the clock, timing analysis removes all clocks that derive from the clock.

You can view CDC Viewer output in any of the following formats:

- A report panel in the Timing Analyzer
- Output in the Timing Analyzer Tcl console
- A plain-text file
- An HTML file you can view in a web browser.

Related Information

Tips for Analyzing Failing Clock Paths that Cross Clock Domains on page 129

5.5.3.9. Timing Closure Recommendations

The **Report Timing Closure Recommendations** command in the Timing Analyzer **Task** pane analyzes paths and provides specific recommendations based on path characteristics. Since Design Assistant now provides more targeted timing closure recommendations, **Report Timing Closure Recommendations** is marked for deprecation.

5.5.3.10. Global Network Buffers

Routing paths allow you to identify global network buffers that fail timing. Buffer locations names reflect the network they drive.

- CLK_CTRL_Gn—for Global driver
- CLK_CTRL_Rn—for Regional driver

Buffers that access the global networks are in the center of each side of the device. Buffering to route a core logic signal on a global signal network causes insertion delay. Trade-offs to consider for global and non-global routing are source location, insertion delay, fan-out, distance a signal travels, and possible congestion if the signal demotes to local routing.

5.5.3.10.1. Source Location

If you cannot move the register feeding the global buffer closer, then consider changing either the design logic or the routing type.

5.5.3.10.2. Insertion Delay

If the design requires a global signal, consider adding half a cycle to timing by using a negative-edge triggered register to generate the signal, and use a multicycle setup constraint.

Figure 57. Negative-Edge Triggered Register

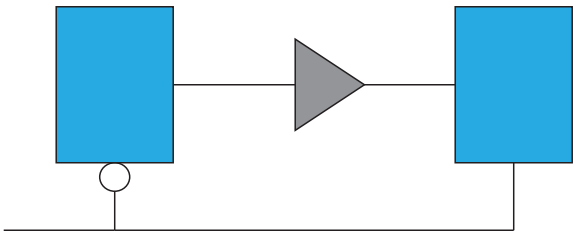
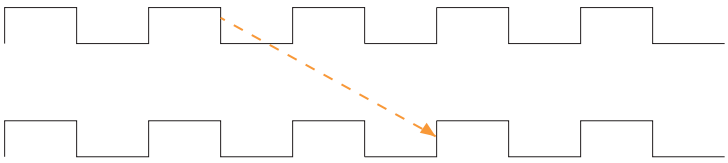


Figure 58. Multicycle Setup Constraint



```
set_multicycle_path -from <generating_register> -setup -end 2
```

5.5.3.10.3. Fan-Out

Nodes with very high fan-out that use local routing tend to pull logic that they drive close to the source node. This can make other paths fail timing. Duplicating registers can help reduce the impact of high fan-out paths. Consider manually duplicating and preserving these registers. Using a MAX_FANOUT assignment may make arbitrary groups of fan-out nodes, whereas a designer can make more intelligent fan-out groups.

5.5.3.10.4. Global Signal Assignment

You can use the Global Signal assignment to control the global signal usage on a per-signal basis. For example, if a signal needs local routing, you set the Global Signal assignment to **Off**.

5.5.3.11. Resets and Global Networks

The Compiler often routes reset signals on global networks. Sometimes, the use of a global network causes recovery failures. Consider reviewing the placement of the register that generates the reset and the routing path of the signal.

5.5.3.12. Suspicious Setup

Suspicious setup failures include paths with very small or very large requirements.

One typical cause is math precision error. For example, 10Mhz/3 = 33.33 ns per period. In three cycles, the time is 99.999 ns vs 100.000 ns. Setting a maximum delay can provide an appropriate setup relationship.

Another cause of failure are paths that must be false by design intent, such as:

- Asynchronous paths handled through FIFOs, or
- Slow asynchronous paths that rely on handshaking for data that remain available for multiple clock cycles.

To prevent the Fitter from having to meet unnecessarily restrictive timing requirements, consider adding false or multicycle path statements.

5.5.3.13. Auto Shift Register Replacement

During synthesis, the Compiler can convert shift registers or register chains into RAMs to save area. However, conversion to RAM often reduces speed. The Compiler names the converted registers with the prefix "altshift_taps".

- If paths that fail timing begin or end in shift registers, consider disabling the **Auto Shift Register Replacement** option. Do not convert registers that are intended for pipelining.
- For shift registers that are converted to a chain, evaluate area/speed trade off of implementing in RAM or logic cells.
- If a design uses nearly the full device capacity, you can save area by shifting register conversion to RAM, benefiting non-critical clock domains. You can change the settings from the default **AUTO** to **OFF** globally, or on a register or hierarchy basis.

5.5.3.14. Clocking Architecture

For better timing results, place all registers driven by a regional clock in one quadrant of the chip. You can review the clock region boundaries in the Chip Planner.

Timing failure can occur when the I/O interface at the top of the device connects to logic driven by a regional clock which is in one quadrant of the device, and placement restrictions force long paths to and from I/Os to logic across quadrants.

Use a different type of clock source to drive the logic, such as global, which covers the whole device, or dual-regional which covers half the device. Alternatively, you can reduce the frequency of the I/O interface to accommodate the long path delays. You can also redesign the pinout of the device to place all the specified I/Os adjacent to the regional clock quadrant. This issue can happen when register locations are restricted, such as with Logic Lock regions, clocking resources, or hard blocks (memories, DSPs, IPs).

The **Extra Fitter Information** tab in the Timing Analyzer timing report informs you when placement is restricted for nodes in a path.

Related Information

Viewing Available Clock Networks in Chip Planner on page 149

5.5.4. Try Optional Fitter Settings

This section focuses only on the optional timing-optimization Fitter settings, which are the **Optimize Hold Timing**, **Optimize Multi-Corner Timing**, and **Fitter Aggressive Routability Optimization**.

Caution: The settings that best optimize different designs might vary. The group of settings that work best for one design does not necessarily produce the best result for another design.

Related Information

Advanced Fitter Setting Dialog Box Help Topic
In *Quartus Prime Help*

5.5.4.1. Optimize Hold Timing

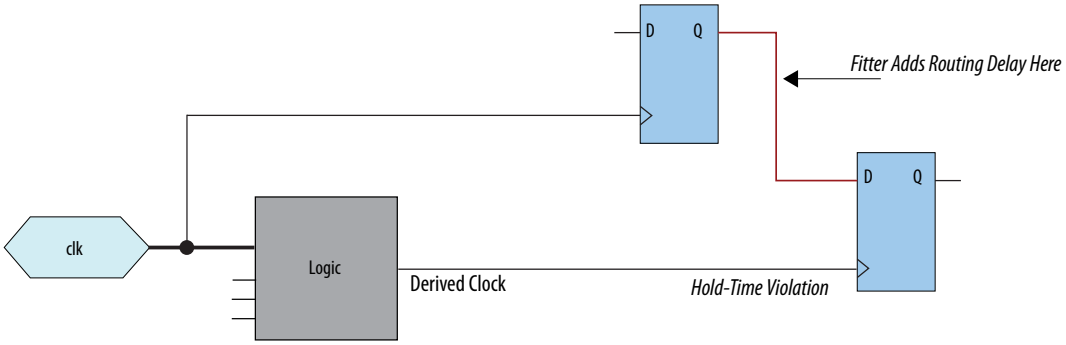
The **Optimize Hold Timing** option directs the Quartus Prime software to optimize minimum delay timing constraints.

When you turn on **Optimize Hold Timing** in the **Advanced Fitter Settings** dialog box, the Quartus Prime software adds delay to paths to ensure that your design meets the minimum delay requirements. If you select **I/O Paths and Minimum TPD Paths**, the Fitter works to meet the following criteria:

- Hold times (t_H) from the device input pins to the registers
- Minimum delays from I/O pins to I/O registers or from I/O registers to I/O pins
- Minimum clock-to-out time (t_{CO}) from registers to output pins

If you select **All Paths**, the Fitter also works to meet hold requirements from registers to registers, as highlighted in blue in the figure, in which a derived clock generated with logic causes a hold time problem on another register.

Figure 59. Optimize Hold Timing Option Fixing an Internal Hold Time Violation



However, if your design still has internal hold time violations between registers, you can manually add delays by instantiating LCELL primitives, or by making changes to your design, such as using a clock enable signal instead of a derived or gated clock.

Related Information

Quartus Prime Pro Edition User Guide: Design Recommendations

5.5.4.2. Fitter Aggressive Routability Optimization

The **Fitter Aggressive Routability Optimizations** logic option allows you to specify whether the Fitter aggressively optimizes for routability. Performing aggressive routability optimizations may decrease design speed, but may also reduce routing wire usage and routing time.

This option is useful if routing resources are resulting in no-fit errors, and you want to reduce routing wire use.

The table lists the settings for the **Fitter Aggressive Routability Optimizations** logic option.

Table 27. Fitter Aggressive Routability Optimizations Logic Option Settings

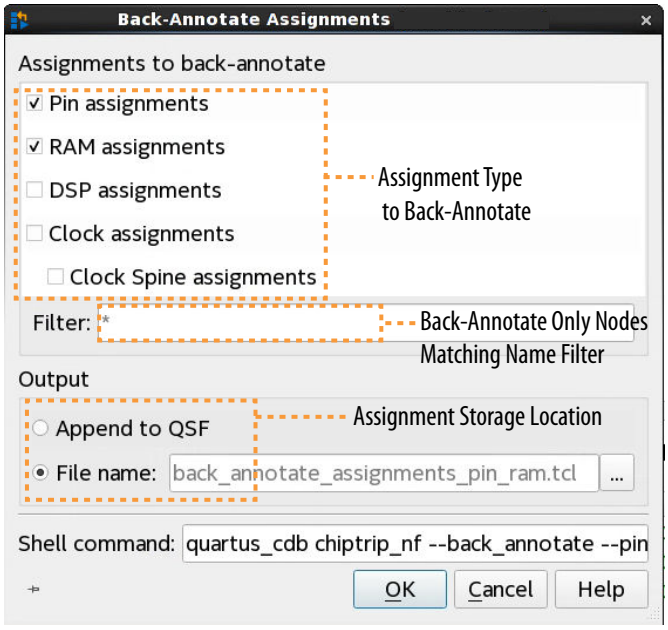
Settings	Description
Always	The Fitter always performs aggressive routability optimizations. If you set the **Fitter Aggressive Routability Optimizations** logic option to **Always**, reducing wire utilization may affect the performance of your design.
Never	The Fitter never performs aggressive routability optimizations. If improving timing is more important than reducing wire usage, then set this option to **Automatically** or **Never**.
Automatically	The Fitter performs aggressive routability optimizations automatically, based on the routability and timing requirements of the design. If improving timing is more important than reducing wire usage, then set this option to **Automatically** or **Never**.

5.5.5. Back-Annotating Optimized Assignments

The Compiler maps the elements of your design to specific device resources during fitting. After compilation, you can back-annotate (copy) the Compiler's resource assignments to preserve that same implementation in subsequent compilations. Back-annotation can simplify timing closure by allowing you to lock down placement of your optimized results.

Locking down placement of large blocks related to Clocks, RAMs, and DSPs can produce higher f_{MAX} with less noise. Large blocks like RAMs and DSPs have heavier connectivity than regular LABs, complicating movement during placement. When a seed produces good results from suitable RAM and DSP placement, you can capture that placement with back-annotation. Subsequent compiles can then benefit from the high quality RAM and DSP placement from the good seed.

Figure 60. Back-Annotate Assignments Dialog Box



To back-annotate (copy) the device resource assignments from the last compilation to the project `.qsf` (or to a Tcl file) for use in the next compilation:

1. Run a full compilation, or run the Fitter through at least the **Place** stage.
2. Click **Assignments ➤ Back-Annotate Assignments**.
3. Under **Assignments to back-annotate**, specify whether you want to preserve **Pin assignments**, **RAM assignments**, **DSP assignments**, **Clock assignments**, and **Clock Spine assignments** in the back-annotation.
4. In **Filter**, specify a text string (including wildcards) if you want to filter back-annotated assignments by entity name.
5. Under **Output**, specify whether to save the back-annotated assignments to the `.qsf` or to a Tcl file. A default Tcl file name displays.

Alternatively, you can run back-annotation with the following `quartus_cdb` executable. The **Shell command** field displays the shell command constructed by the options that you specify in the GUI.

```
quartus_cdb chiptrip_nf --back_annotate --pin --ram --dsp --clocks \
    --spines --file "<file>.tcl"
```

Note: Check available arguments by running `quartus_cdb <project> --back_annotate --help`.

5.5.6. Optimize Settings with Design Space Explorer II

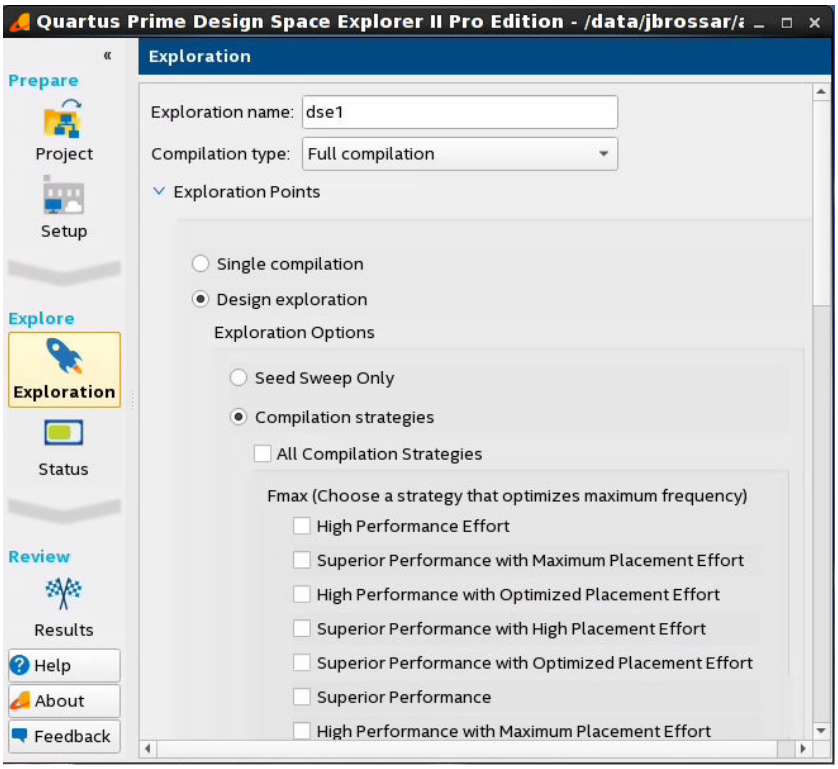
The Design Space Explorer II tool (**Tools ➤ Launch Design Space Explorer II**) allows you to find optimal project settings for resource, performance, or power optimization goals. Design Space Explorer II (DSE II) processes a design using combinations of settings and constraints, and reports the best combination of settings and constraints for the design. You can also take advantage of the DSE II parallelization abilities to compile on multiple computers.

In DSE II, an *exploration point* is a collection of Analysis & Synthesis, Fitter, and placement settings, and a group of exploration points is a *design exploration*. A design exploration can also include different fitter seeds.

DSE II compiles the design using the settings corresponding to each exploration point. When the compilation finishes, DSE II evaluates the performance data against an optimization goal that you specify. You can direct the DSE II to optimize for timing, area, or power.

If a design is close to meeting timing or area requirements, you can try different seeds with the DSE II, and find one seed that meets timing or area requirements.

Figure 61. Design Space Explorer II



You can run DSE II at any step in the design process; however, because large changes in a design can neutralize gains achieved from optimizing settings, Intel FPGA recommends that you run DSE II late in the design cycle.

Note: When comparing the results of different seed sweeps with DSE II, changing any of the following variables can cause differences in the compilation results between seed sweeps, resulting in a somewhat different fit in each case:

- The number or type of CPUs that DSE II uses to perform the seed sweeps
- Any change to the operating system
- Any change to source file content or location
- Any change to the Compiler settings or Timing Analyzer settings

For more information, refer to Fitter Seed on page 127.

Related Information

Using Design Space Explorer
21 Minute Online Course

5.5.6.1. DSE II Computing Resources

You can configure DSE II to take advantage of your computing resources to run the design explorations. In the DSE II GUI, the **Setup** page contains the job launch options, and the **Status** page allows you to monitor and control jobs.

DSE II supports running compilations on your local computer or a remote host through LSF, SSH or Torque. For SSH, you can also define a comma-separated list of remote hosts.

If you have a laptop or standard computer, you can use the single compilation feature to compile your design on a workstation with higher computing performance and memory capacity.

When running on a compute farm, you can direct the DSE II to safely exit after submitting all the jobs while the compilations continue to run until completion. Optionally, you can receive an e-mail when the compilations are complete.

If you launch jobs using SSH, the remote host must enable public and private key authentication. For private keys encrypted with a pass phrase, the remote host must run the ssh key agent to decrypt the private key, so the `quartus_dse` executable can access the key.

Note: Windows remote hosts require Cygwin's sshd server and PuTTY.

5.5.6.2. DSE II Optimization Parameters

DSE II provides a collection of predefined exploration spaces that focus on what you want to optimize. Additionally, you can define a set of compilation seeds. The number of explorations points is the number of seeds multiplied by the number of exploration modes.

Note: The availability of predefined spaces depends on the device family that the design targets.

In the DSE GUI, you specify these settings in the **Exploration** page.

Related Information

Exploration Page (Design Space Explorer II)
In *Quartus Prime Help*

5.5.6.3. DSE II Result Management

DSE II compares the compilation results to determine the best Quartus Prime software settings for the design. The **Report** page displays a summary of results.

In an exploration, DSE II selects the best worst-case slack value from among all timing corners across all exploration points. If you want to optimize for worst-case setup slack or hold slack, specify timing constraints in the Quartus Prime software.

Disk Space

By default, DSE II saves all the compilation data. You can save disk space by limiting the type of files that you want to save after a compilation finishes. These settings are in the **Exploration** page, **Results** section.

Reports

DSE II has reporting tools that help you quickly determine important design metrics, such as worse-case slack, across all exploration points.

DSE II provides a performance data report for all points it explores and saves the information in a `project-name.dse.rpt` file in the project directory. DSE II archives the settings of the exploration points in Quartus Prime Archive Files (`.qar`).

Related Information

Report Page (Design Space Explorer II)
In *Quartus Prime Help*

5.5.6.4. Running DSE II

Note: Before running DSE II, specify the timing constraints for the design.

This description covers the type of settings that you need to define when you want to run a design exploration. For details about all the options available in the GUI, refer to the Quartus Prime Help.

To perform a design exploration with the DSE II tool:

1. Start the DSE II tool.

   If you have an open project in the Quartus Prime software and launch DSE II, a dialog box appears asking if you want to close the Quartus Prime software. Click **Yes**.
2. In the **Project** page, specify the project and revision that you want to explore.
3. In the **Setup** page, specify whether you want to perform a local or a remote exploration, and set up the job launch.
4. In the **Exploration** page, specify optimization settings and goals.
5. When the configuration is complete, click **Start**.

5.5.7. Aggregating and Comparing Compilation Results with Exploration Dashboard

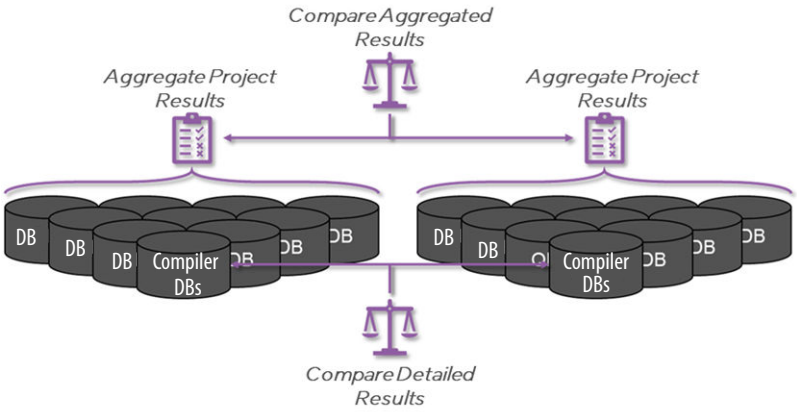
You can use the Exploration Dashboard (`quartus_edw`) in the Quartus Prime Pro Edition software to aggregate and compare compilation results between multiple Quartus Prime projects or sets of compilation results.

The Exploration Dashboard allows you to easily coordinate and view the compilation and timing results from multiple projects running on separate instances of the Quartus Prime software. For example, you can analyze and compare versions of the same design that differ by RTL changes, or perhaps only differ by project settings. The Exploration Dashboard provides the power and flexibility to support multi-project analysis for a diverse range of work flows and analysis tasks.(1)

The Exploration Dashboard interfaces with multiple Quartus Prime projects simultaneously in a single workspace to help you close timing by aggregating and comparing results from multiple seeds or multiple versions of your design.

- Aggregating compilation results—Exploration Dashboard reports what is common in all the compilation results for a version of your design.
- Comparing compilation results—Exploration Dashboard reports the differences between different versions of your design.

Figure 62. Exploration Dashboard Use Model



Use the Exploration Dashboard to quickly compare the aggregated compilation results from multiple projects or sets of results to determine the best implementation and impact of changes. The Exploration Dashboard supports use cases like the following:

- Identify all of the failing timing paths, in all seeds, after completing a seed sweep.
- Determine whether the average f_{MAX} improves after RTL optimization.
- Track a scorecard of design performance as the project proceeds towards completion.
- Compare compilation results across Quartus Prime software versions.

(1) Exploration Dashboard is pre-production status in Quartus Prime Pro Edition v.24.1, and supports analysis of compilation results generated with Quartus Prime Pro Edition software version 21.1 through version 24.1. Other versions may function but are not verified.

For example, you can use Exploration Dashboard to report timing on all of your various compilation seeds, and view the path from all seeds in a single report view. Exploration dashboard allows you to compare historical results against new data. This allows you to track improvement over time on critical design metrics, such as Fmax and power.

The Exploration Dashboard is currently a Tcl-based API that employs an object-property model to aggregate and compare objects across multiple compilation databases that store results from your different compilations.

Note: For a step-by-step tutorial using the Exploration Dashboard GUI and an example design, refer to *AN 1006: Multi-Project Analysis with Exploration Dashboard*.

Related Information

AN 1006: Multi-Project Analysis with Exploration Dashboard

5.5.7.1. Aggregation Use Case

Aggregation helps you see all the results from your compiles in one workspace. For example, you can use aggregation to determine whether particular timing failures occur commonly or occasionally. Focusing your optimization work on failures that occur commonly has higher impact than focusing on failures that occur occasionally.

Aggregation is helpful because of the stochastic nature of the Quartus Prime Compiler. This stochastic nature means that the Compiler employs random heuristics in some decision-making processes. These heuristics perform well on average, but certain sequences can have unusually good or bad outcomes.

Often, the seed value that governs these random decisions is a project input that you can change to explore for a given design without changing the design. Because these random decisions are sensitive to specific netlist topologies, constraints, and design file content ordering, various different project element changes can result in the "seed effect" without any meaningful change to the design. Therefore, any given compilation result is not a precise measure of the quality of the design. While an individual compilation is the only source of programming files, you can run multiple compilations with different seed values in parallel, with each project containing its own seed value, to obtain the best results.

When performing aggregation, you are usually looking for common trends and limitations among the results. Since the compiles that you analyze with aggregation are different compilations of the exact same design, any random differences between the projects should cancel out, and any fundamental issues should recur.

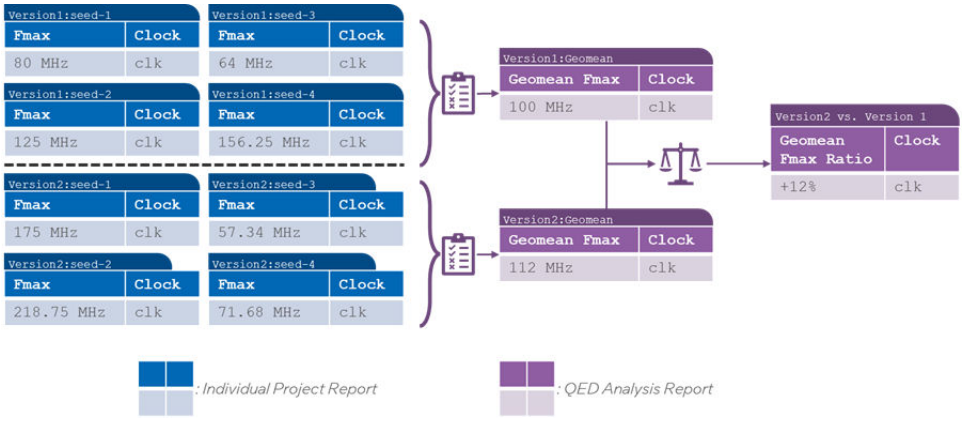
Nevertheless, there can be certain projects and seeds that are outliers. You can then compare to the aggregate of the remaining projects. You can use this method to either explain unusually good or unusually bad results in the outlier projects in comparison with typical results.

5.5.7.2. Comparison Use Case

Comparison of compilation results can be helpful in determining the set of conditions or properties that are different between two sets of compilation results. For example, you can compare the differences between sets of compilation results that you generate at different times from the similar source files. From the perspective of the Quartus Prime Compiler, such projects are technically different. You can also export Exploration Dashboard results for further mathematical analysis in other tools.

Comparison allows you to determine whether certain design changes result in the compilation results that you want. You can compare the aggregate-before results to the aggregate-after results. Exploration Dashboard can analyze multiple seeds of each version of the design together first, and then compare the common results across versions of the design. In comparison use cases, the goal is to decide if the design changes made the design better or worse overall.

Figure 63. Visualization of Fmax Aggregation and Comparison Analysis



Visualization of Fmax Aggregation and Comparison Analysis illustrates an example of aggregation followed by comparison. In this example use-case:

1. The designer first compiles “Version1” of their design four times through four different seed values.
2. The designer next makes some changes to the design for “Version2”.
3. The designer compiles “Version2” four times.

To determine if the “Version2” design changes improve the design performance overall, the designer can follow these steps:

1. Aggregate the Fmax results from each version by taking the geometric mean (geomean) of the Fmax across seeds.
2. Compare the geomean values and conclude that “Version2” has an expected +12% improvement to Fmax compared to "Version1".

The Exploration Dashboard is instrumental in this comparison because no other individual seed versus seed comparison can show this result. Comparison of the individual compilation results can lead to a different conclusion.

5.5.7.3. Exploration Dashboard Object-Property Model

The Exploration Dashboard operates on an object-property model that defines the following roles, responsibilities, and properties of the major object types:

Figure 64. Visualization of the Objects in the Exploration Dashboard Environment

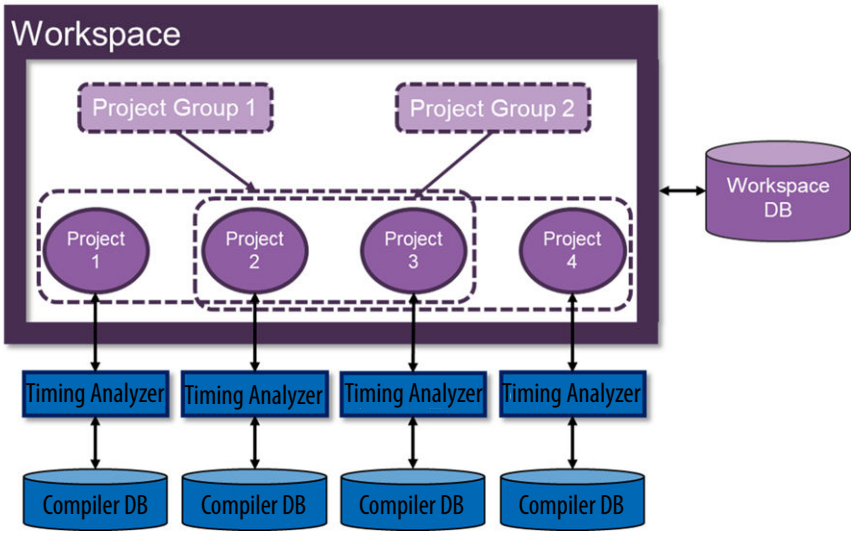


Table 28. Exploration Dashboard Terms

Term	Description
Workspace	The workspace is a single container for all other Exploration Dashboard objects, reports, and results. The workspace governs all persistence, namespace, and portability requirements of Exploration Dashboard flows. The properties of the workspace govern global settings and behaviors that do not pertain to any individual group or project.
Project Handle	Corresponds 1:1 with a Compiler database and the necessary configurations to access it. The purpose of the Project Handle is to launch a new Quartus Prime software process and send commands and data back and forth over a communication channel.
Project Group	Provides a method to refer to and work with an arbitrary subset of the Project Handles that are loaded in the workspace. Each project group can contain any number of projects. Each project ID present in a group's projects property corresponds to a project object that is guaranteed to have that group's ID present in its groups property.
Compiler database	The database that preserves the compilation results from one previous run of the Compiler.

Note: For the complete Exploration Dashboard Tcl API, refer to *::quartus::qed* in the *Quartus Prime Pro Edition User Guide: Scripting*.

Related Information

::quartus::qed, Quartus Prime Pro Edition User Guide: Scripting

5.5.7.3.1. Base Exploration Dashboard Properties

This section describes the Exploration Dashboard objects and properties that you can configure and use for aggregating and comparing compilation results in the Exploration Dashboard.

The following base objects and properties are common across all objects and accessible using `set_property` and `get_property`.

Table 29. Base Exploration Dashboard Objects and Properties

Property Name	Property Type	Property Description	Default Value	Read-Only	Comments
`type`	Either: `project`, `group`, or `workspace`	Type of the object	N/A (must be specified)	True	
`id`	string	Identifier associated with the object	Automatically generated based on the `type` (for example, `project_1`)		All IDs must be globally unique within a Workspace. Constructing an object causes an error if you specify an ID that is already allocated to another object.
`user_data`	string	Arbitrary data you want to store	N/A		You can optionally access `user_data` using methods described below.

::qed::set_user_data and ::qed::get_user_data Specializations

In addition to `set_property` and `get_property` `user_data` also supports the `set_user_data` and `get_user_data` specializations for access. These specializations allow for more convenience when using the recommended (but unrequired) method of managing `user_data` like a Tcl dictionary.

For example, you can use the `qed::get_property` command to retrieve the value of the `user_data` property. You could then treat it as a `dict` and use `dict get` or `dict set` commands to manipulate the value, and optionally store any changes with the `qed::set_property command`.

The following shows equivalent examples using the two methods discussed:

```
::qed::get_user_data <object> -key <key>
```

is equivalent to:

```
set tmp [::qed::get_property <object> -property user_data]
dict get $tmp <key>
```

Similarly,

```
::qed::set_user_data <object> -key <key> -value <value>
```

is equivalent to:

```
set tmp [qed::get_property <object> -property user_data]
dict set tmp <key> <value>
::qed::set_property <object> -property user_data -value $tmp
```

5.5.7.3.2. Project Handle Properties

The main purpose of Project Handle objects is to configure, launch, and manage a connection to a single project compilation database.

Table 30. Project Handle Properties

Property Name	Property Type	Property Description	Default Value	Read-Only	Comments
`qpf_path`	String (must be a valid path to a `.qpf` file)	Project file corresponding to the project and database to open and interface with	N/A		
`project_directory`	String	Directory name of the project file you interface with	N/A	True	Derived from the `qpf_path` value
`project_name`	String	Base name of the project file being interfaced with	N/A	True	Derived from the `qpf_path` value
`revision_name`	String	Revision name to specify	N/A		If unspecified, Exploration Dashboard assumes that the revision name is the default revision name for the project. This updates when you set the `qpf_path` property.
`connection_status`	Enum	Reflects the state of the communication channel after the `launch_connection` or `disconnect` methods complete successfully.	`CLOSED`	True	Can be one of `STARTING`, `READY`, `CLOSED`, `RUNNING`, `TIMEOUT`, or `ANY`
`groups`	List of Project Group IDs	Set of groups to which the project belongs.	N/A		A project can be in any number of groups. Each group ID present in a project's `groups` property corresponds to a group object that is guaranteed to have that project's ID present in its `projects` property . Project `groups` properties are kept consistent with group `projects` properties). *Note:* All projects must be in at least one group. Running `sanitize_workspace` creates and places all ungrouped projects into a default group.
`db_state`	Enum	Indicates whether the project's compilation database is loaded or not	`Unloaded`	True	Can be one of `unloaded` or `loaded`.

5.5.7.3.3. Project Group Objects and Properties

The main purpose of a Project Group is to provide a convenient way to refer to and work with an arbitrary subset of the Project Handles that are loaded in the workspace.

Table 31. Project Group Objects and Properties

Property Name	Property Type	Property Description	Default Value	Read-Only	Comments
`projects`	List of Project Group IDs	Set of projects belonging to the group	N/A		A group can contain any number of projects. Each project ID present in a group's `projects` property corresponds to a project object that is guaranteed to have that group's ID present in its `groups` property (group `projects` properties are kept consistent with project `groups` properties).

5.5.7.3.4. Workspace Objects and Properties

The main purpose of a workspace is to act as a single container for all other Exploration Dashboard objects, reports, and results. The workspace also handles persistence, namespace, and portability for Exploration Dashboard flows.

Table 32. Workspace Objects and Properties

Property Name	Property Type	Property Description	Default Value	Read-Only	Comments
`default_group_id`	Valid Group ID	ID of the default group that `sanitize_workspace` uses to ensure all projects are placed in at least one group.	`default_group_1`		The default group ID is reserved upon workspace construction, but the default group itself isn't constructed until required by `sanitize_workspace`. You can modify this property to designate a specific group as the default group for un-grouped projects.

5.5.7.4. Starting the Exploration Dashboard

To start the Exploration Dashboard using the Tcl API, follow these steps:

1. To start the Exploration Dashboard, perform one of the following:
   - To start Exploration Dashboard in shell mode with the Quartus Prime Pro Edition software, type the following command:
     ```
     quartus_ed -s
     ```
   - To start the Exploration Dashboard in GUI mode with the Quartus Prime Pro Edition software, type the following command:
     ```
     quartus_edw
     ```
2. To create a group in the workspace, type the following command and specify a group name:
   ```
   ::qed::create_object -type group <group name>
   ```
3. To add project objects to the workspace for exploration, type the following command to specify the type of object (project), a unique ID (for example based on the `seednumber`), and the path to the `.qpf` file:
   ```
   ::qed::create_object -type <project|group> -qpf_path <qpf path> <id>
   ```
   The following commands show examples of this syntax:
   ```
   ::qed::create_object -type project -qpf_path ../seed2/top.qpf seed2
   ```
   The Exploration Dashboard creates the `id` if unspecified. Repeat this step for each project object that you want to add to the workspace for aggregation and comparison.
4. To perform workspace legality checks and modify the state of your workspace to make it legal, type the following command:
   ```
   qed::sanitize_workspace
   ```

The Exploration Dashboard is ready to receive other commands to analyze, aggregate, and compare the compilations results from the project objects in the workspace. For the complete Exploration Dashboard Tcl API, search for *::quartus::qed* in the *Quartus Prime Pro Edition User Guide: Scripting*.

Note: For a step-by-step tutorial using the Exploration Dashboard GUI and an example design, refer to *AN 1006: Multi-Project Analysis with Exploration Dashboard*.

Related Information

- AN 1006: Multi-Project Analysis with Exploration Dashboard
- ::quartus::qed, Quartus Prime Pro Edition User Guide: Scripting

5.5.8. I/O Timing Optimization Techniques

This stage of design optimization focuses on I/O timing, including setup delay (t_{SU}), hold time (t_H), and clock-to-output (t_{CO}) parameters.

Before proceeding with I/O timing optimization, ensure that:

- The design's assignments follow the suggestions in the *Initial Compilation: Required Settings* section of the *Design Optimization Overview* chapter.
- Resource utilization is satisfactory.

Note: Complete this stage before proceeding to the register-to-register timing optimization stage. Changes to the I/O paths affect the internal register-to-register timing.

Summary of Techniques for Improving Setup and Clock-to-Output Times

The table lists the recommended order of techniques to reduce t_{SU} and t_{CO} times. Reducing t_{SU} times increases hold (t_H) times.

Note: Verify which options are available to each device family

Table 33. Improving Setup and Clock-to-Output Times

Order	Technique	Affects t_{SU}	Affects t_{CO}
1	Verify of that the appropriate constraints are set for the failing I/Os (refer to *Initial Compilation: Required Settings*)	Yes	Yes
2	Use timing-driven compilation for I/O (refer to *Fast Input, Output, and Output Enable Registers*)	Yes	Yes
3	Use fast input register (refer to *Programmable Delays*)	Yes	N/A
4	Use fast output register, fast output enable register, and fast OCT register (refer to *Programmable Delays*)	N/A	Yes
5	Decrease the value of **Input Delay from Pin to Input Register** or set **Decrease Input Delay to Input Register = ON**	Yes	N/A
6	Decrease the value of **Input Delay from Pin to Internal Cells** or set **Decrease Input Delay to Internal Cells = ON**	Yes	N/A
7	Decrease the value of **Delay from Output Register to Output Pin** or set **Increase Delay to Output Pin = OFF** (refer to *Fast Input, Output, and Output Enable Registers*)	N/A	Yes
8	Increase the value of **Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations** (refer to *Fast Input, Output, and Output Enable Registers*)	Yes	N/A
9	**Use PLLs to shift clock edges**	Yes	Yes
10	Increase the value of **Delay to output enable pin** or set **Increase delay to output enable pin** (refer to *Use PLLs to Shift Clock Edges*)	N/A	Yes

I/O Timing Constraints on page 114
Optimize IOC Register Placement for Timing Logic Option on page 114
Fast Input, Output, and Output Enable Registers on page 114
Programmable Delays on page 115
Use PLLs to Shift Clock Edges on page 116
Use Fast Regional Clock Networks and Regional Clocks Networks on page 116
Spine Clock Limitations on page 116

Related Information

Initial Compiler Settings on page 7

5.5.8.1. I/O Timing Constraints

Timing Analyzer supports the Synopsys* Design Constraints (SDC) format for constraining your design. When using the Timing Analyzer for timing analysis, use the `set_input_delay` constraint to specify the data arrival time at an input port with respect to a given clock. For output ports, use the `set_output_delay` command to specify the data arrival time at an output port's receiver with respect to a given clock. You can use the `report_timing` Tcl command to generate the I/O timing reports.

The I/O paths that do not meet the required timing performance are reported as having negative slack and are highlighted in red in the Timing Analyzer **Report** pane. In cases where you do not apply an explicit I/O timing constraint to an I/O pin, the Quartus Prime timing analysis software still reports the **Actual** number, which is the timing number that must be met for that timing parameter when the device runs in your system.

Related Information

Quartus Prime Pro Edition User Guide: Timing Analyzer

5.5.8.2. Optimize IOC Register Placement for Timing Logic Option

This option moves registers into I/O elements to meet t_{SU} or t_{CO} assignments, duplicating the register if necessary (as in the case in which a register fans out to multiple output locations). This option is turned on by default and is a global setting.

The **Optimize IOC Register Placement for Timing** logic option affects only pins that have a t_{SU} or t_{CO} requirement. Using the I/O register is possible only if the register directly feeds a pin or is fed directly by a pin. Therefore, this logic option does not affect registers with any of the following characteristics:

Note: To optimize registers with these characteristics, use other Quartus Prime Fitter optimizations.

- Have combinational logic between the register and the pin
- Are part of a carry chain
- Have an overriding location assignment
- Use the asynchronous load port and the value is not 1 (in device families where the port is available)

Related Information

Optimize IOC Register Placement for Timing Logic Option Help Topic
In *Quartus Prime Help*

5.5.8.3. Fast Input, Output, and Output Enable Registers

You can place individual registers in I/O cells manually by making fast I/O assignments with the Assignment Editor. By default, with correct timing assignments, the Fitter places the I/O registers in the correct I/O cell or in the core, to meet the performance requirement.

If the fast I/O setting is on, the register is always placed in the I/O element. If the fast I/O setting is off, the register is never placed in the I/O element. This is true even if the **Optimize IOC Register Placement for Timing** option is turned on. If there is

no fast I/O assignment, the Quartus Prime software determines whether to place registers in I/O elements if the **Optimize IOC Register Placement for Timing** option is turned on.

You can also use the four fast I/O options (**Fast Input Register**, **Fast Output Register**, **Fast Output Enable Register**, and **Fast OCT Register**) to override the location of a register that is in a Logic Lock region and force it into an I/O cell. If you apply this assignment to a register that feeds multiple pins, the Fitter duplicates the register and places it in all relevant I/O elements.

For more information about the **Fast Input Register** option, **Fast Output Register** option, **Fast Output Enable Register** option, and **Fast OCT (on-chip termination) Register** option, refer to Quartus Prime Help.

Related Information

- Fast Input Register logic option Help Topic
- Fast Output Register logic option
- Fast Output Enable Register logic option
- Fast OCT Register logic option

5.5.8.4. Programmable Delays

You can use various programmable delay options to minimize the t_{SU} and t_{CO} times. Programmable delays are advanced options that you use only after you compile a project, check the I/O timing, and determine that the timing is unsatisfactory.

The Quartus Prime software automatically adjusts the applicable programmable delays to help meet timing requirements. For detailed information about the effect of these options, refer to the device family handbook or data sheet.

After you have made a programmable delay assignment and compiled the design, you can view the implemented delay values for every delay chain and every I/O pin in the **Delay Chain Summary** section of the Compilation Report.

You can assign programmable delay options to supported nodes with the Assignment Editor. You can also view and modify the delay chain setting for the target device with the Chip Planner and Resource Property Editor. When you use the Resource Property Editor to make changes after performing a full compilation, recompiling the entire design is not necessary; you can save changes directly to the netlist. Because these changes are made directly to the netlist, the changes are not made again automatically when you recompile the design. The change management features allow you to reapply the changes on subsequent compilations.

Although the programmable delays in newer devices are user-controllable, Intel recommends their use for advanced users only. However, the Quartus Prime software might use the programmable delays internally during the Fitter phase.

For details about the programmable delay logic options available for Intel devices, refer to the following Quartus Prime Help topics:

Related Information

- Input Delay from Pin to Input Register logic option Help Topic
- Input Delay from Pin to Internal Cells logic option Help Topic
- Output Enable Pin Delay logic option Help Topic

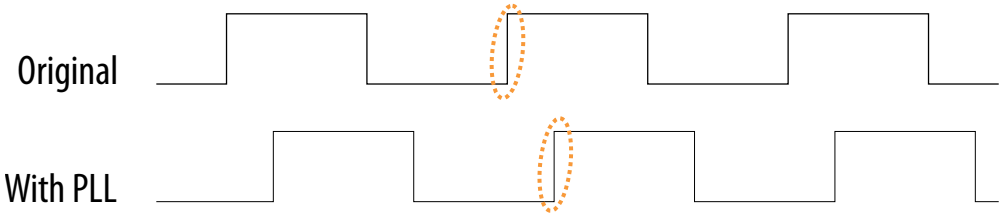
- Delay from Output Register to Output Pin logic option Help Topic
- Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations logic option Help Topic

In *Quartus Prime Help*

5.5.8.5. Use PLLs to Shift Clock Edges

Using a PLL typically improves I/O timing automatically. If the timing requirements are still not met, most devices allow the PLL output to be phase shifted to change the I/O timing. Shifting the clock backwards gives a better t_H at the expense of t_{SU}, while shifting it forward gives a better t_{SU} at the expense of t_H. You can use this technique only in devices that offer PLLs with the phase shift option.

Figure 65. Shift Clock Edges Forward to Improve t_{SU} at the Expense of t_H



You can achieve the same type of effect in certain devices by using the programmable delay called **Input Delay from Dual Purpose Clock Pin to Fan-Out Destinations**.

5.5.8.6. Use Fast Regional Clock Networks and Regional Clocks Networks

Regional clocks provide the lowest clock delay and skew for logic contained in a single quadrant. In general, fast regional clocks have less delay to I/O elements than regional and global clocks, and are used for high fan-out control signals. Placing clocks on these low-skew and low-delay clock nets provides better t_{CO} performance.

Intel devices have a variety of hierarchical clock structures. These include dedicated global clock networks, regional clock networks, fast regional clock networks, and periphery clock networks. The available resources differ between the various Intel device families.

For the number of clocking resources available in your target device, refer to the appropriate device handbook.

5.5.8.7. Spine Clock Limitations

In Arria 10 and Cyclone 10 GX designs with high clock routing demands, limitations in the Quartus Prime software can cause spine clock errors. These limits do not apply to Stratix 10 or Agilex 7 designs.

These errors can occur with designs using multiple memory interfaces and high-speed serial interface (HSSI) channels, especially with PMA Direct mode.

Global clock networks, regional clock networks, and periphery clock networks have an additional level of clock hierarchy known as spine clocks. Spine clocks drive the final row and column clocks to their registers; thus, the clock to every register in the chip is reached through spine clocks. Spine clocks are not directly user controllable.

To reduce these spine clock errors, constrain your design to use your regional clock resources better:

- If your design does not use Logic Lock regions, or if the Logic Lock regions are not aligned to your clock region boundaries, create additional Logic Lock regions and further constrain your logic.
- To ensure that the global promotion process uses the correct locations, assign specific pins to the I/Os using these periphery features.
- By default, some Intel FPGA IP functions apply a global signal assignment with a value of dual-regional clock. If you constrain your logic to a regional clock region and set the global signal assignment to **Regional** instead of **Dual-Regional**, you can reduce clock resource contention.

Related Information

Viewing Available Clock Networks in Chip Planner on page 149

5.5.9. Register-to-Register Timing Optimization Techniques

The next stage of design optimization seeks to improve register-to-register (f_{MAX}) timing. The following sections provide available options if the design does not meet timing requirements after compilation.

Coding style affects the performance of a design to a greater extent than other changes in settings. Always evaluate the code and make sure to use synchronous design practices.

Note: In the context of the Timing Analyzer, register-to-register timing optimization is the same as maximizing the slack on the clock domains in a design. The techniques in this section can improve the slack on different timing paths in the design.

Before performing design optimizations, understand the structure of the design as well as the effects of techniques in different types of logic. Techniques that do not benefit the logic structure can decrease performance.

Related Information

- Quartus Prime Pro Edition User Guide: Design Recommendations
- Design Assistant Rules List

5.5.9.1. Optimize Source Code

In many cases, optimizing the design's source code can have a very significant effect on your design performance. In fact, optimizing your source code is typically the most effective technique for improving the quality of your results and is often a better choice than using Logic Lock or location assignments.

You can use the Design Assistant to help identify areas in the design for timing optimization. Be aware of the number of logic levels needed to implement your logic while you are coding. Too many levels of logic between registers might result in critical paths failing timing. Try restructuring the design to use pipelining or more efficient coding techniques. Also, try limiting high fan-out signals in the source code. When possible, duplicate and pipeline control signals. Make sure the duplicate registers are protected by a preserve attribute, to avoid merging during synthesis.

If the critical path in your design involves memory or DSP functions, check whether you have code blocks in your design that describe memory or functions that are not being inferred and placed in dedicated logic. You might be able to modify your source code to cause these functions to be placed into high-performance dedicated memory or resources in the target device. When using RAM/DSP blocks, enable the optional input and output registers.

Ensure that your state machines are recognized as state machine logic and optimized appropriately in your synthesis tool. State machines that are recognized are generally optimized better than if the synthesis tool treats them as generic logic. In the Quartus Prime software, you can check the State Machine report under **Analysis & Synthesis** in the Compilation Report. This report provides details, including state encoding for each state machine that was recognized during compilation. If your state machine is not recognized, you might have to change your source code to enable it to be recognized.

Related Information

AN 584: Timing Closure Methodology for Advanced FPGA Designs

5.5.9.2. Improving Register-to-Register Timing

The choice of options and settings to improve the timing margin (slack) or to improve register-to-register timing depends on the failing paths in the design. To achieve the results that best approximate your performance requirements, apply the following techniques and compile the design after each step:

1. Ensure that your timing assignments are complete and correct. For details, refer to the *Initial Compilation: Required Settings* section in the *Design Optimization Overview* chapter.
2. Review all Design Assistant rule violations and other warning messages from your initial compilation and check for ignored timing assignments. Design Assistant helps to identify and correct any invalid timing constraints.
3. Apply netlist synthesis optimization options.
4. To optimize for speed, apply the following synthesis options:
   - Optimize Synthesis for Speed, Not Area
   - Flatten the Hierarchy During Synthesis
   - Set the Synthesis Effort to High
   - Prevent Shift Register Inference
   - Use Other Synthesis Options Available in Your Synthesis Tool
5. To optimize for performance, turn on Advanced Physical Optimization
6. Try different Fitter seeds. If only a small number of paths are failing by small negative slack, then you can try with a different seed to find a fit that meets constraints in the Fitter seed noise.

   Note: Omit this step if a large number of critical paths are failing, or if the paths are failing by a long margin.

7. To control placement, make Logic Lock assignments.
8. Modify your design source code to fix areas of the design that are still failing timing requirements by significant amounts.
9. Make location assignments, or as a last resort, perform manual placement by back-annotating the design.

   You can use Design Space Explorer II (DSE) to automate the process of running different compilations with different settings.

   If these techniques do not achieve performance requirements, additional design source code modifications might be required.

Related Information

- Optimize Settings with Design Space Explorer II on page 102
- Initial Compiler Settings on page 7

5.5.9.3. Physical Synthesis Optimizations

The Quartus Prime software offers physical synthesis optimizations that can help improve design performance regardless of the synthesis tool. You can apply physical synthesis optimizations both during synthesis and during fitting.

During the synthesis stage of the Quartus Prime compilation, physical synthesis optimizations operate either on the output from another EDA synthesis tool, or as an intermediate step in synthesis. These optimizations modify the synthesis netlist to improve either area or speed, depending on the technique and effort level you select.

To view and modify the synthesis netlist optimization options, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

If you use a third-party EDA synthesis tool and want to determine if the Quartus Prime software can remap the circuit to improve performance, use the **Perform WYSIWYG Primitive Resynthesis** option. This option directs the Quartus Prime software to un-map the LEs in an atom netlist to logic gates, and then map the gates back to Intel-specific primitives. Intel-specific primitives enable the Fitter to remap the circuits using architecture-specific techniques.

The Quartus Prime Compiler optimizes the design to achieve maximum speed performance, minimum area usage, or balances high performance and minimal logic usage, according to the setting of the **Optimization Technique** option. Set this option to **Speed** or **Balanced**.

During the Fitter stage of the Quartus Prime compilation, physical synthesis optimizations make placement-specific changes to the netlist that improve speed performance results for the specific Intel device.

Related Information

- Perform WYSIWYG Primitive Resynthesis Logic Option Help Topic
- Optimization Technique Logic Option Help Topic
  In *Quartus Prime Help*

5.5.9.4. Set Power Optimization During Synthesis to Normal Compilation

The default value for the Compiler's **Power Optimization During Synthesis** setting is **Normal Compilation**. However, if **Power Optimization During Synthesis** is set to **Extra Effort**, design performance can be affected. To avoid any possible effect, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Power Optimization During Synthesis** to confirm the **Normal Compilation** setting value.

Related Information

Power Optimization

5.5.9.5. Optimize Synthesis for Performance, Not Area

Design performance varies depending on coding style, synthesis tool used, and options you specify when synthesizing. Change your synthesis options if a large number of paths are failing, or if specific paths fail by a great margin and have many levels of logic.

Identify the default optimization targets of your Synthesis tool, and set your device and timing constraints accordingly. For example, if you do not specify a target frequency, some synthesis tools optimize for area.

Optimize for performance by specifying the **High Performance Effort**, **High Performance with Maximum Placement Effort**, **High Performance with Aggressive Power Effort**, **Superior Performance**, or **Superior Performance with Maximum Placement Effort** optimization mode.

Related Information

Optimization Technique Logic Option Help Topic
In *Quartus Prime Help*

5.5.9.6. Flatten the Hierarchy During Synthesis

Synthesis tools typically let you preserve hierarchical boundaries, which can be useful for verification or other purposes. However, the best optimization results generally occur when the synthesis tool optimizes across hierarchical boundaries, because doing so often allows the synthesis tool to perform the most logic minimization, which can improve performance. Whenever possible, flatten your design hierarchy to achieve the best results.

5.5.9.7. Set the Synthesis Effort to High

Synthesis tools offer varying synthesis effort levels to trade off compilation time with synthesis results. Set the synthesis effort to **high** to achieve best results when applicable.

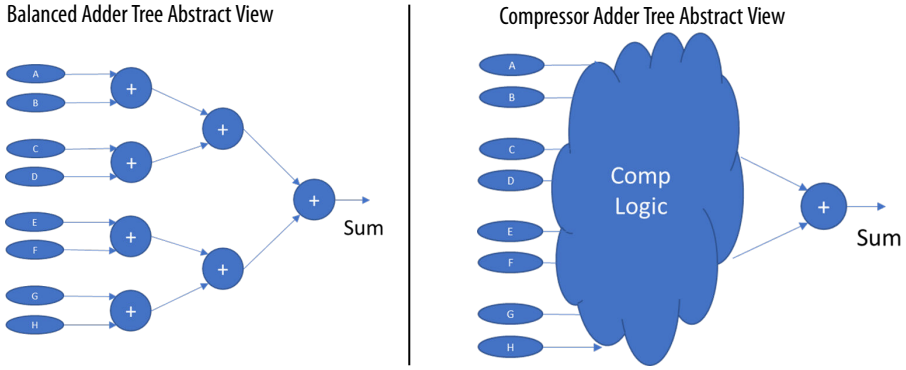
5.5.9.8. Change Adder Tree Styles

Structuring adder trees appropriately to match your targeted Intel FPGA device architecture and application can provide significant improvements in your design's efficiency and performance.

A good example of an application using a large adder tree is a finite impulse response (FIR) correlator. Using a pipelined binary or ternary adder tree appropriately can greatly improve the quality of your results for such applications.

Because ALMs can implement functions of up to six inputs, you can improve the performance of certain designs by using a compressor implementation for adder trees, rather than the default balanced binary tree implementation. The expected downside tradeoff of the compressor implementation is the use of more ALM logic resources. However, the overall logic depth is lower, and the final timing characteristics improve.

Figure 66. Balanced Binary Versus Compressor Style Adder Trees



For designs that may benefit, you can apply the **Use Compressor Implementation** (USE_COMPRESSOR_IMPLEMENTATION) global, entity, or instance assignment to specify whether the Compiler synthesizes adder trees as balanced binary trees, or as compressor style trees.

You can specify this assignment in the Assignment Editor, or with the following assignment in the `.qsf`.

```
set_instance_assignment -name USE_COMPRESSOR_IMPLEMENTATION ALWAYS -to <foo>
```

You can specify this assignment as either a global assignment, entity assignment, or instance assignment. You can alternatively use this assignment with `altera_attribute` to create instance assignments as well. For example:

```
(* altera_attribute = "-name USE_COMPRESSOR_IMPLEMENTATION ALWAYS" *) module
foo(a, b, c, o);
```

The following options are available for this assignment:

Table 34. Use Compressor Implementation Assignment Options

Option	Description
Always	The Compiler always synthesizes all adder trees with this assignment as compressor style trees. There is a limit of at least 2 non-constant operands before this triggers (otherwise synthesis implements a binary add or a pure-LUT implementation depending on size).
Never	The Compiler never synthesizes the assigned adder tree as a compressor. The Compiler synthesizes the adder as either a balanced binary tree, or if sufficiently small, in pure LUTs.
Auto	This setting currently behaves the same as the **Never** setting. The Compiler synthesizes the adder as either a balanced binary tree, or if sufficiently small, in pure LUTs. This setting never uses compressor style adder trees.

5.5.9.9. Duplicate Registers for Fan-Out Control

Often, timing failures can occur due to the influence of signals that are not directly involved in the failing transfers. This condition tends to manifest when off-critical nets, most commonly with a high fan-out, span a large distance and consequentially, warp the optimization of other paths around them.

Duplicating the sources of these types of globally-influential signals can help to disperse them across many hops, or even across many clock cycles, and focus more on local transfers.

For example, by duplicating a high fan-out signal in the form of a tree of registers, you can disperse the signal over several clock cycles. As the signal progresses down the tree, it progressively feeds more into local copies of the original registers, such that any individual register's destinations are well-localized and its influence on register optimization is minimal. The key to this optimization is to determine how to assign the original signal’s fan-outs among the duplicates. If any individual register requires driving a large distance, the benefit of the tree can be removed.

You can manually create a register tree and group the endpoints in the RTL by leveraging your system-level knowledge about how best to disperse the signal throughout your design, but it can be time consuming and have a widespread impact. For more information about manually creating a register tree, refer to Manual Register Duplication on page 122.

You can create register trees automatically in one of the following ways.

- Estimated Physical Proximity
- Hierarchical Proximity

Each method has its own methodology to determine the number of duplicates to create and how to assign the fan-outs between the duplicates.

5.5.9.9.1. Manual Register Duplication

Synthesis tools support options or attributes that specify the maximum fan-out of a register. When using Quartus Prime synthesis, you can set the **Maximum Fan-Out** logic option in the Assignment Editor to control the number of destinations for a node so that the fan-out count does not exceed a specified value. You can also use the `maxfan` attribute in your HDL code. The software duplicates the node as required to achieve the specified maximum fan-out.

Logic duplication using **Maximum Fan-Out** assignments normally increases resource utilization, and can potentially increase compilation time, depending on the placement and the total resource usage within the selected device.

The improvement in timing performance that results from **Maximum Fan-Out** assignments is design-specific. This is because when you use the **Maximum Fan-Out** assignment, the Fitter duplicates the source logic to limit the fan-out, but does not control the destinations that each of the duplicated sources drive. Therefore, it is possible for duplicated source logic to be driving logic located all around the device. To avoid this situation, you can use the **Manual Logic Duplication** logic option.

If you are using **Maximum Fan-Out** assignments, benchmark your design with and without these assignments to evaluate whether they give the expected improvement in timing performance. Use the assignments only when you get improved results.

You can manually duplicate registers in the Quartus Prime software regardless of the synthesis tool used. To duplicate a register, apply the **Manual Logic Duplication** logic option to the register with the Assignment Editor.

Note: Some Fitter optimizations may cause a small violation to the **Maximum Fan-Out** assignments to improve timing.

5.5.9.9.2. Automatic Register Duplication: Estimated Physical Proximity

The DUPLICATE_REGISTER assignment helps in leveraging estimated physical proximity information to guide the creation of duplicates and their fan-out assignments.

```
set_instance_assignment -name DUPLICATE_REGISTER -to <register_name>
<num_duplicates>
```

where,

- register_name is the register to duplicate. To create a register tree from a chain, create a unique assignment for each register in the chain. DUPLICATE_REGISTER assignments are processed in the appropriate order if they apply to registers that drive each other in a chain.
- num_duplicates is the number of duplicates of the register to create (including the original). If the original signal has M fan-out, the average fan-outs of the duplicates are M/N but any individual duplicate may have more or fewer, at the discretion of the algorithm.

The DUPLICATE_REGISTER assignment is processed during the Fitter stage. It is necessary to create the duplicates and assign fan-outs between the duplicates based on early estimates of physical proximity to maximize the amount of time spent optimizing the design post-duplication. However, this renders fine-grained assignment decisions imprecise. The DUPLICATE_REGISTER assignment is best used when the number of duplicates is small (under 100) and the groups created are coarse-grained enough to allow for flexibility during optimization after the duplicates are created.

The **Fitter Duplication Summary** panel of the Fit report details the DUPLICATE_REGISTER assignments picked up by Quartus Prime Pro Edition. It also summarizes any registered signal with greater than 1000 fan-outs, as they could be reasonable candidates for DUPLICATE_REGISTER assignments in future.

Important:

- Setting PHYSICAL_SYNTHESIS to OFF disables DUPLICATE_REGISTER.
- Unlike other physical synthesis optimizations, the DUPLICATE_REGISTER assignment does allow duplication of registers that feed asynchronous clears and registers having location assignments.
- The DUPLICATE_REGISTER assignment does not process registers that have any of the following conditions:
  - Registers drive global signals or clock signals.
  - Registers have timing assignments or exceptions applied to them.
  - Registers have a `preserve` attribute or a PRESERVE_REGISTER assignment.
  - Registers are marked as `don't touch`.
  - Registers drive or are driven by other partitions.

5.5.9.9.3. Automatic Register Duplication: Hierarchical Proximity

Leveraging design hierarchy information to guide the creation of duplicates and their fan-out assignments is enabled by the DUPLICATE_HIERARCHY_DEPTH assignment.

```
set_instance_assignment -name DUPLICATE_HIERARCHY_DEPTH -to <register_name>
<num_levels>
```

where,

- `register_name` is the last register in a chain that fans out to multiple hierarchies. To create a register tree, ensure that there are sufficient simple registers behind the node and those simple registers are automatically pulled into the tree.
- `num_levels` corresponds to the upper bound of the number of registers that exist in the chain to use for duplicating down the hierarchies.

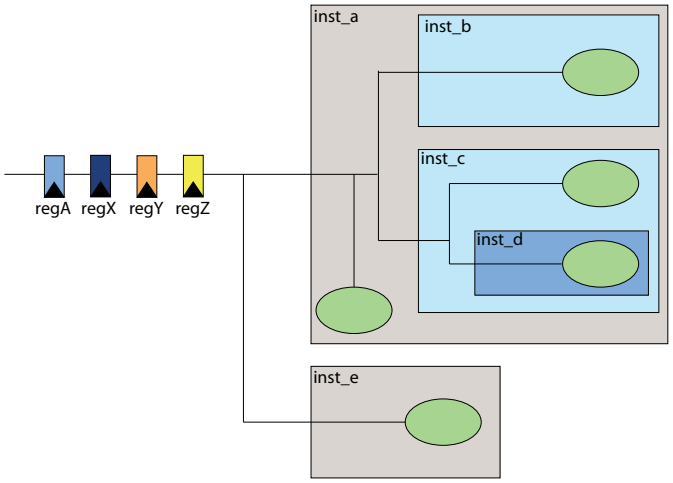
The DUPLICATE_HIERARCHY_DEPTH assignment is processed during the Synthesis stage. It is common for high-fanout signals to go through a pipeline of registers and drive into a sub-hierarchy of modules. For example, a system-wide reset can be propagated over several clock cycles and driven into many modules across the design. In several scenarios, it is useful to take advantage of the structure of this sub-hierarchy to infer the structure of the register tree to be created, such that endpoints within similar hierarchies are assigned the same copy of the signal, and branches in the design hierarchy dictates where to place branches in the register tree.

Important: The registers in the chain must satisfy all of the following conditions to be included in duplication:

- Registers must be fed only by another register.
- Registers must not be fed by a combinational logic.
- Registers must not be part of a synchronizer chain.
- Registers must not have any secondary signals.
- Registers must not have a `preserve` attribute or a PRESERVE_REGISTER assignment.
- All registers in the chain except the last one must have only one fan-out.

Consider the following example illustration of a netlist with a register chain and hierarchical organization of the endpoints it drives. The DUPLICATE_HIERARCHY_DEPTH assignment duplicates the pipeline registers across hierarchies, as shown in Registers Duplicated Across Hierarchies.

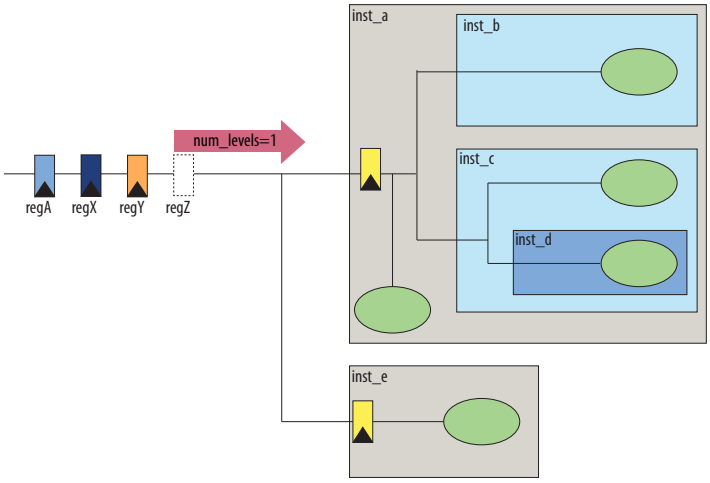
Figure 67. Original Diagram Showing Four Pipeline Registers Connected to Multiple Hierarchies



In this case, regZ is the appropriate assignment target as it is the endpoint in a chain of four registers. There is a maximum of three duplication candidates in this example (regZ, regY, and regX), so the assignment value can be anywhere between 1 and 3. regA is not pulled into the hierarchy to preserve the timing and optimization of paths that precede it. The DUPLICATE_HIERARCHY_DEPTH assignment is best used when a signal must be duplicated to more than 100 duplicates and the sub-hierarchy below the chain is deep and meaningful enough to guide the structure of the tree required.

Figure 68. Netlist After Duplicating regZ to Hierarchy Level One

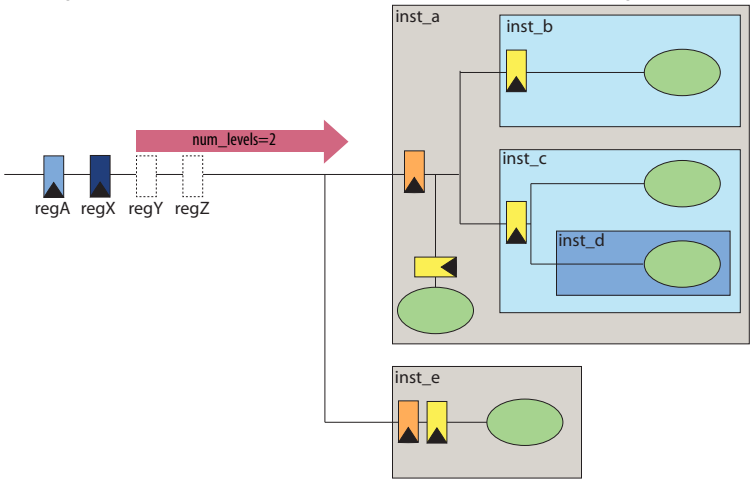
```
set_instance_assignment -name DUPLICATE_HIERARCHY_DEPTH -to regZ 1
```



When num_levels is set to 1, only regZ is pulled out of the chain and pushed down one hierarchy level into its fan-out tree.

Figure 69. Netlist After Duplicating regZ to Hierarchy Level Two

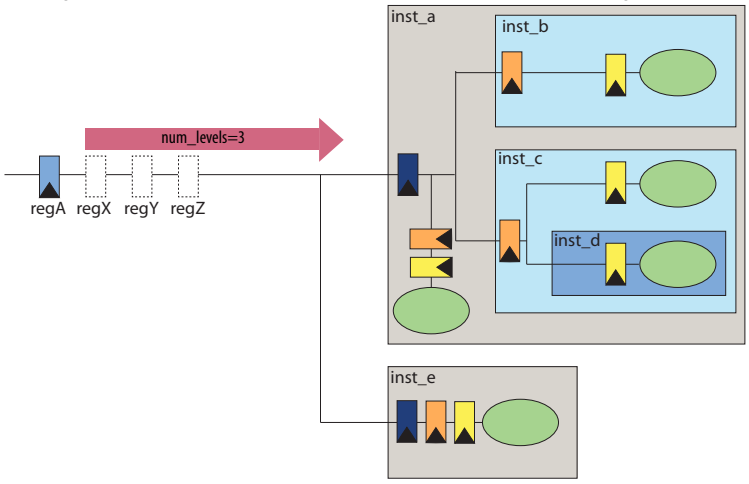
```
set_instance_assignment -name DUPLICATE_HIERARCHY_DEPTH -to regZ 2
```



When `num_levels` is set to 2, both `regY` and `regZ` are pulled out of the chain. `regZ` ends up at a maximum hierarchy depth two and `regY` ends up at hierarchy depth one.

Figure 70. Registers Duplicated Across Hierarchies

```
set_instance_assignment -name DUPLICATE_HIERARCHY_DEPTH -to regZ 3
```



When `num_levels` is set to 3, all three registers (`regZ`, `regY` and `regZ`) are pulled out of the chain and pushed to a maximum hierarchy depth of three, two, and one levels, respectively.

The **Hierarchical Tree Duplication Summary** panel in the Synthesis report provides information on the registers specified by the `DUPLICATE_HIERARCHY_DEPTH` assignment. It also includes a reason for the chain length that can be used as a starting point for further improvements with the assignment. The Synthesis report also provides a panel named **Hierarchical Tree Duplication Details**, which provides information about the individual registers in the chain that can be used to better understand the structure of the implemented duplicates.

Related Information

Synchronous Reset Design Strategies, AN 917 Reset Design Techniques for Intel Hyperflex Architecture FPGAs

5.5.9.10. Prevent Shift Register Inference

Turning off the inference of shift registers can increase performance. This setting forces the software to use logic cells to implement the shift register, instead of using the ALTSHIFT_TAPS IP core to implement the registers in memory block. If you implement shift registers in logic cells instead of memory, logic utilization increases.

5.5.9.11. Use Other Synthesis Options Available in Your Synthesis Tool

With your synthesis tool, experiment with the following options if they are available:

- Turn on register balancing or retiming
- Turn on register pipelining
- Turn off resource sharing

These options can increase performance, but typically increase the resource utilization of your design.

5.5.9.12. Fitter Seed

The Fitter seed affects the initial placement configuration of the design. Any change in the initial conditions changes the Fitter results; accordingly, each seed value results in a somewhat different fit. You can experiment with different seeds to attempt to obtain better fitting results and timing performance.

Changes in the design impact performance between compilations. This random variation is inherent in placement and routing algorithms—it is impossible to try all seeds and get the absolute best result.

Note: Any design change that directly or indirectly affects the Fitter has the same type of random effect as changing the seed value. This includes any change in source files, **Compiler Settings** or **Timing Analyzer Settings**. The same effect can appear if you use a different computer processor type or different operating system, because different systems can change the way floating point numbers are calculated in the Fitter.

If a change in optimization settings marginally affects the register-to-register timing or number of failing paths, you cannot always be certain that your change caused the improvement or degradation, or whether it is due to random effects in the Fitter. If your design is still changing, running a seed sweep (compiling your design with multiple seeds) determines whether the average result improved after an optimization change, and whether a setting that increases compilation time has benefits worth the increased time, such as with physical synthesis settings. The sweep also shows the amount of random variation to expect for your design.

If your design is finalized you can compile your design with different seeds to obtain one optimal result. However, if you subsequently make any changes to your design, you might need to perform seed sweep again.

Click **Assignments ➤ Compiler Settings** to control the initial placement with the seed. You can use the DSE II to perform a seed sweep easily.

To specify a Fitter seed use the following Tcl command :

```
set_global_assignment -name SEED <value>
```

Related Information

Optimize Settings with Design Space Explorer II on page 102

5.5.9.13. Set Maximum Router Timing Optimization Level

To improve routability in designs where the router did not pick up the optimal routing lines, set the **Router Timing Optimization Level** to **Maximum**. This setting determines how aggressively the router tries to meet the timing requirements. Setting this option to **Maximum** can marginally increase design speed at the cost of increased compilation time. Setting this option to **Minimum** can reduce compilation time at the cost of marginally reduced design speed. The default value is **Normal**.

Related Information

Router Timing Optimization Level Logic Option
In *Quartus Prime Help*

5.5.9.14. Register-to-Register Timing Analysis

Your design meets timing requirements when you do not have negative slack on any register-to-register path on any of the clock domains. When timing requirements are not met, a report on the failed paths can uncover more detail.

5.5.9.14.1. Tips for Analyzing Failing Paths

When you are analyzing failing paths, examine the reports and waveforms to determine if the correct constraints are being applied, and add timing exceptions as appropriate. A multicycle constraint relaxes setup or hold relationships by the specified number of clock cycles. A false path constraint specifies paths that can be ignored during timing analysis. Both constraints allow the Fitter to work harder on affected paths.

- Focus on improving the paths that show the worst slack. The Fitter works hardest on paths with the worst slack. If you fix these paths, the Fitter might be able to improve the other failing timing paths in the design.
- Check for nodes that appear in many failing paths. These nodes are at the top of the list in a timing report panel, along with their minimum slacks. Look for paths that have common source registers, destination registers, or common intermediate combinational nodes. In some cases, the registers are not identical, but are part of the same bus.
- In the timing analysis report panels, click the **From** or **To** column headings to sort the paths by source or destination registers. If you see common nodes, these nodes indicate areas of your design that might be improved through source code changes or Quartus Prime optimization settings. Constraining the placement for just one of the paths might decrease the timing performance for other paths by moving the common node further away in the device.

Related Information

- Exploring Paths in the Chip Planner on page 160
- Design Evaluation for Timing Closure on page 67

- Review Timing Path Details on page 77

5.5.9.14.2. Tips for Analyzing Failing Clock Paths that Cross Clock Domains

When analyzing clock path failures:

- Check whether these paths cross two clock domains. In paths that cross two clock domains, the **From Clock** and **To Clock** in the timing analysis report are different.

Figure 71. Different Value in From Clock and To Clock Field

Setup Transfers

	From Clock	To Clock	RR Paths	FR Paths	RF Paths	FF Paths
1	clkin	clkin	21	0	0	0
2	clkin	clkout	false path	0	0	0
3	clkout	clkout	31	0	0	0

- Check if the design contains paths that involve a different clock in the middle of the path, even if the source and destination register clock are the same.
- Check whether failing paths between these clock domains need to be analyzed synchronously. Set failing paths that are not to be analyzed synchronously as false paths.
- When you run `report_timing` on a design, the report shows the launch clock and latch clock for each failing path. Check whether the relationship between the launch clock and latch clock is realistic and what you expect from your knowledge of the design. For example, the path can start at a rising edge and end at a falling edge, which reduces the setup relationship by one half clock cycle.
- Review the clock skew that appears in the Timing Report. A large skew may indicate a problem in the design, such as a gated clock, or a problem in the physical layout (for example, a clock using local routing instead of dedicated clock routing). When you have made sure the paths are analyzed synchronously and that there is no large skew on the path, and that the constraints are correct, you can analyze the data path. These steps help you fine tune your constraints for paths across clock domains to ensure you get an accurate timing report.
- Check if the PLL phase shift is reducing the setup requirement. You might adjust this by using PLL parameters and settings.
- Ignore paths that cross clock domains for logic protected with synchronization logic (for example, FIFOs or double-data synchronization registers), even if the clocks are related. Alternatively, specify the `set_clock_groups -exclusive` setting between unrelated clocks
- Set false path constraints on all unnecessary paths. Attempting to optimize unnecessary paths can prevent the Fitter from meeting the timing requirements on timing paths that are critical to the design.

Related Information

Report CDC Viewer on page 93

5.5.9.14.3. Tips for Critical Path Analysis

When analyzing the failing paths in a design, it is helpful to understand the interactions around the critical paths.

To understand what may be pulling on a critical path, the following `report_timing` command can be useful.

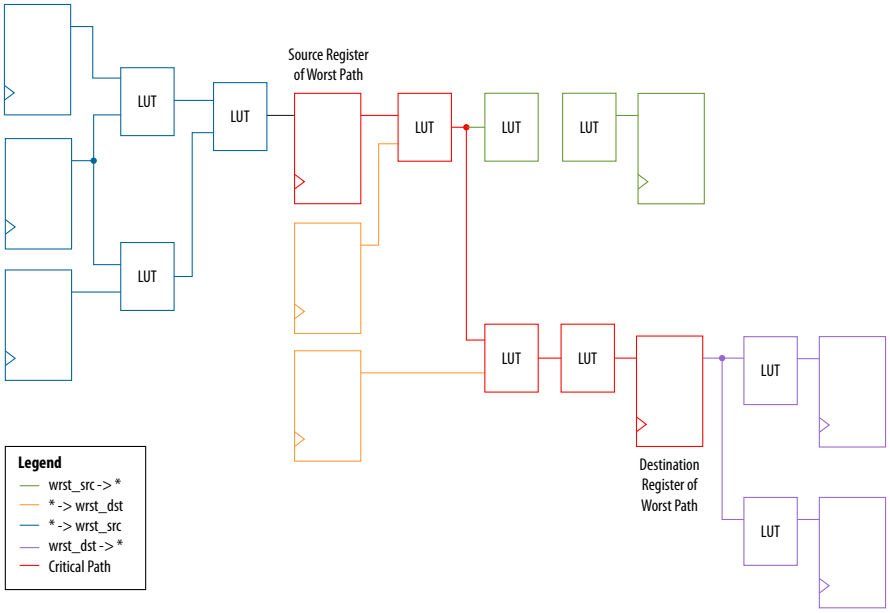
1. In the project directory, run the `report_timing` command to find the nodes in a critical path.
2. Copy the code below in a `.tcl` file, and replace the first two variable with the node names from the **From Node** and **To Node** columns of the worst path. The script analyzes the path between the worst source and destination registers.

```
set wrst_src <insert_source_of_worst_path_here>
set wrst_dst <insert_destination_of_worst_path_here>
report_timing -setup -npaths 50 -detail path_only -from $wrst_src \
-panel_name "Worst Path||wrst_src -> *"
report_timing -setup -npaths 50 -detail path_only -to $wrst_dst \
-panel_name "Worst Path||* -> wrst_dst"
report_timing -setup -npaths 50 -detail path_only -to $wrst_src \
-panel_name "Worst Path||* -> wrst_src"
report_timing -setup -npaths 50 -detail path_only -from $wrst_dst \
-panel_name "Worst Path||wrst_dst -> *"
```

3. From the **Script** menu, source the `.tcl` file.
4. In the resulting timing panel, locate timing failed paths (highlighted in red) in the Chip Planner, and view information such as distance between the nodes and large fan-outs.

   The figure shows a simplified example of what these reports analyzed.

Figure 72. Timing Report



The critical path of the design is in red. The relation between the `.tcl` script and the figure is:

- The first two lines show everything inside the two endpoints of the critical path that are pulling them in different directions.
  - The first `report_timing` command analyzes all paths the source is driving, shown in green.
  - The second `report_timing` command analyzes all paths going to the destination, including the critical path, shown in orange.
- The last two `report_timing` commands show everything outside of the endpoints pulling them in other directions.

If any of these neighboring paths have slacks near the critical path, the Fitter is balancing these paths with the critical path, trying to achieve the best slack.

Related Information

Review Timing Path Details on page 77

5.5.9.14.4. Tips for Creating a .tcl Script to Monitor Critical Paths Across Compiles

Many designs have the same critical paths show up after each compile. In other designs, critical paths bounce around between different hierarchies, changing with each compile.

This behavior happens in high speed designs where many register-to-register paths have very little slack. Different placements can then result in timing failures in the marginal paths.

1. In the project directory, create a script named `TQ_critical_paths.tcl`.
2. After compilation, review the critical paths and then write a generic `report_timing` command to capture those paths.

   For example, if several paths fail in a low-level hierarchy, add a command such as:

```
report_timing –setup –npaths 50 –detail path_only \
   –to “main_system: main_system_inst|app_cpu:cpu|*” \
   –panel_name “Critical Paths||s: * -> app_cpu”
```

3. If there is a specific path, such as a bit of a state-machine going to other `*count_sync*` registers, you can add a command similar to:

```
report_timing –setup –npaths 50 –detail path_only \
   –from “main_system: main_system_inst|egress_count_sm:egress_inst|update” \
   –to “*count_sync*” –panel_name “Critical Paths||s: egress_sm|update ->
count_sync”
```

4. Execute this script in the Timing Analyzer after every compilation, and add new `report_timing` commands as new critical paths appear.

   This helps you monitor paths that consistently fail and paths that are only marginal, so you can prioritize effectively

5.5.9.14.5. Global Routing Resources

Global routing resources are designed to distribute high fan-out, low-skew signals (such as clocks) without consuming regular routing resources. Depending on the device, these resources can span the entire chip or a smaller portion, such as a

quadrant. The Quartus Prime software attempts to assign signals to global routing resources automatically, but you might be able to make more suitable assignments manually.

For details about the number and types of global routing resources available, refer to the relevant device handbook.

Check the global signal utilization in your design to ensure that the appropriate signals have been placed on the global routing resources. In the Compilation Report, open the Fitter report and click **Resource Section**. Analyze the Global & Other Fast Signals and Non-Global High Fan-out Signals reports to determine whether any changes are required.

You might be able to reduce skew for high fan-out signals by placing them on global routing resources. Conversely, you can reduce the insertion delay of low fan-out signals by removing them from global routing resources. Doing so can improve clock enable timing and control signal recovery/removal timing, but increases clock skew. Use the **Global Signal** setting in the Assignment Editor to control global routing resources.

5.5.9.14.6. Register RAMS and DSPs

If your design includes long timing paths going to and from RAMs and DSPs, you must fully register the RAMs and DSPs.

RAM and DSP performance can vary, depending on the memory mode. A memory using read-during-write mode is slower than a memory that uses a different mode. Refer to your FPGA device documentation for hardware performance specifications. If the f_{MAX} is restricted due to mode, change to a different memory mode with a higher performance specification, if possible.

5.5.10. Metastability Analysis and Optimization Techniques

Metastability problems can occur when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the designer cannot guarantee that the signal meets its setup and hold time requirements. The mean time between failures (MTBF) is an estimate of the average time between instances when metastability could cause a design failure.

You can use the Quartus Prime software to analyze the average MTBF due to metastability when a design synchronizes asynchronous signals and to optimize the design to improve the MTBF. These metastability features are supported only for designs constrained with the Timing Analyzer, and for select device families.

Synchronization identification can affect retiming. Registers that the Compiler identifies as being part of a synchronizer are not retimed. The default chain length is 3, but in some cases, a synchronizer chain is not necessary and should not be inferred. Use the report_metastability command to identify synchronizer chains that you can reduce.

For example, consider a bus that uses a synchronized enable when crossing clock domains. If you pipeline such a bus, the pipeline stages can be considered as part of a synchronizer chain, and are not used to retime the paths. Setting the chain length to 1 for these paths allows the pipeline registers to be used for retiming.

Related Information

Quartus Prime Pro Edition User Guide: Design Recommendations

5.6. Periphery to Core Register Placement and Routing Optimization

The Periphery to Core Register Placement and Routing Optimization (P2C) option specifies whether the Fitter performs targeted placement and routing optimization on direct connections between periphery logic and registers in the FPGA core. P2C is an optional pre-routing-aware placement optimization stage that enables you to more reliably achieve timing closure.

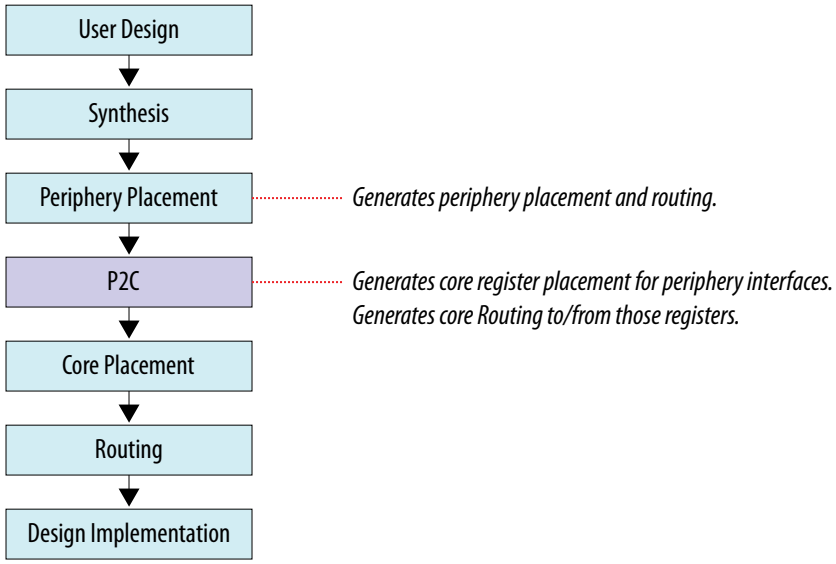
Note: The **Periphery to Core Register Placement and Routing Optimization** option applies in both directions, periphery to core and core to periphery.

Transfers between external interfaces (for example, high-speed I/O or serial interfaces) and the FPGA often require routing many connections with tight setup and hold timing requirements. When this option is turned on, the Fitter performs P2C placement and routing decisions before those for core placement and routing. This reserves the necessary resources to ensure that your design achieves its timing requirements and avoids routing congestion for transfers with external interfaces.

This option is available as a global assignment, or can be applied to specific instances within your design.

Figure 73. Periphery to Core Register Placement and Routing Optimization (P2C) Flow

P2C runs after periphery placement, and generates placement for core registers on corresponding P2C/C2P paths, and core routing to and from these core registers.



Setting Periphery to Core Optimizations in the Advanced Fitter Setting Dialog Box on page 134

Setting Periphery to Core Optimizations in the Assignment Editor on page 134

Viewing Periphery to Core Optimizations in the Fitter Report on page 135

5.6.1. Setting Periphery to Core Optimizations in the Advanced Fitter Setting Dialog Box

The **Periphery to Core Placement and Routing Optimization** setting specifies whether the Fitter optimizes targeted placement and routing on direct connections between periphery logic and registers in the FPGA core.

You can optionally perform periphery to core optimizations by instance with settings in the Assignment Editor.

1. In the Quartus Prime software, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.
2. In the **Advanced Fitter Settings** dialog box, for the **Periphery to Core Placement and Routing Optimization** option, select one of the following options depending on how you want to direct periphery to core optimizations in your design:
   a. Select **Auto** to direct the software to automatically identify transfers with tight timing windows, place the core registers, and route all connections to or from the periphery.
   b. Select **On** to direct the software to globally optimize all transfers between the periphery and core registers, regardless of timing requirements.

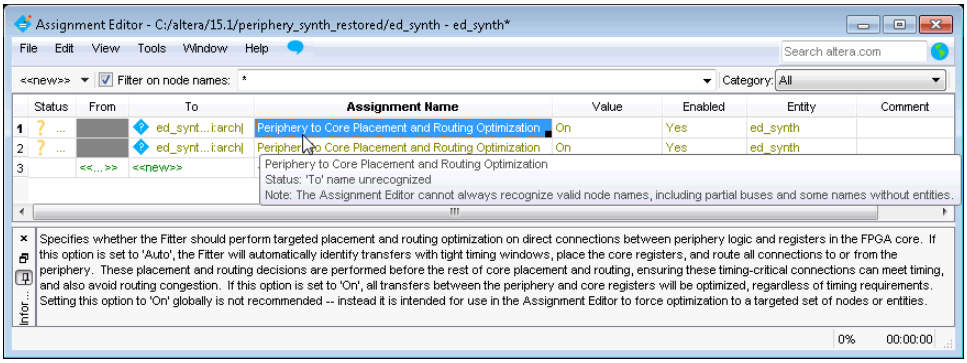
      Note: Setting this option to **On** in the **Advanced Fitter Settings** is not recommended. The intended use for this setting is in the Assignment Editor to force optimization for a targeted set of nodes or instance.
   c. Select **Off** to disable periphery to core path optimization in your design.

5.6.2. Setting Periphery to Core Optimizations in the Assignment Editor

When you turn on the **Periphery to Core Placement and Routing Optimization** (P2C/C2P) setting in the Assignment Editor, the Quartus Prime software performs periphery to core, or core to periphery optimizations on selected instances in your design.

You can optionally perform periphery to core optimizations by instance with settings in the **Advanced Fitter Settings** dialog box.

1. In the Quartus Prime software, click **Assignments ➤ Assignment Editor**.
2. For the selected path, double-click the **Assignment Name** column, and then click the **Periphery to core register placement and routing optimization** option in the drop-down list.
3. In the **To** column, choose either a periphery node or core register node on a P2C/C2P path you want to optimize. Leave the **From** column empty.
   For paths to appear in the Assignments Editor, you must first run Analysis & Synthesis on your design.



5.6.3. Viewing Periphery to Core Optimizations in the Fitter Report

The Quartus Prime software generates a periphery to core placement and routing optimization summary in the **Fitter (Place & Route)** report after compilation.

1. Compile your Quartus Prime project.
2. In the **Tasks** pane, select **Compilation**.
3. Under **Fitter (Place & Route)**, double-click **View Report**.
4. In the **Fitter** folder, expand the **Place Stage** folder.
5. Double-click **Periphery to Core Transfer Optimization Summary**.

Table 35. Fitter Report - Periphery to Core Transfer Optimization (P2C) Summary

From Path	To Path	Status
Node 1	Node 2	**Placed and Routed**—Core register is locked. Periphery to core/core to periphery routing is committed.
Node 3	Node 4	**Placed but not Routed**—Core register is locked. Routing is not committed. This occurs when P2C is not able to optimize all targeted paths within a single group, for example, the same delay/wire requirement, or the same control signals. Partial P2C routing commitments may cause unresolvable routing congestion.
Node 5	Node 6	**Not Optimized**—This occurs when P2C is set to **Auto** and the path is not optimized due to one of the following issues: a. The delay requirement is impossible to achieve. b. The minimum delay requirement (for hold timing) is too large. The P2C algorithm cannot efficiently handle cases when many wires need to be added to meet hold timing. c. P2C encountered unresolvable routing congestion for this particular path.

Periphery to Core Transfer Optimization Summary

	From	To	Status
1	Periphery to Core Transfer		
2	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[1].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|single_port.afi_rdata_valid_regs|sr_out[0]	Placed but Not Routed
3	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[2].lane_gen[2].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[63]	Placed but Not Routed
4	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[1].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[11]	Placed but Not Routed
5	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[7]	Placed but Not Routed
6	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[3]	Placed but Not Routed
7	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[0]	Placed but Not Routed
8	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[79]	Placed but Not Routed
9	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[75]	Placed but Not Routed
10	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[0].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[73]	Placed but Not Routed
11	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[2].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[71]	Placed but Not Routed
12	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[2].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[69]	Placed but Not Routed
13	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[1].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[35]	Placed but Not Routed
14	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[31]	Placed but Not Routed
15	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[3].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[27]	Placed but Not Routed
16	dut|arch|arch_inst|io_tiles_wrap_inst|io_tiles_inst|tile_gen[0].lane_gen[2].lane_inst	dut|arch|arch_inst|afi.if_inst|mem_sp_bidir_data.mem_dq.afi_regs_i|sr_out[23]	Placed but Not Routed

5.7. Scripting Support

You can run procedures and make settings described in this manual in a Tcl script. You can also run procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus Prime command-line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp
```

You can specify many of the options described in this section either in an instance, or at a global level, or both.

Use the following Tcl command to make a global assignment:

```
set_global_assignment -name <.qsf variable name> <value>
```

Use the following Tcl command to make an instance assignment:

```
set_instance_assignment -name <.qsf variable name> <value> -to <instance name>
```

Note: If the <*value*> field includes spaces (for example, 'Standard Fit'), you must enclose the value in straight double quotation marks.

Related Information

- Quartus Prime Pro Edition Settings File Reference Manual
- Quartus Prime Pro Edition User Guide: Scripting
- Quartus Prime Pro Edition User Guide: Scripting

5.7.1. Initial Compilation Settings

Use the Quartus Prime Settings File (`.qsf`) variable name in the Tcl assignment to make the setting along with the appropriate value. The **Type** column indicates whether the setting is supported as a global setting, an instance setting, or both.

The top table lists the `.qsf` variable name and applicable values for the settings described in the *Initial Compilation: Required Settings* section in the *Design Optimization Overview* chapter. The bottom table lists the advanced compilation settings.

Table 36. Initial Compilation Settings

Setting Name	.qsf File Variable Name	Values	Type
Optimize IOC Register Placement For Timing	OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING	ON, OFF	Global
Optimize Hold Timing	OPTIMIZE_HOLD_TIMING	OFF, IO PATHS AND MINIMUM TPD PATHS, ALL PATHS	Global

Table 37. Advanced Compilation Settings

Setting Name	.qsf File Variable Name	Values	Type
Router Timing Optimization level	ROUTER_TIMING_OPTIMIZATION_LEVEL	NORMAL, MINIMUM, MAXIMUM	Global

Related Information

Design Optimization Overview on page 6

5.7.2. I/O Timing Optimization Techniques

The table lists the `.qsf` file variable name and applicable values for the I/O timing optimization settings.

Table 38. I/O Timing Optimization Settings

Setting Name	.qsf File Variable Name	Values	Type
Optimize IOC Register Placement For Timing	OPTIMIZE_IOC_REGISTER_PLACEMENT_FOR_TIMING	ON, OFF	Global
Fast Input Register	FAST_INPUT_REGISTER	ON, OFF	Instance
Fast Output Register	FAST_OUTPUT_REGISTER	ON, OFF	Instance
Fast Output Enable Register	FAST_OUTPUT_ENABLE_REGISTER	ON, OFF	Instance
Fast OCT Register	FAST_OCT_REGISTER	ON, OFF	Instance

5.7.3. Register-to-Register Timing Optimization Techniques

The table lists the `.qsf` file variable name and applicable values for the settings described in *Register-to-Register Timing Optimization Techniques*.

Table 39. Register-to-Register Timing Optimization Settings

Setting Name	.qsf File Variable Name	Values	Type
Perform WYSIWYG Primitive Resynthesis	ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP	ON, OFF	Global, Instance
Fitter Seed	SEED	*<integer>*	Global
Maximum Fan-Out	MAX_FANOUT	*<integer>*	Instance
Manual Logic Duplication	DUPLICATE_ATOM	*<node name>*	Instance
Optimize Power during Synthesis	OPTIMIZE_POWER_DURING_SYNTHESIS	NORMAL, OFF EXTRA_EFFORT	Global
Optimize Power during Fitting	OPTIMIZE_POWER_DURING_FITTING	NORMAL, OFF EXTRA_EFFORT	Global

5.8. Timing Closure and Optimization Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Revised *Aggregating and Comparing Compilation Results with Exploration Dashboard* and added links to *AN 1006: Multi-Project Analysis with Exploration Dashboard*. • Added link to *AN 917: Reset Design Techniques for Intel Hyperflex Architecture FPGAs* to *Synchronous Reset Design Strategies* topic. • Updated *Starting the Exploration Dashboard* for new GUI and added links to *AN 1006: Multi-Project Analysis with Exploration Dashboard*.
2023.10.02	23.3	• Updated version support information and introduction of the Exploration Dashboard GUI (`quartus_edw`) in *Aggregating and Comparing Compilation Results with Exploration Dashboard* topic. • Revised command syntax in *Base Exploration Dashboard Properties* topic. • Added new property names to *Project Handle Properties* topic. • Updated *Starting the Exploration Dashboard* for the Exploration Dashboard GUI (`quartus_edw`) and options.
2023.08.01	23.2	• Updated version support information in *Aggregating and Comparing Compilation Results with Exploration Dashboard* topic. • Corrected graphic size in *Use PLLs to Shift Clock Edges*.
2023.06.26	23.2	• Revised *Base Exploration Dashboard Properties* topic to describe user_data property.
2023.04.03	23.1	• Added new *Aggregating and Comparing Compilation Results with Exploration Dashboard* section. • Added new *Change Adder Tree Styles* topic describing new USE_COMPRESSOR_IMPLEMENTATION assignment. • Updated product family name to "Intel Agilex 7."
2022.01.07	21.4	• Clarified device applicability in *Spine Clock Limitations* topic. • Added Design Assistant information to *Review Timing Constraints* topic. • Added Design Assistant information to *Review Timing Constraints* topic. • Removed references to obsolete Advisors throughout. • Added Design Assistant information to *Optimize Source Code* topic. • Added Design Assistant information to *Improving Register-to-Register Timing* topic. • Added Design Assistant information to *Optimize Synthesis for Performance, Not Area* topic. • Added set_clock_groups -exclusive setting information to *Tips for Analyzing Failing Clock Paths that Cross Clock Domains* topic. • Added new *Register RAMS and DSPs* topic. • Revised *Metastability Analysis and Optimization Techniques* topic for synchronizers.
2021.10.04	21.3	• Updated name of Report Hierarchical Retiming Restrictions command and report to Report Retiming Restrictions.
2021.06.21	21.2	• Added note about variables that can cause differences in the compilation results between seed sweeps with DSE II.
2020.09.28	20.3	• Added "Back-Annotate Optimized Assignments" topic to describe new GUI support for back-annotation of pin, RAM, DSP, and clock assignments. • Added "Correct Design Assistant Rule Violations" topic. • Updated "Report Timing" topic for Extra Info tab data.

continued...

Document Version	Quartus Prime Version	Changes
		• Added new "Report Logic Depth," "Report Neighbor Paths, "Report Register Spread," "Report Route Net of Interest," "Report Hierarchical Retiming Restrictions," and "Report Pipelining Information," topics to "Review Details of Timing Paths" section. • Moved "Optimize Settings with Design Space Explorer II" to "Design Evaluation for Timing Closure" section and updated links to Help. • Retitled "Intel Stratix 10 Timing Closure Recommendations" topic to "Implement Fast Forward Timing Closure Recommendations".
2019.07.01	19.1	Added important notes to *Automatic Register Duplication: Estimated Physical Proximity* and *Automatic Register Duplication: Hierarchical Proximity* topics.
2019.04.01	19.1	• Added more information about register duplication methods in *Duplicate Logic for Fan-out Control* topic. • Moved content related to manual register duplication from *Duplicate Logic for Fan-out Control* topic to a newly created sub-topic *Manually Adding Duplicate Registers*. • Added *Automatic Register Duplication: Estimated Physical Proximity* and *Automatic Register Duplication: Hierarchical Proximity* as new sub-topics under *Duplicate Logic for Fan-out Control* to describe automatic register duplication process.
2018.11.12	18.1.0	• Updated "Placement Effort Multiplier" figure and text descriptions in "Adjust Placement Effort" topic. • Updated "Fitter Effort" figure and text descriptions in "Adjust Fitter Effort" topic. • Updated "Optimize Hold Timing Option" screenshot in "Wires Added for Hold" topic.
2018.09.24	18.1.0	• Removed duplicated topic: *Resource Utilization Optimization Techniques*. The topic is now in the *Area Optimization* chapter. • Removed reference to unsupported CARRY and CASCADE buffers from "Optimize IOC Register Placement for Timing Logic Option" topic.
2017.11.06	17.1.0	• Added support for Stratix 10 Hyper-Retiming, Fast Forward compilation, and Fast Forward Viewer. — Added topics: Critical Chains, Viewing Critical Chains, Intel Stratix 10 Timing Closure Recommendations, Retiming Limit Details Report, Using the Retiming Limit Details Report, Fast Forward Timing Closure Recommendations, Generating Fast Forward Timing Closure Recommendations, Implementing Fast Forward Recommendations. • Added topic about using partitions to achieve timing closure. • Moved Topic: Design Evaluation for Timing Closure after Initial Compilation: Optional Fitter Settings. • Removed statement about applying physical synthesis optimizations in a portion of a design. • Removed references to optimizing hold timing for selected paths. • Updated logic options about resource utilization optimization settings.
2017.05.08	17.0.0	• Added topic: *Critical Paths*. • Updated *Register-to-Register Timing* and renamed to *Register-to-Register Timing Analysis*. • Renamed topic: *Timing Analysis with the Timing Analyzer* to *Displaying Path Reports with the Timing Analyzer*. • Removed (LUT-Based Devices) remark from topic titles. • Renamed topic: *Optimizing Timing (LUT-Based Devices)* to *Timing Optimization*. • Renamed topic: *Debugging Timing Failures in the Timing Analyzer* to *Displaying Timing Closure Recommendations for Failing Paths*. • Renamed topic: *Improving Register-to-Register Timing Summary* to *Improving Register-to-Register Timing* .

continued...

Document Version	Quartus Prime Version	Changes
		• Removed topics: *Tips for Locating Multiple Paths to the Chip Planner*, *LogicLock Assignments* and *Hierarchy Assignments*, . • Removed reference to deprecated Fitter Effort Logic Option. • Removed information about Pin Advisor and Resource Optimization Advisor. • Removed figure: Clock Regions
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.02	16.0.0	• Removed information about deprecated physical synthesis options. • Added information about monitoring clustering difficulty.
2015.11.02	15.1.0	• Added: *Periphery to Core Register Placement and Routing Optimization*. • Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Updated DSE II content.
June 2014	14.0.0	• Dita conversion. • Removed content about obsolete devices that are no longer supported in QII software v14.0: Arria GX, Arria II, Cyclone III, Stratix II, Stratix III. • Replaced Megafunction content with IP core content.
November 2013	13.1.0	• Added Design Evaluation for Timing Closure section. • Removed Optimizing Timing (Macrocell-Based CPLDs) section. • Updated Optimize Multi-Corner Timing and Fitter Aggressive Routability Optimization. • Updated Timing Analysis with the Timing Analyzer to show how to access the **Report All Summaries** command. • Updated Ignored Timing Constraints to include a help link to *Fitter Summary Reports* with the **Ignored Assignment Report** information.
May 2013	13.0.0	• Renamed chapter title from Area and Timing Optimization to Timing Closure and Optimization. • Removed design and area/resources optimization information. • Added the following sections: Fitter Aggressive Routability Optimization. Tips for Analyzing Paths from/to the Source and Destination of Critical Path. Tips for Locating Multiple Paths to the Chip Planner. Tips for Creating a .tcl Script to Monitor Critical Paths Across Compiles.
November 2012	12.1.0	• Updated "Initial Compilation: Optional Fitter Settings" on page 13–2, "I/O Assignments" on page 13–2, "Initial Compilation: Optional Fitter Settings" on page 13–2, "Resource Utilization" on page 13–9, "Routing" on page 13–21, and "Resolving Resource Utilization Problems" on page 13–43.
June 2012	12.0.0	• Updated "Optimize Multi-Corner Timing" on page 13–6, "Resource Utilization" on page 13–10, "Timing Analysis with the Timing Analyzer" on page 13–12, "Using the Resource Optimization Advisor" on page 13–15, "Increase Placement Effort Multiplier" on page 13–22, "Increase Router Effort Multiplier" on page 13–22 and "Debugging Timing Failures in the Timing Analyzer" on page 13–24. • Minor text edits throughout the chapter.

continued...

Document Version	Quartus Prime Version	Changes
November 2011	11.1.0	• Updated the "Timing Requirement Settings", "Standard Fit", "Fast Fit", "Optimize Multi-Corner Timing", "Timing Analysis with the Timing Analyzer", "Debugging Timing Failures in the Timing Analyzer", "LogicLock Assignments", "Tips for Analyzing Failing Clock Paths that Cross Clock Domains", "Flatten the Hierarchy During Synthesis", "Fast Input, Output, and Output Enable Registers", and "Hierarchy Assignments" sections • Updated Table 13–6 • Added the "Spine Clock Limitations" section • Removed the Change State Machine Encoding section from page 19 • Removed Figure 13-5 • Minor text edits throughout the chapter
May 2011	11.0.0	• Reorganized sections in "Initial Compilation: Optional Fitter Settings" section • Added new information to "Resource Utilization" section • Added new information to "Duplicate Logic for Fan-Out Control" section • Added links to Help • Additional edits and updates throughout chapter
December 2010	10.1.0	• Added links to Help • Updated device support • Added "Debugging Timing Failures in the Timing Analyzer" section • Removed Classic Timing Analyzer references • Other updates throughout chapter
August 2010	10.0.1	Corrected link
July 2010	10.0.0	• Moved Compilation Time Optimization Techniques section to new *Reducing Compilation Time* chapter • Removed references to Timing Closure Floorplan • Moved Smart Compilation Setting and Early Timing Estimation sections to new *Reducing Compilation Time* chapter • Added Other Optimization Resources section • Removed outdated information • Changed references to DSE chapter to Help links • Linked to Help where appropriate • Removed Referenced Documents section

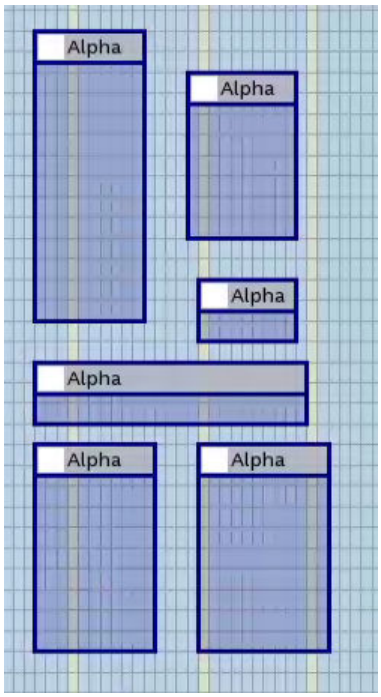
6. Analyzing and Optimizing the Design Floorplan

Determining the layout (placement) of your design elements into physical resources on the FPGA device is known as floorplanning. Floorplanning is a critical design step that helps to ensure that the Compiler places important design logic in the most effective locations for optimum performance and rapid timing closure.

By default, the Compiler determines the best location for logic placement based on your design characteristics and project settings and constraints. You can use the Quartus Prime Chip Planner to visualize the available device resources, and then use a variety of constraints to implement specific placement for important logic, and to group blocks together within specific device regions.

For example, you can define a Logic Lock placement constraint to assign design logic to any arbitrary region of physical resources on the target device that you define. When you assign nodes or entities to the Logic Lock region, the Compiler always places that logic inside the region during fitting. You can define the Logic Lock region's size and location.

Figure 74. Logic Lock Regions in Chip Planner Floorplan



After compilation, you can back-annotate (copy) the Compiler's resource assignments to preserve that same implementation in subsequent compilations. Assignment back-annotation can simplify timing closure by allowing you to lock down placement of your optimized results.

Related Information

Back-Annotate Optimized Assignments, Quartus Prime Pro Edition User Guide: Getting Started

6.1. Location Assignment Optimization Guidelines

Refer to the following optimization guidelines for assigning locations to specific registers and combinational nodes.

Guideline: Assigning Logic to Specific Locations

As part of design optimization, you may want to assign logic in your design to specific locations in the target device floorplan. You may want to make this type of assignment to preserve a good placement result, or to replicate a result in future compiles. In most cases, design partitions are the best way to preserve placement. For more information, refer to *Creating a Design Partition* in *Quartus Prime Pro Edition User Guide: Design Compilation*

Guideline: Assigning the Location of One or Two Registers

Sometimes, you may want to assign the location of one or two registers or combinational nodes. In cases where the amount of logic is extremely small, a design partition is not usually practical. If you want to restrict placement to an area of the floorplan, you can use a Logic Lock region placement constraint. For more information, refer to Defining Logic Lock Placement Constraints.

Guideline: Assigning a Register to a Specific Location in an ALM

If you want to assign a register to a specific location in an ALM, you must know the specific location you want to assign. The specific location includes the X and Y coordinates of the LAB that contains the ALM, as well as the sub-location of the register in the ALM in the LAB. The easiest way to find this information during design optimization is to start from a compiled version of the design, and review a timing path report in the Timing Analyzer. Use the string in the Location column as the value for a location assignment.

For example, to assign the `LOOP[42].my_div|r[6]` register to the location shown in Example Compilation Results, use the following QSF statement:

```
set_location_assignment -to LOOP[42].my_div|r[6] FF_X117_Y26_N49
```

Figure 75. Example Compilation Results

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element Type	Element
17	8.103	0.317	RR	IC	1	FF_X117_Y26_N49	ALM Register	LOOP[42].my_div|r[6]|d
18	8.103	0.000	RR	CELL	1	FF_X117_Y26_N49	ALM Register	LOOP[42].my_div|r[6]

Guideline: Making Assignments to Compiler-Modified Nodes

If you make location assignments to registers that the Compiler modifies or optimizes during compilation, is unlikely that the Compiler will honor the assignment in subsequent compiles. Compiler-modified or optimized registers include a suffix that begins with a tilde character (~). If the Compiler modifies a register during compilation, the suffix is likely to change on subsequent compiles as you change other parts of the circuit. When the suffix changes, the name associated with the assignment also changes, so the Compiler does not honor the assignment.

Guideline: Assigning Combinational Logic to Specific Locations

Generally it is not worth assigning combinational logic to specific locations. Combinational logic names are more likely to change in subtle ways from one compile to another. When the names change, any location assignments are ignored if the names don't match.

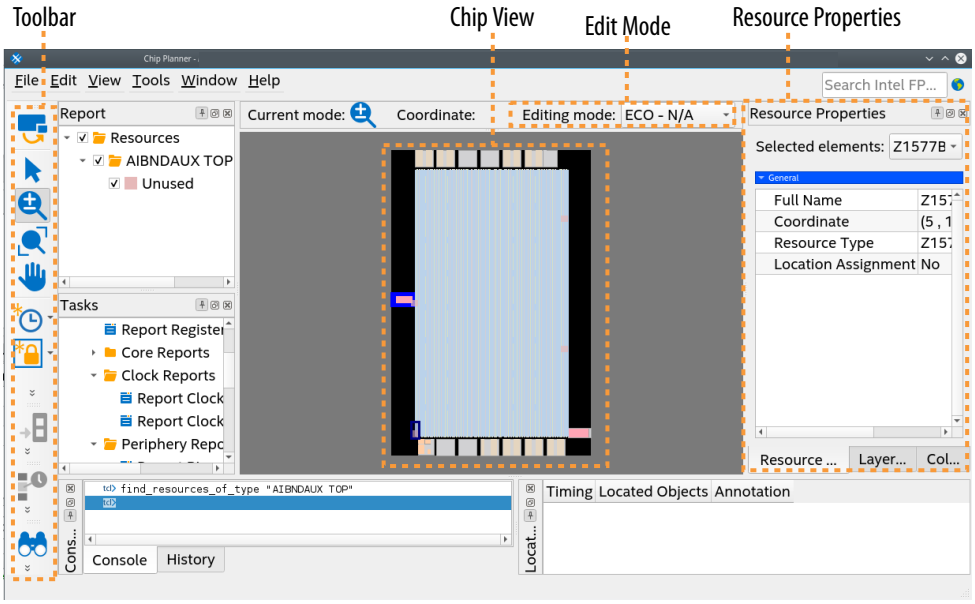
Related Information

Creating a Design Partition, Quartus Prime Pro Edition User Guide: Design Compilation

6.2. Design Floorplan Analysis in Chip Planner

The Chip Planner simplifies floorplanning by allowing you to view and constrain design logic within a visual display of the FPGA chip resources. You can use the Chip Planner to view and modify the logic placement, connections, and routing paths after running the Fitter. You can also make assignment changes, such as creating and deleting Logic Lock, clock region, and resource assignments.

Figure 76. The Chip Planner

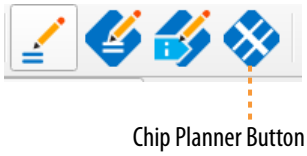


6.2.1. Starting the Chip Planner

To start the Chip Planner, select **Tools ➤ Chip Planner**. You can also start the Chip Planner by using any of the following methods:

- Click the Chip Planner button on the Quartus Prime software toolbar.

Figure 77. Chip Planner Button on Toolbar

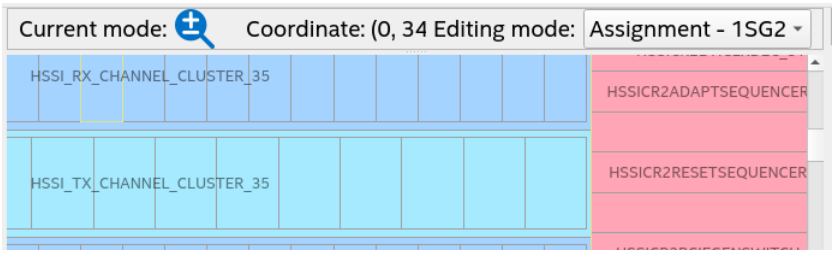


- In the following tools, right-click any chip resource and select **Locate ➤ Locate in Chip Planner**:
  - Compilation Report
  - **Logic Lock Regions Window**
  - Technology Map Viewer
  - **Project Navigator** window
  - Node Finder
  - Simulation Report
  - Report Timing panel of the Timing Analyzer

6.2.2. Chip Planner GUI

The Chip Planner GUI helps you to visualize and modify the use of device resources for your design. As you zoom in, the level of abstraction decreases, revealing more details about your design.

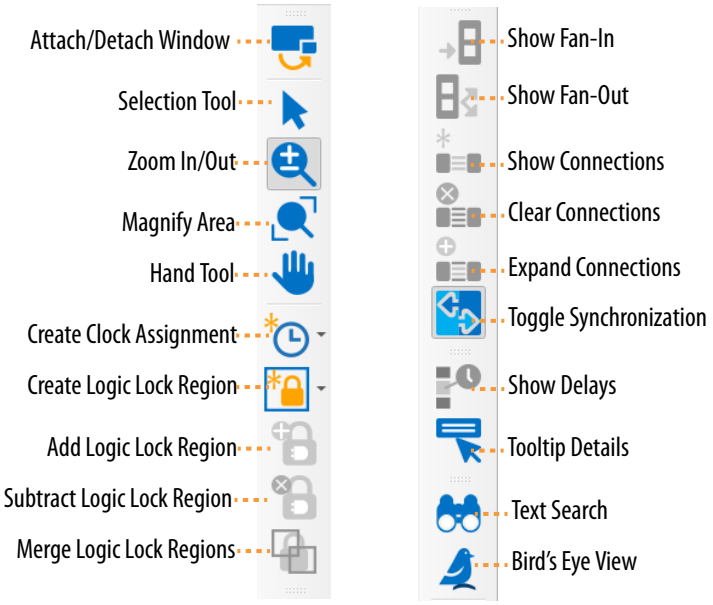
Figure 78. Zoom to View Device Resource Details in Chip Planner



Chip Planner Toolbar

The Chip Planner toolbar provides access to the main Chip Planner functions for visualizing and modifying device resources. Alternatively, you can access the same Chip Planner commands from the Chip Planner **View** menu.

Figure 79. Chip Planner Toolbar

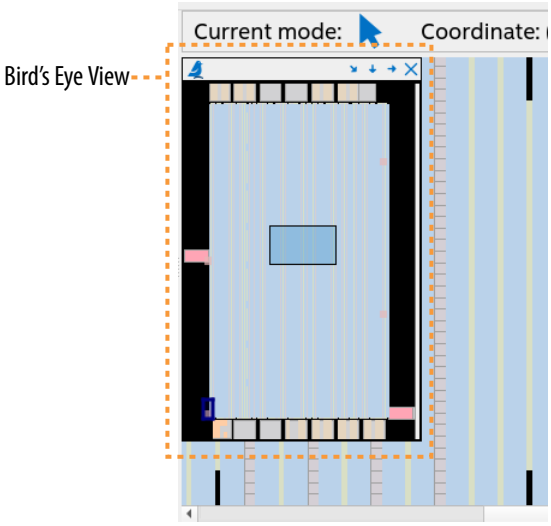


Chip Planner Floorplan Views

The Chip Planner includes multiple views to shows various levels of detail for the targeted Intel FPGA device. You can toggle between these different views when you require more or less detail. As you zoom in to the chip, the level of abstraction decreases, revealing more details about the resources that your design targets.

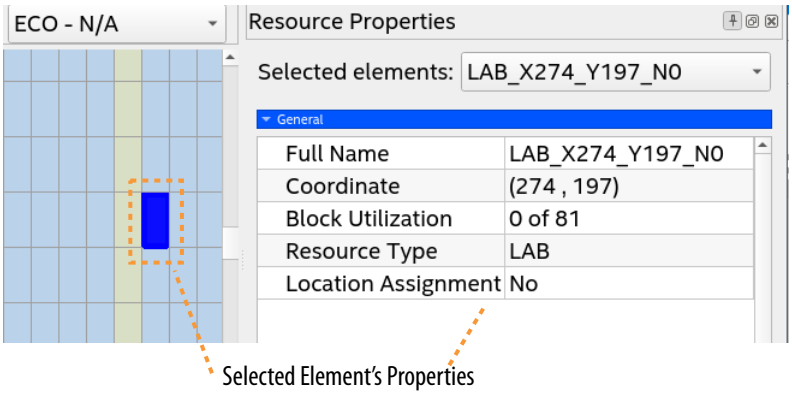
Click the **Bird's Eye View** button to instantly display a summary high-level chip view, on top of your current Chip Planner view. Use this Bird's Eye view to show your current selection within the larger chip, and to navigate quickly between areas of interest.

Figure 80. Bird's Eye View



The Bird's Eye View is particularly useful when the parts of your design that you want to view are at opposite ends of the chip, allowing you to quickly navigate between resource elements without losing the current frame of reference.

Figure 81. Selected Element Properties



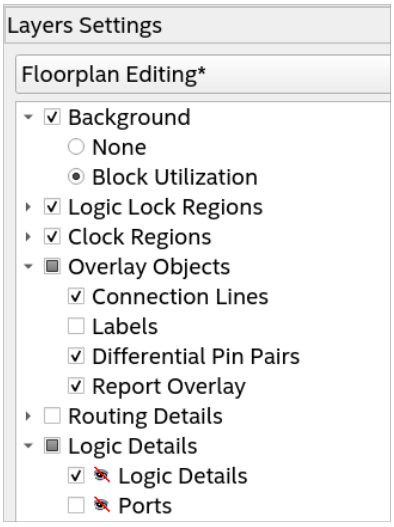
When you select any element in the Chip Planner, the **Properties** window displays the detailed properties of the objects (such as atoms, paths, Logic Lock regions, or routing elements). To display the **Properties** window, right-click the object and select **View ➤ Properties**.

Layers Settings Pane

clicking **View ➤ Layers Settings** to customize which device structures the Chip Planner displays.

You can select the **Basic**, **Detailed**, or **Floorplan Editing** settings that are preconfigured for specific planning tasks, or specify your own layer settings.

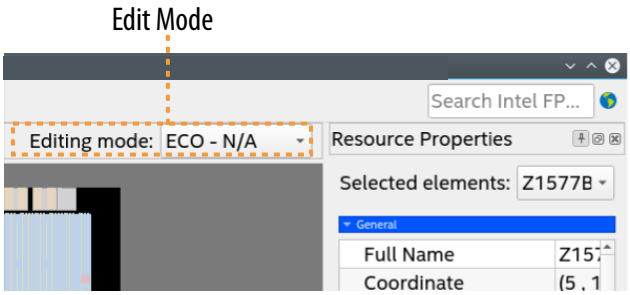
Figure 82. Layer Settings Control Display of Device Resources



Editing Mode

The Chip Planner has two editing modes.

Figure 83. Editing Mode Selection



- **Assignment**—editing mode allows you to make assignment changes that are implemented the next time you run the Fitter.
- **ECO**—editing mode allows you to make post-compilation changes, commonly referred to as engineering change orders (ECOs), without running a full compilation.

Locate History

The **Locate History** window records all searches you perform using the **Locate in Chip Planner** command, allowing you to quickly rerun common searches.

6.2.3. Viewing Design Elements in Chip Planner

The Chip Planner allows you to locate and report details on various elements of your design, such as viewing available clock networks, routing congestion, I/O banks, design partitions, and high-speed serial interfaces in the floorplan.

The following section describes how to view various design elements in the Chip Planner.

6.2.3.1. Viewing Architecture-Specific Design Information in Chip Planner

The Chip Planner allows you to view architecture-specific information related to your design. By enabling the options in the **Layers Settings** pane and **Properties** tab, you can view:

- **Device routing resources used by your design**—view how blocks are connected, as well as the signal routing that connects the blocks.
- **LE configuration**—view logic element (LE) configuration in your design. For example, you can view which LE inputs are used; whether the LE utilizes the register, the look-up table (LUT), or both; as well as the signal flow through the LE.
- **ALM configuration**—view ALM configuration in your design. For example, you can view which ALM inputs are used; whether the ALM utilizes the registers, the upper LUT, the lower LUT, or all of them. You can also view the signal flow through the ALM.

- **I/O configuration**—view device I/O resource usage. For example, you can view which components of the I/O resources are used, whether the delay chain settings are enabled, which I/O standards are set, and the signal flow through the I/O.
- **PLL configuration**—View phase-locked loop (PLL) configuration in your design. For example, you can view which control signals of the PLL are used with the settings for your PLL.
- **Timing**—view the delay between the inputs and outputs of FPGA elements. For example, you can analyze the timing of the DATAB input to the COMBOUT output.

6.2.3.2. Viewing Available Clock Networks in Chip Planner

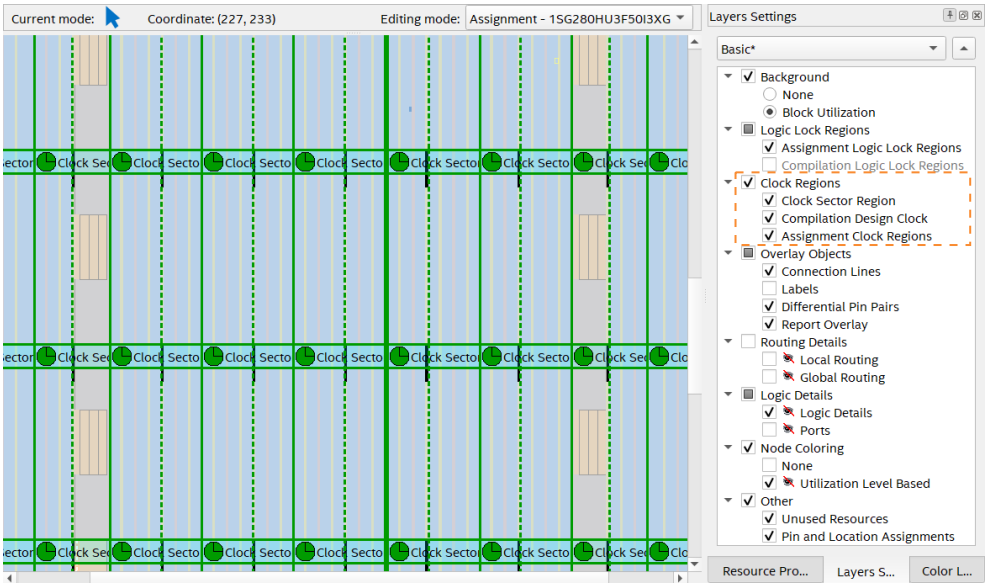
When you enable a clock region layer in the **Layers Settings** pane, you display the areas of the chip that are driven by global and regional clock networks. When the selected device does not contain a given clock region, the option for that category is unavailable in the dialog box.

Depending on the clock layers that you activate in the **Layers Settings** pane, the Chip Planner displays regional and global clock regions in the device, and the connectivity between clock regions, pins, and PLLs.

Note: The Stratix 10 and Agilex 7 device clocking architecture does not include regional clocks nor spine clocks.

Clock regions appear as rectangular overlay boxes with labels indicating the clock type and index. Select a clock network region by clicking the clock region. The clock-shaped icon at the top-left corner indicates that the region represents a clock network region.

Figure 84. Clock Regions



Spine/sector clock regions have a dotted vertical line in the middle. This dotted line indicates where two columns of row clocks meet in a sector clock.

To change the color in which the Chip Planner displays clock regions, select **Tools ➤ Options ➤ Colors ➤ Clock Regions**.

Related Information

Spine Clock Limitations on page 116

6.2.3.3. Viewing Clock Sector Utilization in Chip Planner

The Chip Planner provides a visual representation of a design's clock sector utilization.

To generate the report in the Chip Planner:

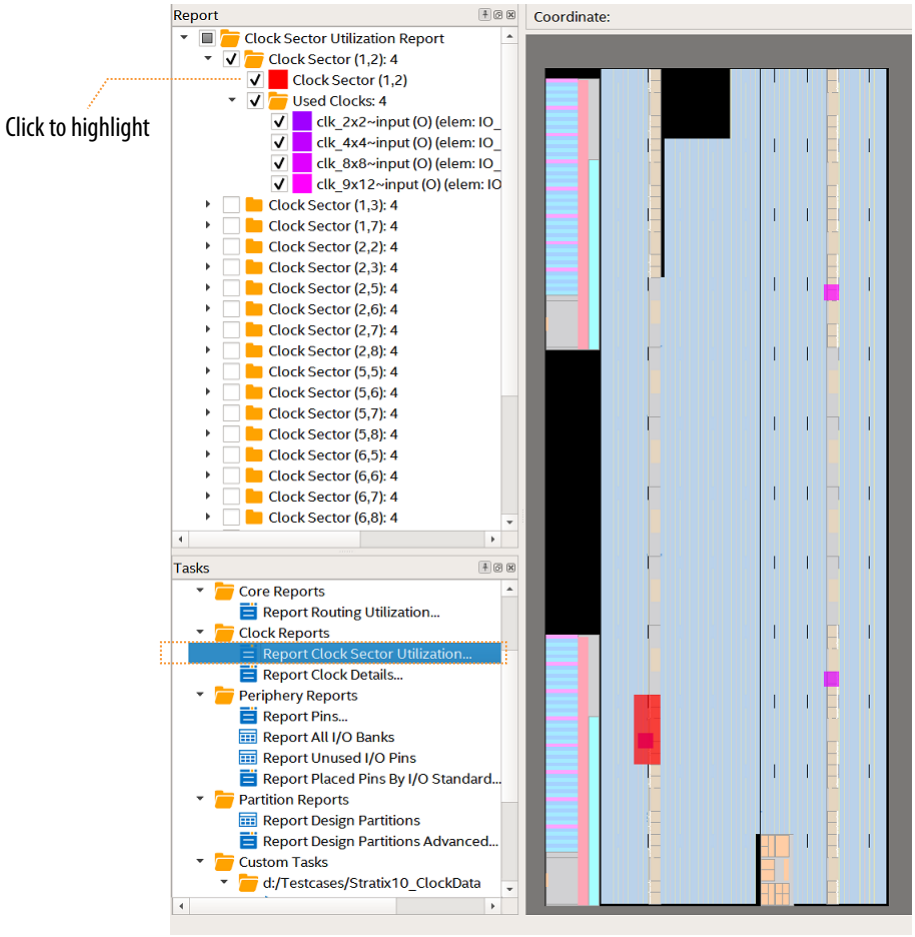
1. In the **Tasks** pane, double-click **Report Clock Sector Utilization** to open the **Report Clock Sector Utilization** dialog box.
2. If you want the report to include the source nodes, turn on **Report source nodes**.
   The equivalent TCL command appears at the bottom of the Dialog Box.
3. Click **OK**.

   The report output shows the most used clock sectors.

The **Report** pane displays a list of clock sectors, with colors according to utilization. The clock sector with the highest utilization appears in red, and the sector with least utilization appears in blue.

You can turn on or off the sector visibility from the **Report** pane. You can also highlight nodes, if applicable.

Figure 85. Clock Sector Utilization Report



6.2.3.4. Viewing Routing Congestion in Chip Planner

The **Report Routing Utilization** task allows you to determine the percentage of routing resources in use following a compilation. This feature can identify zones with lack of routing resources, helping you to make design changes to meet routing congestion design requirements.

To view the routing congestion in the Chip Planner:

1. In the **Tasks** pane, double-click the **Report Routing Utilization** command to launch the **Report Routing Utilization** dialog box.
2. Click **Preview** in the **Report Routing Utilization** dialog box to preview the default congestion display.
3. Change the **Routing Utilization Type** to display congestion for specific resources.
   The default display uses dark blue for 0% congestion (blue indicates zero utilization) and red for 100%. You can adjust the slider for **Threshold percentage** to change the congestion threshold level.

The congestion map helps you determine whether you can modify the floorplan, or modify the RTL to reduce routing congestion. Consider:

- The routing congestion map uses the color and shading of logic resources to indicate relative resource utilization; darker shading represents a greater utilization of routing resources. Areas where routing utilization exceeds the threshold value that you specify in the **Report Routing Utilization** dialog box appear in red.
- To identify a lack of routing resources, you must investigate each routing interconnect type separately by selecting each interconnect type in turn in the **Routing Utilization Settings** dialog box.
- The Compiler's messages contain information about average and peak interconnect usage. Peak interconnect usage over 75%, or average interconnect usage over 60%, can indicate difficulties fitting your design. Similarly, peak interconnect usage over 90%, or average interconnect usage over 75%, show increased chances of not getting a valid fit.

Related Information

Viewing Routing Resources on page 162

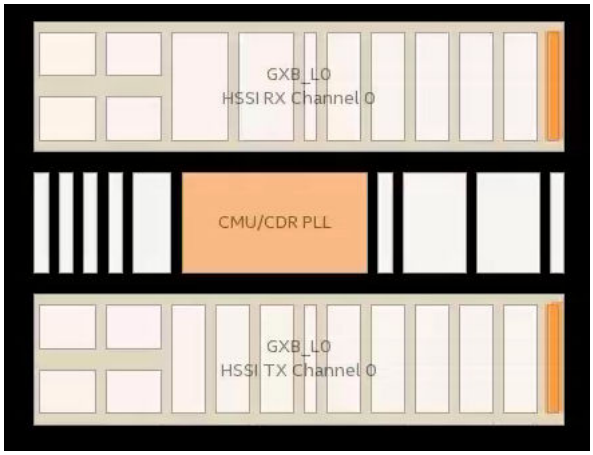
6.2.3.5. Viewing I/O Banks in Chip Planner

To view the I/O bank map of the device in the Chip Planner, double-click **Report All I/O Banks** in the **Tasks** pane.

6.2.3.6. Viewing High-Speed Serial Interfaces (HSSI) in Chip Planner

The Chip Planner displays a detailed block view of the receiver and transmitter channels of the high-speed serial interfaces. To display the HSSI block view, double-click **Report HSSI Block Connectivity** in the **Tasks** pane.

Figure 86. Arria 10 HSSI Channel Blocks



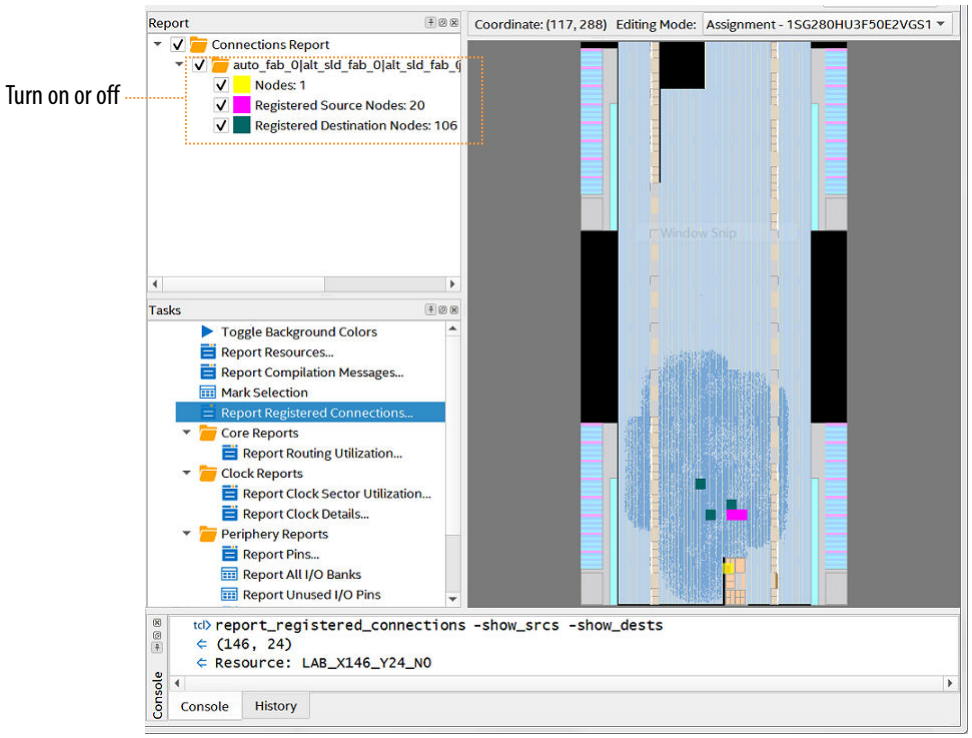
6.2.3.7. Viewing Source and Destination Nodes in Chip Planner

The Chip Planner allows you to view the registered fan-in or fan-outs of nodes in compiled designs with the **Report Registered Connections** task. This report is different from the **Generate Fanin/Fanout connections** report in that the source and destination nodes appear without connection lines, which may obscure the view.

1. In the Chip Planner, select one or more nodes.
2. In the **Task** pane, double-click **Report Registered Connections**.
3. Select the options from the dialog box, and click **OK**.

The **Reports** pane displays the registered source and destination nodes. Turn on or off to switch visibility in the graphic view.

Figure 87. Report Registered Connections



Related Information

Viewing Fan-In and Fan-Out in Chip Planner on page 153

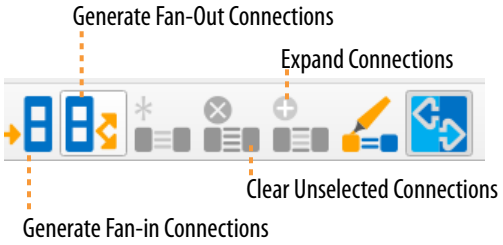
6.2.3.8. Viewing Fan-In and Fan-Out in Chip Planner

Displays the atoms that fan-in to or fan-out from a resource, including connectivity lines.

To display the fan-in or fan-out connections from a resource you selected,

1. In the Chip Planner toolbar, click the **Generate Fan-In Connections** button or the **Generate Fan-Out Connections** button.

Figure 88. Chip Planner Toolbar Buttons



2. To remove other connections that appear on the Chip Planner view, click the **Clear Unselected Connections** button.

You can also perform this actions from the Chip Planner **View** menu.

Related Information

Viewing Source and Destination Nodes in Chip Planner on page 152

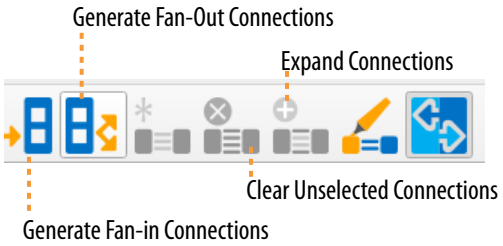
6.2.3.9. Viewing Immediate Fan-In and Fan-Out in Chip Planner

Displays the immediate fan-in or fan-out connection for the selected atom.

For example, when you view the immediate fan-in for a logic resource, you see the routing resource that drives the logic resource. You can generate immediate fan-ins and fan-outs for all logic resources and routing resources.

- To display the immediate fan-in or fan-out connections, click **View ➤ Generate Immediate Fan-In Connections** or **View ➤ Generate Immediate Fan-Out Connections**.
- To remove the connections displayed, use the **Clear Unselected Connections** button in the Chip Planner toolbar.

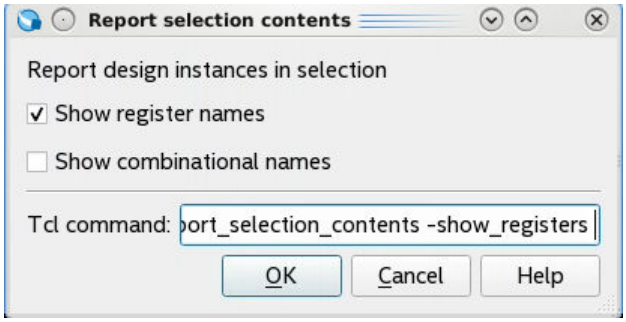
Figure 89. Chip Planner Toolbar Buttons



6.2.3.10. Viewing the Selected Contents in Chip Planner

You can view a detailed report of the contents of any area that you select in the Chip Planner. When you view the contents of a selected area, Chip Planner generates a hierarchical, color coded list of the design elements in the selection. This functionality allows you to quickly determine where the Compiler places specific modules of the design.

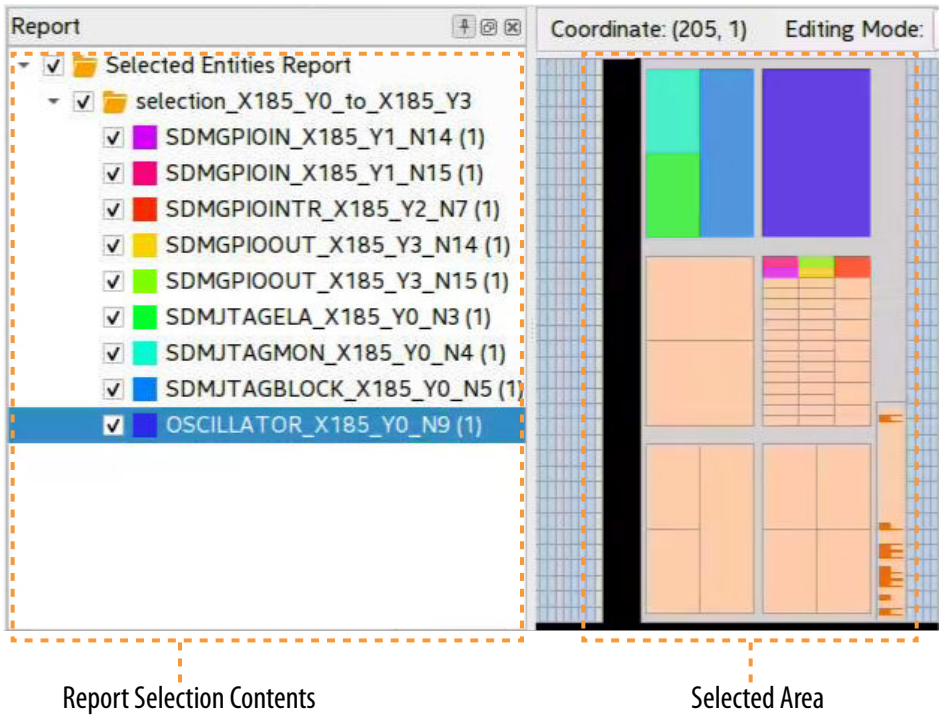
Figure 90. Report Selection Contents Dialog Box



Follow these steps to view selected contents in the Chip Planner:

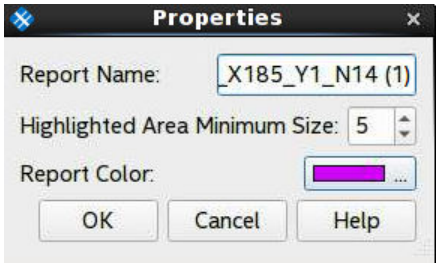
1. In the **Tasks** pane, double-click **Report Selection Contents**. The **Report Selection Contents** dialog box appears.
2. Under **Report design instances in selection**, turn on or off **Show registers names** and **Show combinational names** to display names of those type in the report.
3. Click **OK**. The report generates and displays the list of selected elements in the **Reports** pane.

Figure 91. Viewing Selected Contents



4. To customize the color coding of report folders, right-click any report, and then click **Properties**. You can customize the **Report Name**, **Report Color**, and the **Highlighted Area Minimum Size** for the report.

Figure 92. Selected Entities Report Properties



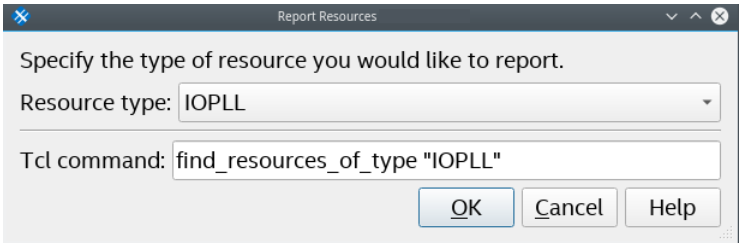
6.2.3.11. Viewing the Location and Utilization of Device Resources in Chip Planner

Chip Planner can generate reports about the different types of device resources located in the in Chip Planner view, including the resource location and utilization.

To view the location and utilization of a device resource in Chip Planner:

1. On the Chip Planner **Tasks** pane, click **Report Resources**. The **Report Resources** dialog box appears.

Figure 93. Report Resources Dialog Box



2. In **Resource Type**, select the device resource type that you want to locate.
3. Click **OK**. The report generates and displays the list of selected resources in the **Reports** pane.
4. In the Reports pane, right-click a resource type to Zoom to Report or view the Properties of the resource in the Resources report.

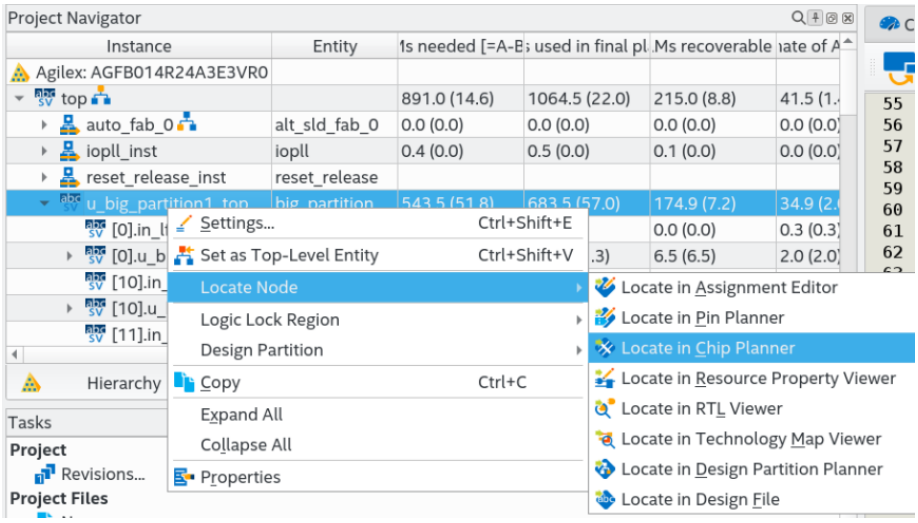
6.2.3.12. Viewing Module Placement by Cross-Probing to Chip Planner

You can use cross-probing to determine the location of a design module on the device in Chip Planner.

To view the placement of a module with cross-probing following place and route:

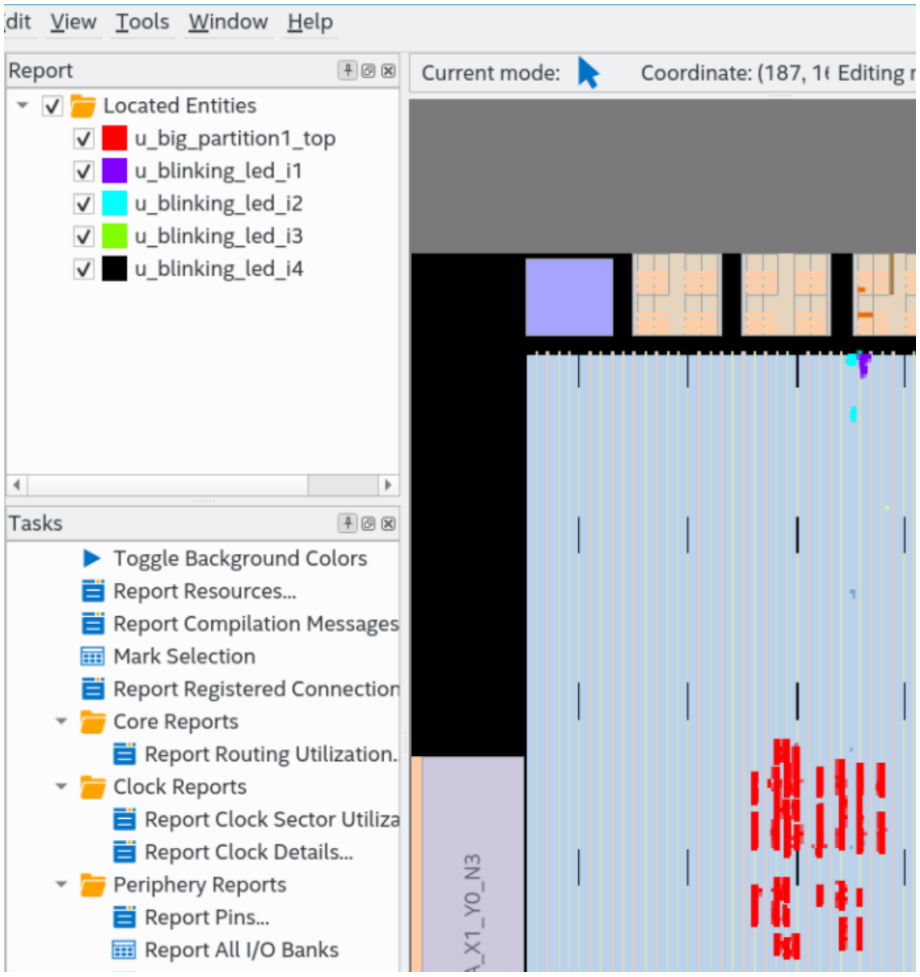
1. On the Project Navigator Hierarchy tab, right-click on a module name and click **Locate in Chip Planner**.

Figure 94. Locate in Chip Planner Command from Project Navigator



2. In the Reports pane, view the list of **Located Entities** color-coded and separated by module.

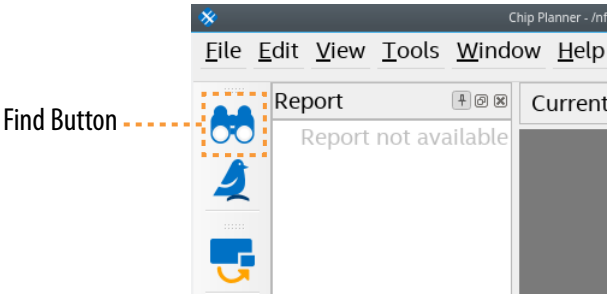
Figure 95. Located Entities in Reports Pane



6.2.4. Finding Design Elements in the Chip Planner

Use the **Find** tab to locate any design atom, port, location, or routing element by name within the Chip Planner view. The **Find** tab located in Chip Planner, you can view more information about the element on the **Properties** tab.

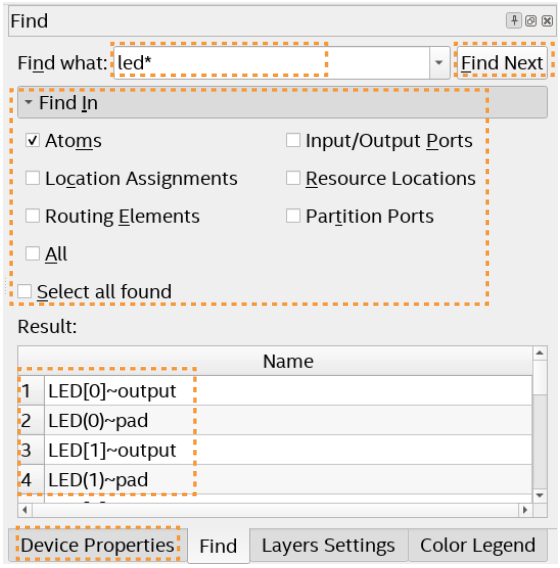
Figure 96. Click the Find Button to Search Chip Planner



To use the **Find** tab to locate design elements in Chip Planner:

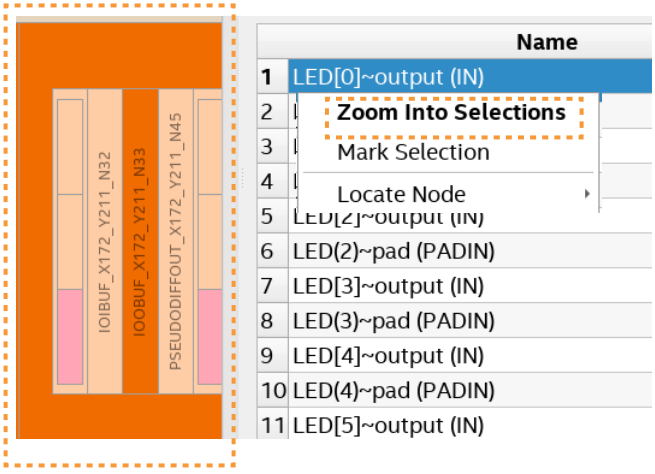
1. In Chip Planner, click the **Find** button. The **Find** tab opens.
2. In **Find what**, enter the design element name(s) (and any wildcard characters) to find in Chip Planner.
3. Under **Find In**, enable or disable the types of design elements to include in the search, as Find Options (Chip Planner Search) on page 160 describes.

Figure 97. Find Options and Controls



4. To begin the search, click **Find Next**. The search results display in the **Results** list.
5. To view details about any item in the **Results** list, click the **Properties** tab.
6. To locate found elements in Chip Planner, right-click the design element in the **Results** list and click **Zoom Into Selections**. Chip Planner zooms into the selected elements.

Figure 98. Zoom Into Selections from Search Results



6.2.4.1. Find Options (Chip Planner Search)

You can enable or disable the following **Find In** options to narrow or expand the search for design elements in Chip Planner:

Table 40. Find In Options

Option	Description
Atoms	Finds all matching atom names in the design within Chip Planner.
Location Assignments	Finds all matching location assignments in the design within Chip Planner.
Routing Elements	Find all matching routing element names in the design within Chip Planner.
All	Find all matching elements of all supported types within Chip Planner.
Input/Output Ports	Find all matching input and output port names in the design within Chip Planner.
Resource Locations	Find all matching routing element names in the design within Chip Planner.
Partition Ports	Find all matching partition port names in the design within Chip Planner.
Select all found	Automatically selects all found elements following search.

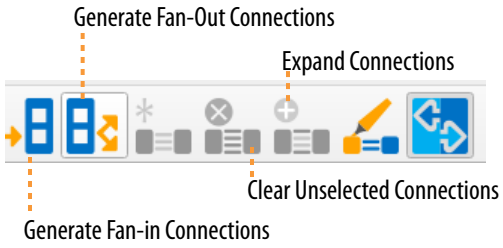
6.2.5. Exploring Paths in the Chip Planner

Use the Chip Planner to explore paths between logic elements. The following examples use the Chip Planner to traverse paths from the Timing Analysis report.

6.2.5.1. Analyzing Connections for a Path

To determine the elements forming a selected path or connection in the Chip Planner, click the **Expand Connections** button in the Chip Planner toolbar.

Figure 99. Chip Planner Toolbar Buttons



6.2.5.2. Locate Path from the Timing Analysis Report to the Chip Planner

To locate a path from the Timing Analysis report to the Chip Planner, perform the following steps:

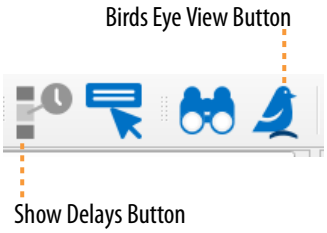
1. Select the path you want to locate in the Timing Analysis report.
2. Right-click the path and point to **Locate Path ➤ Locate in Chip Planner**. The path appears in the **Locate History** window of the Chip Planer.

Figure 100. Path List in the Locate History Window

6.2.5.3. Show Delays

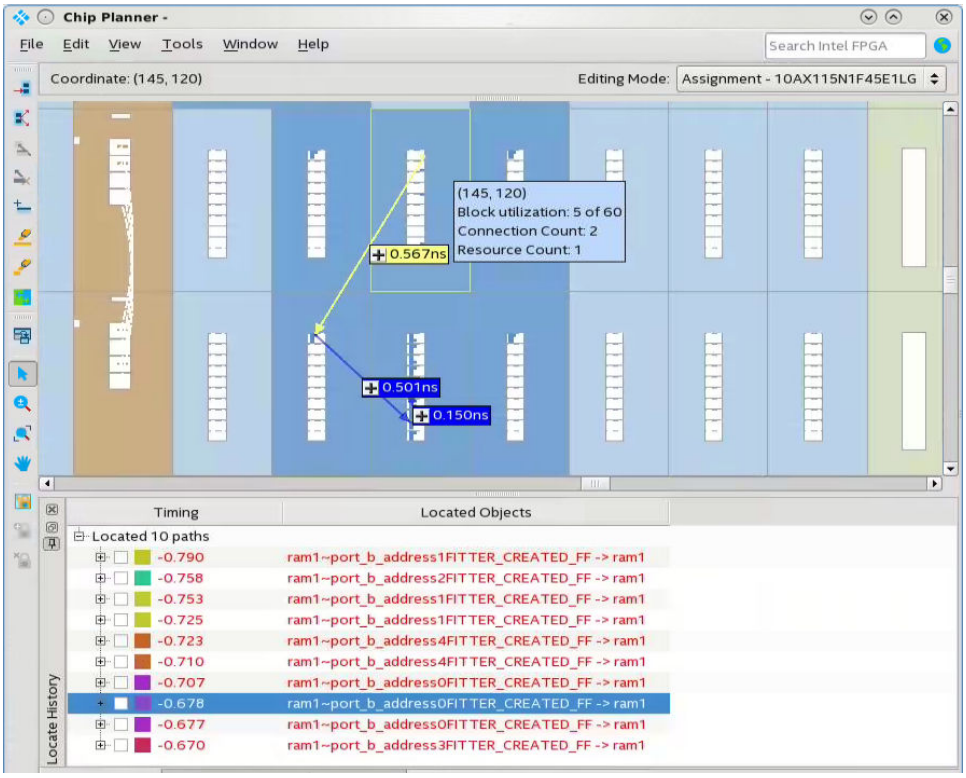
With the **Show Delays** feature, you can view timing delays for paths appearing in Timing Analyzer reports. To access this feature, click **View ➤ Show Delays** in the main menu. Alternatively click the **Show Delays** button in the Chip Planner toolbar. To see the partial delays on the selected path, click the "+" sign next to the path delay displayed in the **Locate History** window.

Figure 101. Show Delays Button on Chip Planner Toolbar



For example, you can view the delay between two logic resources or between a logic resource and a routing resource.

Figure 102. Show Delays Associated in a Timing Analyzer Path



6.2.5.4. Viewing Routing Resources

With the Chip Planner and the **Locate History** window, you can view the routing resources that a path or connection uses. You can also select and display the Arrival Data path and the Arrival Clock path.

In the **Locate History** window, right-click a path and select **Show Physical Routing** to display the physical path. To adjust the display, right-click and select **Zoom to Selection**.

Figure 103. Show Physical Routing

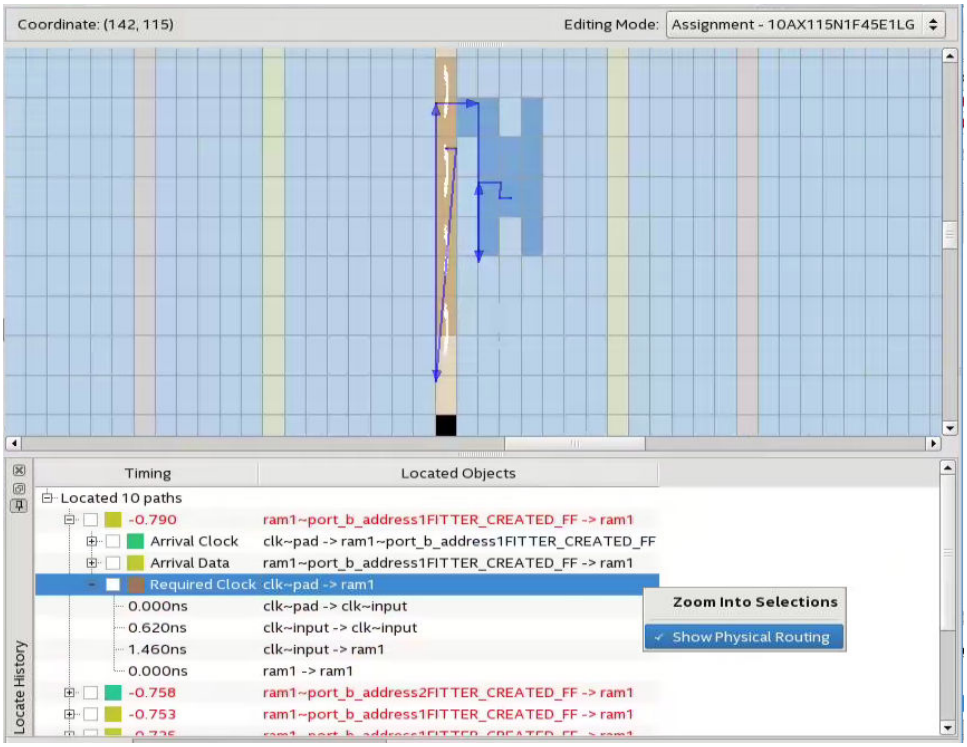
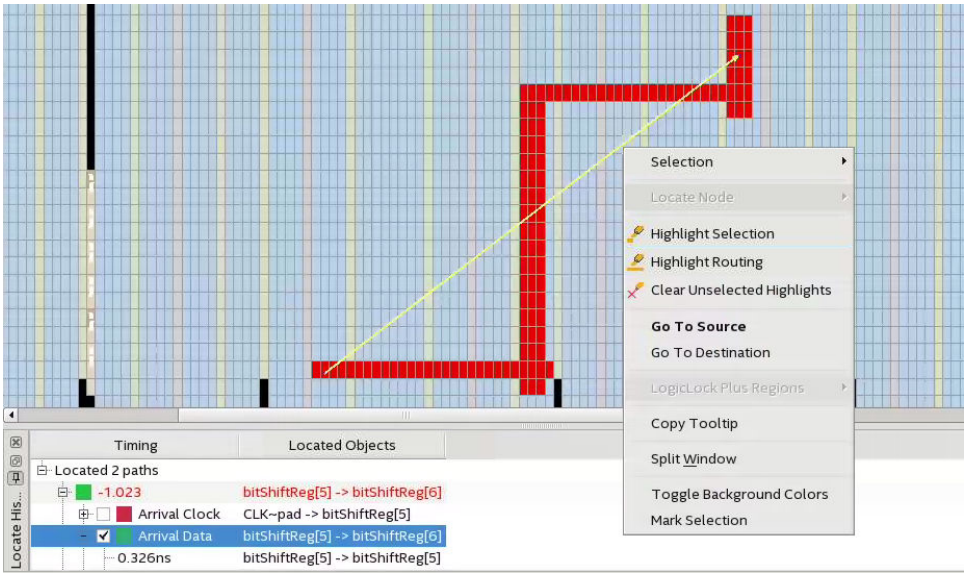


Figure 104. Highlight Routing

To see the rows and columns where the Fitter routed the path, right-click a path and select **Highlight Routing**.



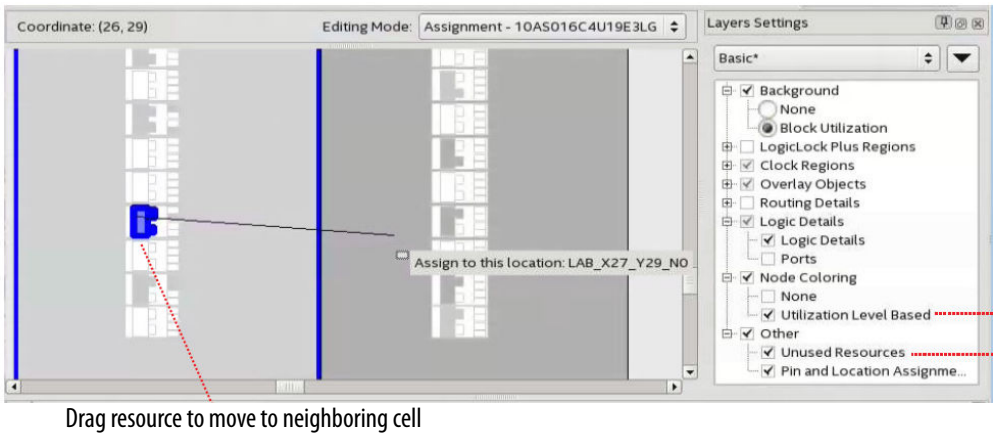
Related Information

Viewing Routing Congestion in Chip Planner on page 151

6.2.6. Viewing Assignments in the Chip Planner

You can view location assignments in the Chip Planner by selecting the appropriate layer, or any custom preset that displays block utilization in the **Layers Settings** pane. The Chip Planner displays assigned resources in a predefined color (gray, by default).

Figure 105. Viewing Assignments in the Chip Planner

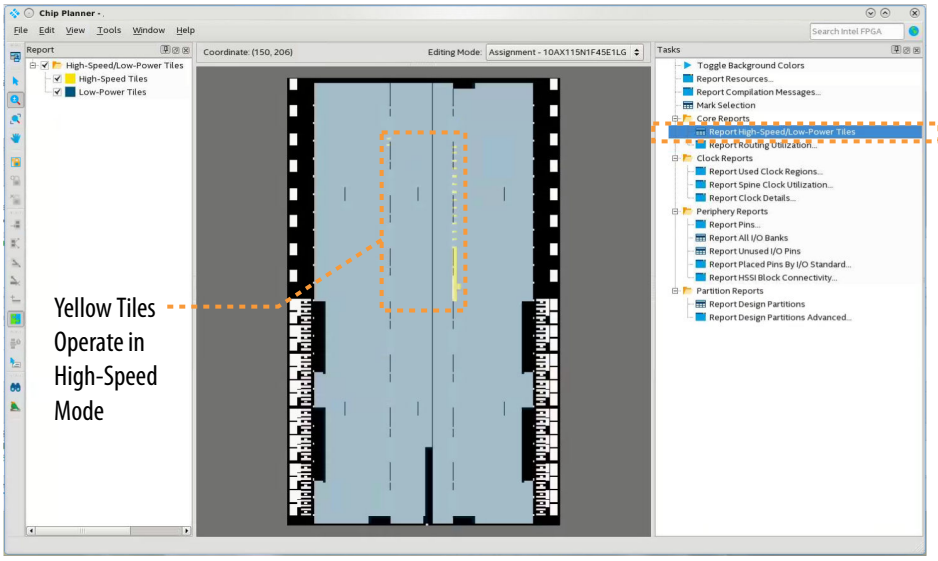


To create or move an assignment, or to make node and pin location assignments to Logic Lock regions, drag the selected resource to a new location. The Fitter applies the assignments that you create during the next place-and-route operation.

6.2.7. Viewing High-Speed and Low-Power Tiles in the Chip Planner

Some Intel devices have ALMs that can operate in either high-speed mode or low-power mode. The power mode is set during the fitting process in the Quartus Prime software. These ALMs are grouped together to form larger blocks, called "tiles".

Figure 106. High-Speed and Low Power Tiles in an Arria 10 Device



To view a power map, double-click **Tasks ➤ Core Reports ➤ Report High-Speed/ Low-Power Tiles** after running the Fitter. The Chip Planner displays low-power and high-speed tiles in contrasting colors; yellow tiles operate in a high-speed mode, while blue tiles operate in a low-power mode.

6.2.8. Viewing Design Partition Placement

With the **Report Design Partitions** command, you can view the physical placement of design partitions using the same color map as the Design Partition Planner.

The **Report Design Partitions Advanced** command opens the **Report Design Partitions Advanced** dialog box that allows you to select a partition and generate a report of the pins belonging to the partition. It highlights the selected partition's boundary ports and pins in the Chip Planner, and optionally reports the routing utilization and routing element details.

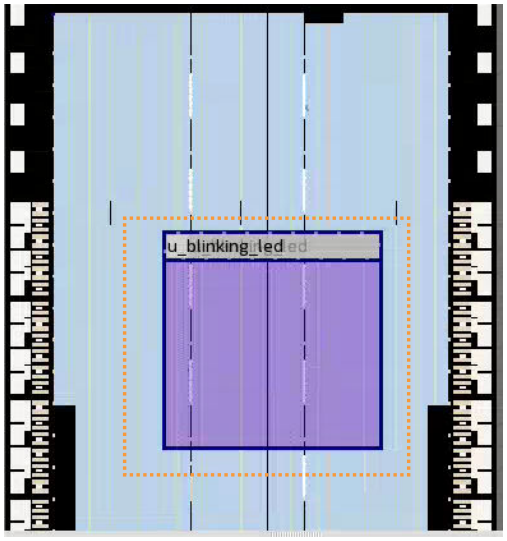
6.3. Defining Logic Lock Placement Constraints

A Logic Lock region is a powerful type of logic placement and routing constraint. You can define any arbitrary region of physical resources on the target device as a Logic Lock region, and then assign design nodes and other properties to the region. When you constrain design nodes to a Logic Lock region, the Fitter always places those nodes within the region resulting in more predictable results with each design iteration.

Your floorplan can contain multiple Logic Lock regions, depending on your design characteristics. You can also define a routing region as part of a Logic Lock region. The routing region specifies the routing area constraint.

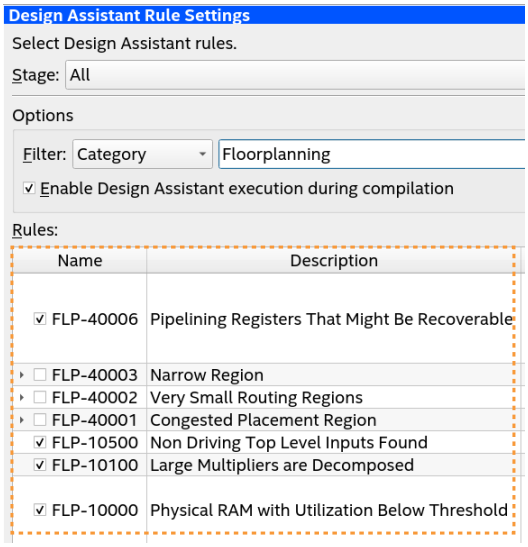
The Chip Planner makes it easy to visualize and constrain device resources within a device floorplan. You can draw or specify the dimensions of a Logic Lock region in the floorplan using the Logic Lock Regions window. After running synthesis or fitting, you can then assign design nodes as members of the region to implement the constraint.

Figure 107. u_blinking_led Logic Lock Region Defined in Chip Planner



To detect and resolve any potential problems with Logic Lock regions in your project, click **Report DRC** to run the Design Assistant to check for the FLP rule category. FLP Design Assistant rules detect possible issues with floorplanning and Logic Lock regions.

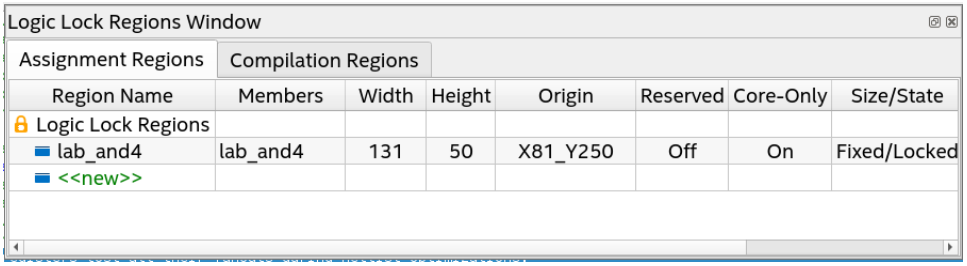
Figure 108. Floorplanning (FLP) Design Assistant Rules



6.3.1. The Logic Lock Regions Window

Use the Logic Lock Regions window GUI to view, define, and modify the attributes of Logic Lock regions in your project.

Figure 109. Logic Lock Regions Window



The Logic Lock Regions window organizes the constraints into the following two tabs:

Table 41. Logic Lock Regions Window Tabs

Logic Lock Regions Window Tab	Description
Assignment Regions	Displays the properties of all Logic Lock regions that you define in the current project (saved in the `.qsf`). Modify Logic Lock region properties in this tab.
Compilation Regions	Displays the properties of any Logic Lock region contained in a `.qdb` file in the project. Tab is read-only because you must edit imported Logic Lock region properties in the original project.

Open the Logic Lock Regions window by clicking:

- **Assignments ➤ Logic Lock Window**.
- **View ➤ Logic Lock Window** in the Chip Planner.
- Right-click **Logic Lock Region ➤ Logic Lock Regions Window** in Project Navigator.

You can customize the appearance of Logic Lock Regions window by dragging the columns to change their order and showing or hiding optional columns by right-clicking any column.

6.3.2. Defining Logic Lock Regions

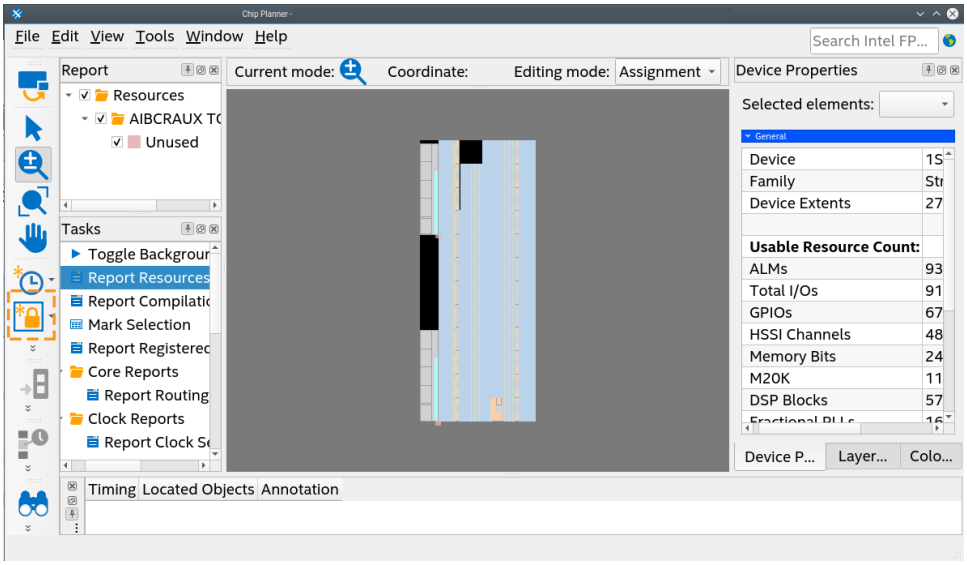
The Quartus Prime provides multiple entry points in the GUI to create and modify Logic Lock constraints as appropriate in your workflow.

- Before Analysis & Elaboration—you can use the Logic Lock Regions window and the Chip Planner to visualize the chip and define empty regions without member nodes.
- After Analysis & Elaboration or Fitter—you can assign member nodes, and even define a Logic Lock region from a selected design entity.

You can assign an entity in the design to only one Logic Lock region, but the entity can inherit regions by hierarchy. This hierarchy allows a reserved region to have a sub region without reserving the resources in the sub region.

If a Logic Lock region boundary includes part of a device resource, the Quartus Prime software allocates the entire resource to that Logic Lock region.

Figure 110. Chip Planner with Logic Lock Button on Toolbar



6.3.2.1. Defining a Logic Lock Region in Chip Planner

The Chip Planner allows you to easily see Logic Lock region locations and properties in relation to other resources in the device.

Before Analysis and Elaboration, the Chip Planner displays the device floorplan resources that are available. You can define Logic Lock regions in this floorplan. After running Analysis & Elaboration, you can add member nodes to the region.

To draw a Logic Lock region in the Chip Planner:

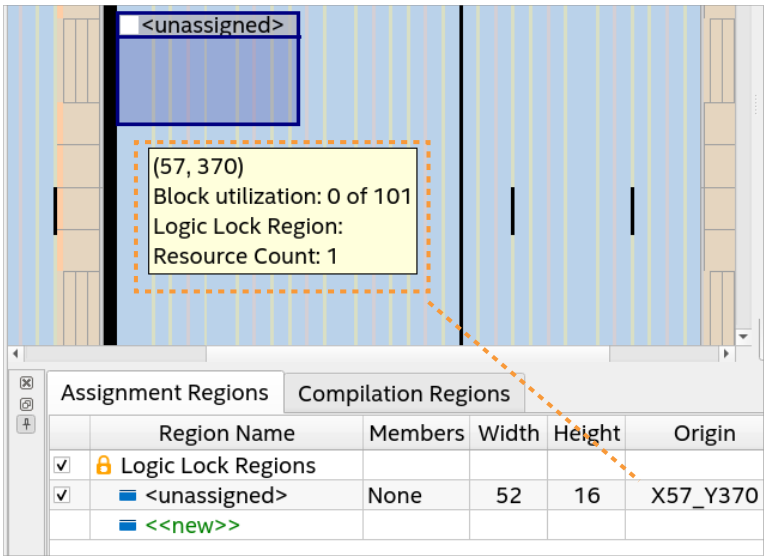
1. Open an Quartus Prime project.
2. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.

   Note: You can this step if you want to reserve the empty region without adding member nodes yet.
3. To open the Chip Planner, click **Tools ➤ Chip Planner**. Chip Planner opens and loads device resource information.
4. Click the **Create Logic Lock Region** button on the Chip Planner Toolbar.

Figure 111. Create Logic Lock Region Button on Toolbar

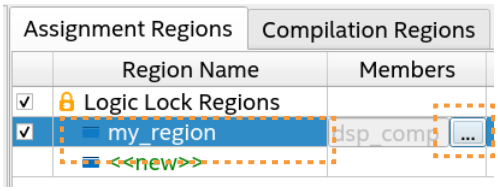
5. To define the region dimensions and location, click and drag the cursor on the Chip Planner floorplan to draw a region of your preferred location and size. An **<<unassigned>>** Logic Lock region appears in the Chip Planner and Logic Lock Regions window at the coordinates you specify.

Figure 112. Drag Cursor to Define Region Location and Size



6. In the Logic Lock Regions window, double-click **<<unassigned>>** and type a descriptive name for the region.
7. To add member nodes to the region, click the **Members** cell, and then click the **(...)** button to search for the nodes you want to add. You must complete step 2 before this step.

Figure 113. Specifying Region Name and Members



8. Confirm or customize the region **Width**, **Height**, and point of **Origin** settings in the Logic Lock Regions window.
9. To prevent the Fitter from placing any other logic in the region, turn on the **Reserved** option. This option is useful for preliminary floorplanning and for reserving device resources for logic to be added later. Otherwise, leave this option off.
10. To exclude periphery device resources from the region, turn on the **Core-Only** option.
11. For region **Size/State**, specify whether you or the Fitter determines the size and placement of the Logic Lock region:
    - If set to **Fixed/Locked**, the default value, you define the Logic Lock region's size and placement.
    - If set to **Auto/Floating**, the Fitter determines the size and placement of the Logic Lock region.
12. For **Routing Region**, specify the type of routing region constraint, such as **Unconstrained**, **Whole Chip**, or **Fixed Width Expansion** options. Refer to Defining Routing Regions on page 173.

6.3.2.1.1. Logic Lock Region Properties

You can view and modify the following properties for the Logic Lock regions that you define. You can access these properties using either of these methods:

- Click **Assignments ➤ Logic Lock Regions Window**.
- Right-click and existing Logic Lock region, and then click **Logic Lock Region Properties**.

Table 42. Attributes of Logic Lock Regions

Option	Values	Behavior
Width	**Number of columns**	Specifies the width of the Logic Lock region. If **Size/State** is set to **Auto/Floating**, the attribute is set to **Undetermined**.
Height	**Number of rows**	Specifies the height of the Logic Lock region. If **Size/State** is set to **Auto/Floating**, the attribute is set to **Undetermined**.
Origin	**Any Floorplan Location**	Specifies the location of the Logic Lock region on the floorplan. The origin is at the lower left corner of the Logic Lock region.
Reserved	Off | On	Prevents the Fitter from placing other logic in the region. Unless enabled, the Fitter fills unoccupied resources with other nodes and entities that have not been assigned to another region. You cannot apply the **Reserved** assignment to routing regions.

continued...

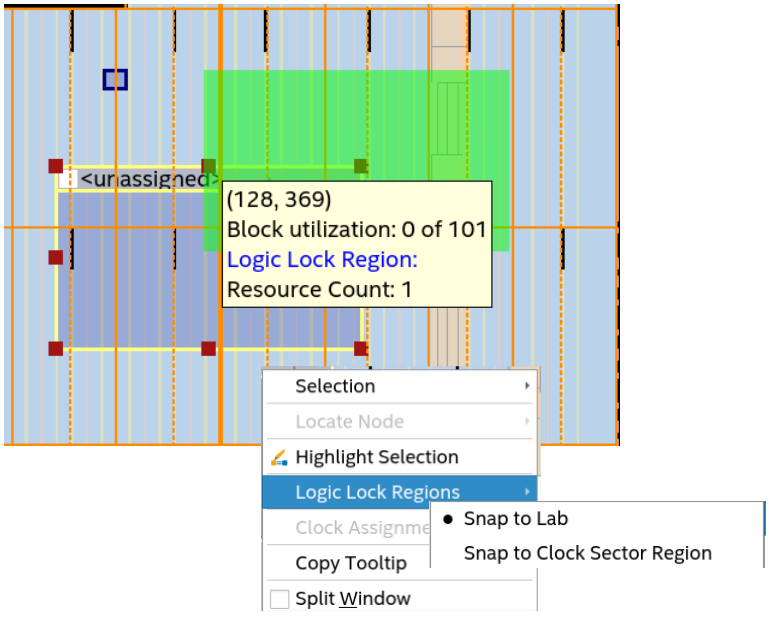
Option	Values	Behavior
Core-Only	Off | On	Excludes periphery resources from a region. Quartus Prime Pro Edition region assignments apply to periphery resources by default. If the region is designated as **Reserved** and **Core Only**, periphery resources are not reserved from the region.
Size/State	**Fixed/Locked** | **Auto/Floating**	Specifies whether you or the Fitter determines the size and placement of the Logic Lock region. • If set to **Fixed/Locked**, the default value, you define the Logic Lock region's size and placement. • If set to **Auto/Floating**, the Fitter determines the size and placement of the Logic Lock region.
Routing Region	**Unconstrained** | **Whole Chip** | **Fixed with Expansion** | **Custom**	The type of routing region constraint, such as **Unconstrained**, **Whole Chip**, or **Fixed Width Expansion** options. For more details, refer to Defining Routing Regions on page 173.

6.3.2.1.2. Snapping to a Region

When placing Logic Lock regions in the Chip Planner by hand, Chip Planner is set to snap the placement of the region to adjacent LAB boundaries by default. This means that as you drag the region to a location, the region snaps to the adjacent lab boundary when you drop the region in the floorplan.

Alternatively, you can click **View ➤ Logic Lock Regions ➤ Snap Logic Lock Region to** to toggle the snapping between **Snap to Lab** or **Snap to Clock Sector Region**. When your turn on **Snap to Clock Sector Region**, orange grid lines appear to show the clock sector region boundaries when you create, resize, or move the region.

Figure 114. Snapped to the Region



- **Creating Region**: Left-click on the mouse to create the Logic Lock region. Upon releasing the mouse, the created Logic Lock region snaps to the containing clock region or sector.
- **Resize region (and resize diagonal)**: Left-click on the mouse and drag the Logic Lock region handle. Upon releasing the mouse, the Logic Lock region resizes and snaps to the containing clock region or sector.
- **Move region**: Select and drag the Logic Lock region to highlight the clock region boundaries. Upon releasing the mouse button, the Logic Lock region moves to the new position and snaps to the containing clock region or sector.
  - **Same place and route regions are moved**: Both Logic Lock regions move and snap to the containing clock sectors.
  - **Only place | route region is moved**: The selected region moves and snaps to the clock sector, and prompts warning if the new location or size of the region does not adhere to 'place bboxes contained within route bboxes' rule.
- **Subtract or make a hole**: When performing subtract in the snap-to-clock-region mode, you create a region where the region is snapped to a clock region or a sector, and then subtract away.

6.3.2.1.3. Considerations for Auto Sized Regions

If you use **Auto/Floating** Size/State Logic Lock regions, consider the following limitations and effects:

- **Auto/Floating** regions cannot be reserved.
- Verify that your Logic Lock region is not empty. If you do not assign any instance to the region, the Fitter reduces the size to 0 by 0, making the region invalid.
- The region may or may not be associated with a partition. When you combine partitions with **Auto/Floating** Size/State Logic Lock regions, you get flexibility to solve your particular fitting challenges. However, every constraint that you add reduces the solutions available, and too many constraints can result in the Fitter not finding a solution. Some cases are:
  - If a partition is preserved at synthesis or not preserved, the Logic Lock region confines the logic to a specific area, allowing the Fitter to optimize the logic within the partition, and optimize the placement within the Logic Lock region.
  - If a partition is preserved at placement, routed, or final; a Logic Lock region is not an effective placement boundary, because the location of the partition's logic is fixed.
  - However, if the Logic Lock region is reserved, the Fitter avoids placing other logic in the area, which can help you reduce resource congestion.
- Once the outcome of the Logic Lock region meets your specification, you can:
  - Convert the Logic Lock region to **Fixed/Locked** Size/State.
  - Leave the Logic Lock region with **Auto/Floating** Size/State attribute and use the region as a “keep together” type of function.
  - If the Logic Lock region is also a partition, you can preserve the place and route through the partition and remove the Logic Lock region entirely.

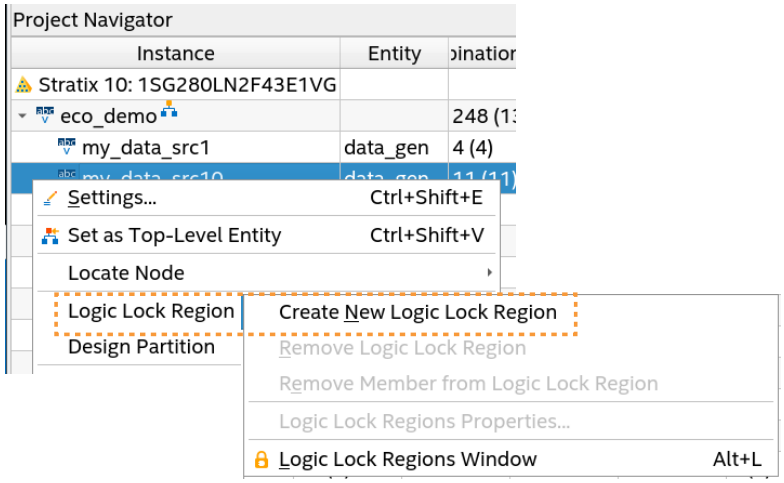
6.3.2.2. Defining a Logic Lock Region from the Project Navigator

After you run Analysis & Elaboration or the Fitter, you can assign the member nodes to the Logic Lock region. The Project Navigator facilitates easy Logic Lock constraint creation and member assignment from your design entities in a single step.

To define a Logic Lock region from the Project Navigator:

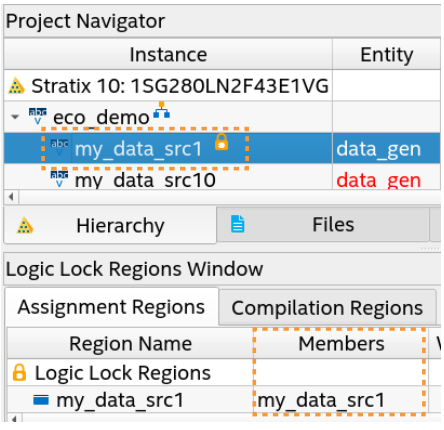
1. Run **Analysis & Elaboration** or the **Fitter (Finalize)** from the Compilation Dashboard. The Project Navigator displays the design hierarchy when complete.
2. Expand the design hierarchy in the Project Navigator, right-click any design entity, and click **Logic Lock Region ➤ Create New Logic Lock Region**.

Figure 115. Create New Logic Lock Region



The new region appears with the assigned node in the Logic Lock Regions window. Modify the **Region Name** and other properties in the Logic Lock Regions window.

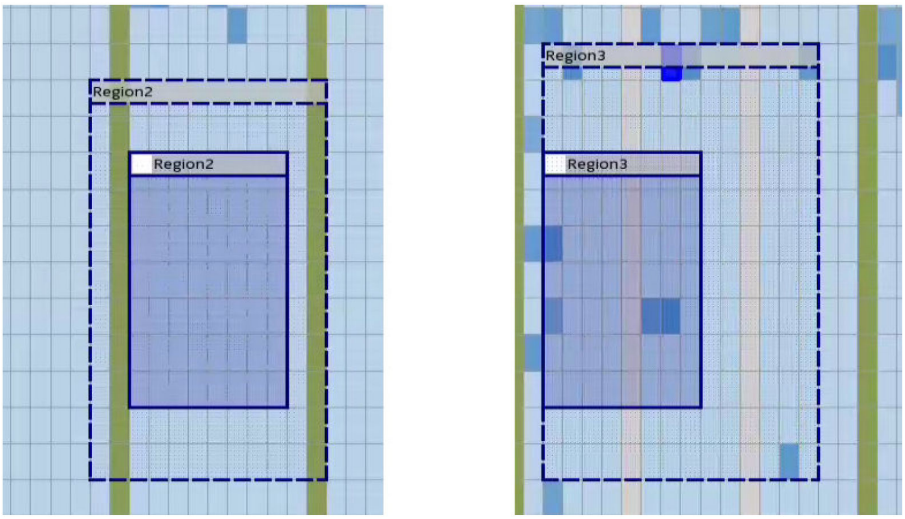
Figure 116. New Logic Lock Region In Logic Lock Regions Window



6.3.2.3. Defining Routing Regions

A routing region is an element of a Logic Lock region that specifies the routing area. A routing region must encompass the existing Logic Lock placement region. Routing regions cannot be set as reserved. To define the routing region, double-click the **Routing Region** cell in the **Logic Lock Regions** window, and select an option from the drop-down menu.

Figure 117. Routing Regions



Fixed with Expansion +2

Custom Routing

Table 43. Routing Region Options

Option	Description
Unconstrained (default)	Allows the Fitter to use any available routes on the device.
Whole Chip	Same result as **Unconstrained**, but writes the constraint in the Quartus Prime settings file (`.qsf`).
Fixed with Expansion	Follows the outline of the placement region. The routing region scales by a number of rows and columns larger than the placement region.
Custom	Allows you to define a custom shape routing region around the Logic Lock region. When you select the **Custom** option, the placement and routing regions move independently in the Chip Planner. In this case, move the placement and routing regions by selecting both using the Shift key.

6.3.2.4. Defining Empty Logic Lock Regions

The Quartus Prime supports the use of Logic Lock regions without any members. You can use such empty regions to reserve device resources to contain logic to be add later. This technique require that you turn on the region's **Reserved** setting to prevent the Fitting from placing any other logic within this region.

Empty Logic Lock regions can be useful for the following scenarios:

- Preliminary floorplanning
- Complex incremental builds, such as root partition reuse
- Team based design and interconnect logic
- Confining logic placements

Since Logic Lock regions do not reserve any routing resources by default, the Fitter may use the reserved area for routing purposes.

Use the **Core Only** attribute for empty Logic Lock regions. When you include periphery resources in empty regions, you restrict the periphery component placement, which can result in a no fit design. After you name the empty region, you can perform the same manipulations as with any Logic Lock region that includes members.

Figure 118. All Logic Placed Outside an Empty region

The figure shows an empty Logic Lock region and the logic placed around it. However, some I/Os, HSSIO, and PLLs are present in the empty region because the output port connects to the I/O that is part of the root_partition (top-level partition).

6.3.2.5. Defining Hierarchical Logic Lock Regions

Logic Lock regions can be fully hierarchical. Parent regions must completely contain all child regions. The **Reserved** and **Core-Only** assignments also apply hierarchically.

Logic Lock assignments follow the same precedence as other constraints and assignments.

You can assign an entity in the design to only one Logic Lock region, but the entity can inherit regions by hierarchy. This hierarchy allows a reserved region to have a sub region without reserving the resources in the sub region.

6.3.3. Customizing the Shape of Logic Lock Regions

To create custom shaped Logic Lock regions, you can perform logic operations. Non-rectangular Logic Lock regions can help you exclude certain resources, or place parts of your design around specific device resources to improve performance.

Attention: There is no undo feature for the Logic Lock shapes for 17.1.

Logic Lock Regions Properties Dialog Box

Use the **Logic Lock Regions Properties** dialog box to view and modify detailed information about your Logic Lock region, such as which entities and nodes are assigned to your region, and which resources are required.

To open the **Logic Lock Regions Properties** dialog box, right-click the region and select **Logic Lock Regions Properties...**.

6.3.3.1. Adding a New Shape to a Logic Lock Region

To add a new shape to an existing Logic Lock region, perform the following steps in the Chip Planner:

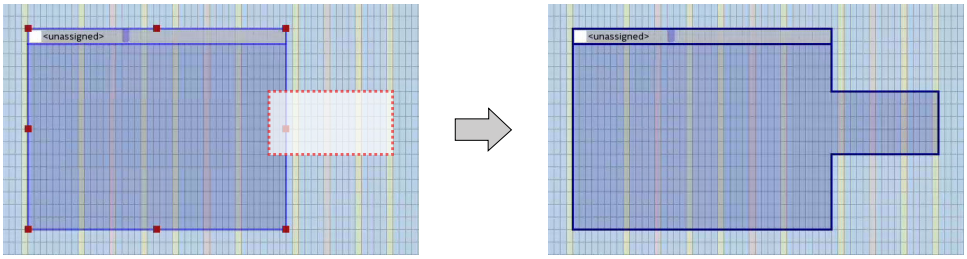
1. Select the Logic Lock region.
2. In the **Navigation** toolbar, click the **Add Logic Lock Region** button.

Figure 119. Add Logic Lock Region Button in Toolbar

3. Click and drag to generate the shape you want to add. The new shape merges automatically with the selected Logic Lock region.

   Attention: If you selected more than one region, the operation appends the new shape to all of the regions.

Figure 120. Using the Add Logic Lock Region Feature

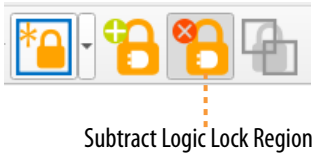


6.3.3.2. Subtracting Shape from Logic Lock Region

To subtract a shape from an existing Logic Lock region, perform the following steps in the Chip Planner:

1. Select the Logic Lock region.
2. In the **Navigation** toolbar, click the **Subtract Logic Lock Region** button.

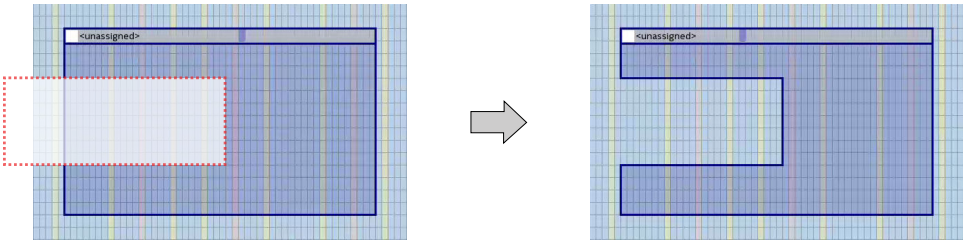
Figure 121. Subtract Logic Lock Region Toolbar Button



3. Click and drag the shape you want to subtract. The modified region displays automatically.

   The operation performs in all selected regions.

Figure 122. Using the Subtract Logic Lock Region Feature



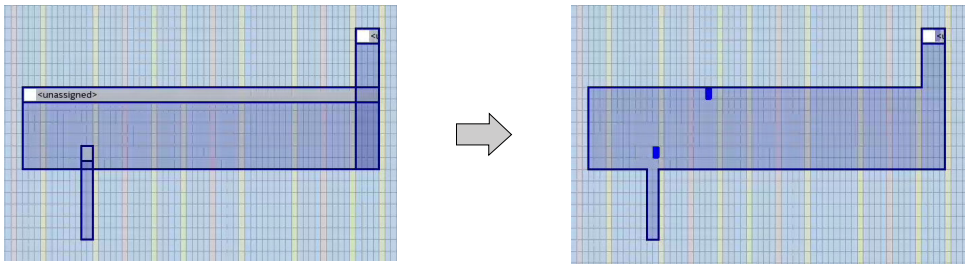
6.3.3.3. Merging Logic Lock Regions

To merge two or more Logic Lock regions, perform the following steps:

1. Ensure that no more than one of the regions that you intend to merge has logic assignments.
2. Arrange the regions into the locations where you want the resultant region.
3. Select all the individual regions that you want to merge by clicking each of them while pressing the Shift key.
4. Right-click the title bar of any of the selected Logic Lock regions and select **Logic Lock Regions ➤ Merge Logic Lock Region**. The individual regions that you select merge to create a single new region.

If you select multiple named regions, the **Merge Logic Lock Region** option is deactivated.

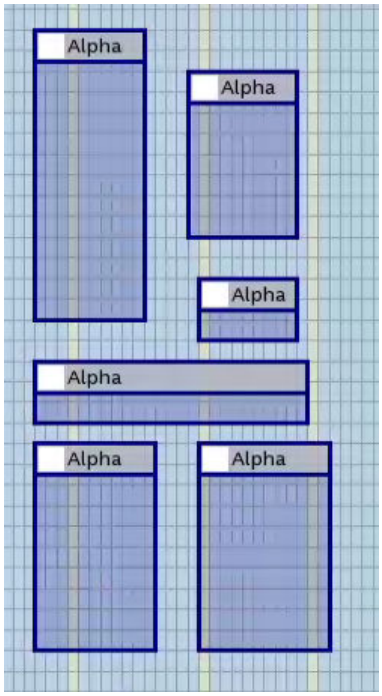
Figure 123. Using the Merge Logic Lock Region command



6.3.3.4. Defining Noncontiguous Logic Lock Regions

You can create disjointed regions by using the Logic Lock region manipulation tools. Noncontiguous regions act as a single Logic Lock region for all Logic Lock region attributes.

Figure 124. Logic Lock Regions in Chip Planner Floorplan



6.3.4. Assigning Device Pins to Logic Lock Regions

A Logic Lock region incorporates all device resources within its boundaries, including memory and pins. The Quartus Prime Pro Edition software does not include pins automatically when you assign an entity to a region, unless the **Core Only** attribute is off.

You can manually assign pins to Logic Lock regions; however, this placement puts location constraints on the region. The software only obeys pin assignments to locked regions that border the periphery of the device. The locked regions must include the I/O pins as resources.

6.3.5. Viewing Connections Between Logic Lock Regions in Chip Planner

You can view and edit Logic Lock regions using the Chip Planner. To view and edit Logic Lock regions, use **Floorplan Editing** in the **Layers Settings** window, or any layers setting mode that has the **User-assigned Logic Lock regions** setting enabled.

The Chip Planner shows the connections between Logic Lock regions. By default, you can view each connection as an individual line. You can choose to display connections between two Logic Lock regions as a single bundled connection rather than as individual connection lines. To use this option, open the Chip Planner and on the View menu, click **Inter-region Bundles**.

Related Information

Inter-region Bundles Dialog Box
For more information about the Inter-region Bundles dialog box, refer to Quartus Prime Help.

6.3.6. Example: Placement Best Practices for Arria 10 FPGAs

Logic Lock regions must take into account the device topology.

This example describes how I/O Columns constrain locations in Logic Lock regions in designs targeting Arria 10 FPGAs.

Figure 125. I/O Columns in Arria 10 FPGAs

Arria 10 FPGAs have I/O columns located in the middle of the device. Signals can only enter or exit these columns from the side that faces the device edge.

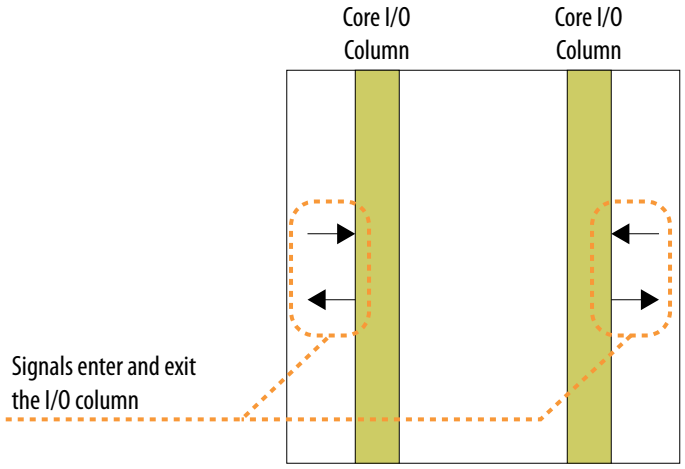


Figure 126. Signals Crossing I/O Columns in Arria 10 FPGAs

Routing a signal to cross the I/O column increases the routing delay, and can reduce design performance.

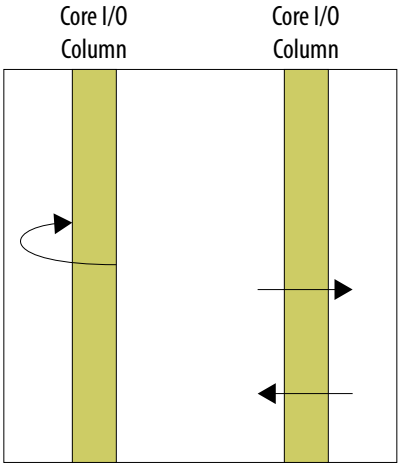
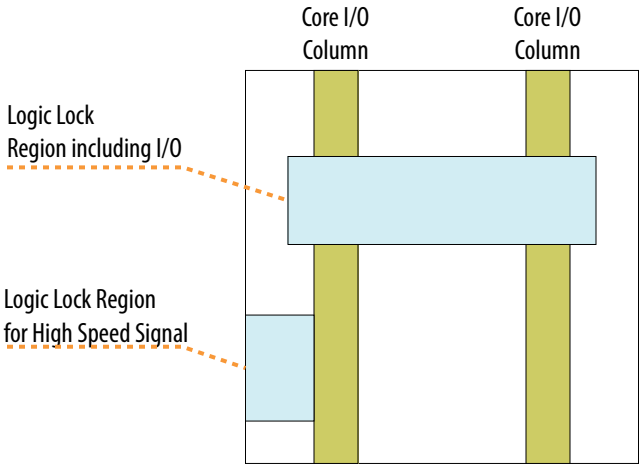


Figure 127. Strategic Placement for Logic Lock Regions in Arria 10 FPGAs

- If a Logic Lock region contains a register that interface with the I/O column, place the Logic Lock region so that the region covers the I/O column and the core logic, for better access to the I/O column adjacent to the outer column edge.
- For high speed signal, you can get best results if you place the Logic Lock region on the outside of the I/O column, because the fitter is less likely to cross the column and incur delay.



6.3.7. Migrating Assignments between Quartus Prime Standard Edition and Quartus Prime Pro Edition

The Quartus Prime Pro Edition software does not support the Quartus Prime Standard Edition Logic Lock (Standard) assignments. Therefore, if you are migrating a design from Quartus Prime Standard Edition to Quartus Prime Pro Edition, you must convert the Logic Lock (Standard) assignments into Logic Lock assignments.

Related Information

Replace Logic Lock Regions
In *Quartus Prime Pro Edition User Guide: Getting Started*

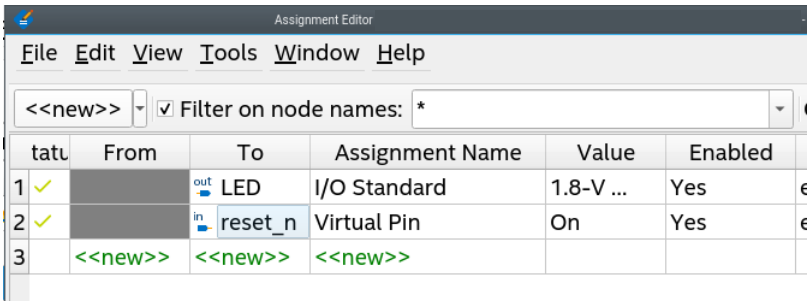
6.4. Defining Virtual Pins

A virtual pin is an I/O element that the Compiler temporarily maps to a logic element, rather than to a pin. The Compiler implements these virtual pins as LUTs.

By making assignments to virtual pins, you can ensure that the Fitter places those pins in the same device region as the corresponding internal nodes in the top-level module. You can apply the **Virtual Pin** option to successfully compile a Logic Lock module that has more pins than the target device. The **Virtual Pin** option can enable timing analysis of a design module that more closely matches the performance of the module after you integrate it into the top-level design.

You can create and assign virtual pins to an I/O element using the **Virtual Pin** logic option in the Assignment Editor (**Assignments ➤ Assignment Editor**).

Figure 128. Virtual Pin Logic Option in the Assignment Editor



When you apply the **Virtual Pin** assignment to an input pin, the pin no longer appears as an FPGA pin. Rather, the Compiler fixes the virtual pin to GND in the design. The virtual pin is not a floating node.

Use virtual pins only for I/O elements in lower-level design entities that become nodes after you import the entity to the top-level design; for example, when compiling a partial design. In the top-level design, you connect these virtual pins to an internal node of another module

Note: You must assign the **Virtual Pin** logic option to an input or output pin. If you assign this option to a bidirectional pin, tri-state pin, or registered I/O element, synthesis ignores the assignment. If you assign this option to a tri-state pin, the Fitter inserts an I/O buffer to account for the tri-state logic; therefore, the pin cannot be a virtual pin. You can use multiplexer logic instead of a tri-state pin if you want to continue to use the assigned pin as a virtual pin. Do not use tri-state logic except for signals that connect directly to device I/O pins.

To display all assigned virtual pins in the design with the Node Finder, you can set **Filter Type** to **Pins: Virtual**. To access the Node Finder from the Assignment Editor, double-click the **To** field; when the arrow appears on the right side of the field, click and select **Node Finder**.

Related Information

Assigning Virtual Pins with a Tcl command on page 190

6.5. Using Logic Lock Regions in Combination with Design Partitions

You can optimize timing in a design by placing entities that share significant logical connectivity close to each other on the device.

By default, the Fitter attempts to place closely connected entities in the same area of the device. However, without constraint, this same placement is not assured for each compilation. You can use Logic Lock regions, together with design partitions, to ensure that logically connected entities retain optimal placement from one compilation to the next.

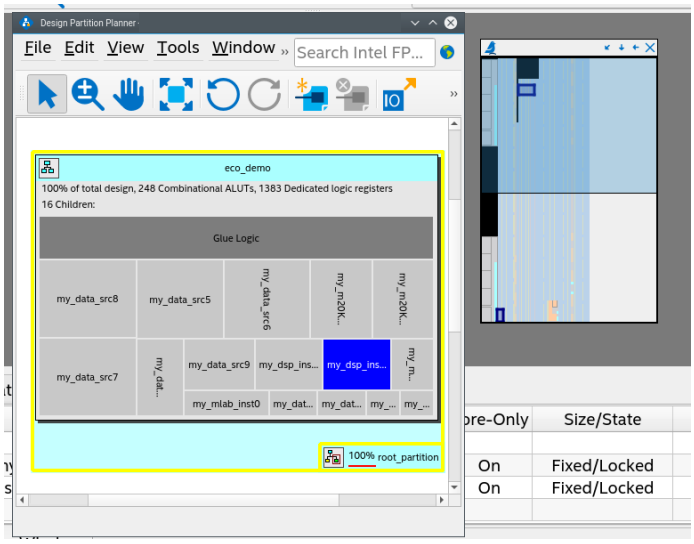
Using Logic Lock regions in combination with design partitions allows you to preserve the location and performance of a block, so that the Fitter focuses time and effort on other portions of the design.

Note: For more details about these techniques, refer to *Quartus Prime Pro Edition User Guide: Block-Based Design*

To use the Design Partition Planner in conjunction with the Chip Planner to readily create partitions and define Logic Lock regions, follow these steps:

1. On the Compilation Dashboard, double-click **Plan** to compile through that Fitter stage, or run a full compilation.
2. Open the Chip Planner and the Design Partition Planner:
   - Click **Tools ➤ Chip Planner**
   - Click **Tools ➤ Design Partition Planner**
3. In the Chip Planner, double-click **Report Design Partitions** in the **Tasks** pane. The Chip Planner displays the physical locations of design partitions using the same colors as the entities in the Design Partition Planner.

Figure 129. Design Partition Planner Overlaying Chip Planner



4. In the Chip Planner, click **View ➤ Bird's Eye View**

5. In the Design Partition Planner, drag all the larger entities out from their parents. Alternatively, you can right-click the entity and click **Extract from Parent**. The Chip Planner displays the physical placement of the entities shown in the Design Partition Planner, with consistent colors between the two tools. You can view physical placement in the Chip Planner and connectivity in the Design Partition Planner.
6. Identify entities that are unsuitable to place in Logic Lock regions:
   - The Chip Planner shows an entity to be physically dispersed over noncontiguous areas of the device.
   - The Design Partition Planner shows an entity to have a large number of connections to other entities.
7. Drag the entities that are unsuitable for placement in Logic Lock regions back to the parent entities. Alternatively, right-click the entity and click **Collapse to Parent**.
8. Create a partition for each remaining entity by right-clicking the entity, and then clicking **Create Design Partition**.
9. Create a Logic Lock region for each partition by right-clicking the partition, and then clicking **Create Logic Lock Region**.

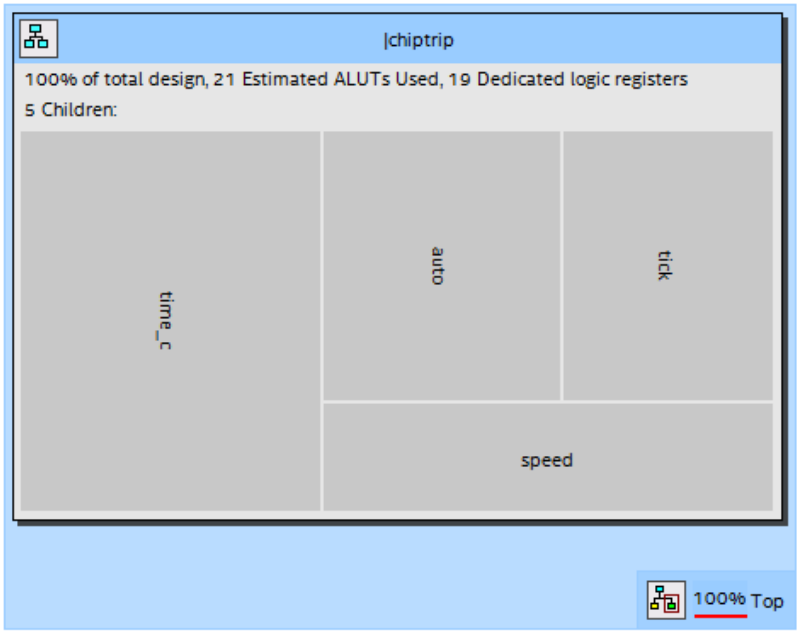
Related Information

- Quartus Prime Pro Edition User Guide: Block-Based Design
- Quartus Prime Pro Edition User Guide: Partial Reconfiguration

6.5.1. Viewing Design Connectivity and Hierarchy

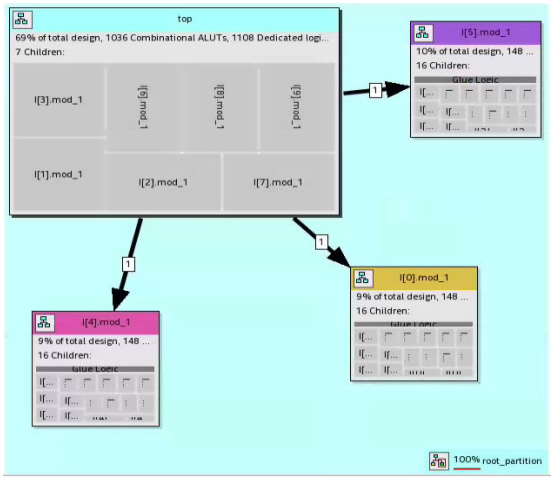
By default, when you open a compiled design, the Design Partition Planner displays the design as a single top-level entity, containing lower-level entities. If the Design Partition Planner has opened the design previously, the design appears in its last state.

Figure 130. Top-Level Entity in the Design Partition Planner



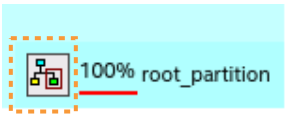
- To show connectivity between entities, extract entities from the top-level entity by dragging them into the surrounding white space, or by right-clicking an entity and clicking **Extract from Parent** on the shortcut menu.
  When you extract entities, Design Partition Planner draws the connection bundles between entities, showing the number of connections between pairs of entities.

Figure 131. Partitioned Design with Connection Bundles



- To customize the appearance of connection bundles or to set thresholds for connection counts, click **View ➤ Bundle Configuration**, and set the necessary options in the **Bundle Configuration** dialog box.
- To see bundles containing failing paths, open the Timing Analyzer, and then click **View ➤ Show Timing Data** in the Design Partition Planner. Bundles containing failing paths are displayed in red, as are entities having nodes that reside on failing paths.
- To see detailed information about the connections in a bundle, right-click the bundle, and then click **Bundle Properties** to open the **Bundle Properties** dialog box.
- To switch between connectivity display mode and hierarchical display mode, click **View ➤ Hierarchy Display**. Alternatively, click and hold the hierarchy button in the top-left corner of any entity to switch temporarily to a hierarchy display.

Figure 132. Hierarchy Display Button



6.6. Creating Clock Region Assignments in Chip Planner

You can easily create and manipulate clock regions in the Chip Planner and make clock assignments to the regions.

You can create a user-defined clock region assignment to ensure that a given global clock signal is available to resources in a certain area of the device throughout all design iterations. In instances of congestion involving global signal resources, you may specify a smaller clock region assignment to prevent a signal from using congested clock resources in other sectors.

If you create user-defined clock regions and subsequently compile the design, those user-defined clock regions become Fitter-defined clock regions, and are read-only.

Summary of User-Defined Clock Region Feature Support

Feature	Clock Region Support
Shapes of clock regions	Limited to rectangular regions that snap to clock sector grids.
Peripheral element assignments	Limited to clocking design elements.
Clock region name	Identified by the source clocking design element.
Support for multiple instances per region	Create one region per clock design element, and then specify the same definition for multiple clock design elements to assign to the same clock region.

Using Clock Region Assignments in Stratix 10 and Agilex 7 Devices

You can constrain clock regions to a rectangle whose dimensions are defined by the sector grid, as seen in the **Clock Sector Region** layer of the Chip Planner. The rectangle is defined by the coordinates of its bottom-left and top-right corners. For example, *SX0, SY0, SX1, SY1* constrains the clock to a 2 × 2 region, from the bottom left of sector 0,0 to the top right of sector 1,1.

You can alternatively specify the bounding rectangle in chip coordinates, for example *X37 Y181 X273 Y324*. However, you should sector-align such a constraint. The Fitter automatically snaps to the smallest sector-aligned rectangle that encompasses the original assignment.

6.6.1. Creating Clock Assignments in Chip Planner

To create clock assignments with the Chip Planner, follow these steps:

1. Select the **Create Clock Assignment** button, or click **View ➤ Clock Assignments ➤ Create Clock Assignment**.

Figure 133. Create Clock Assignment Button on Chip Planner Toolbar

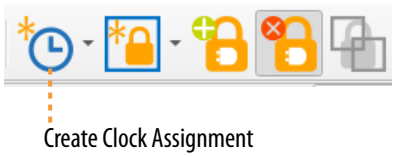
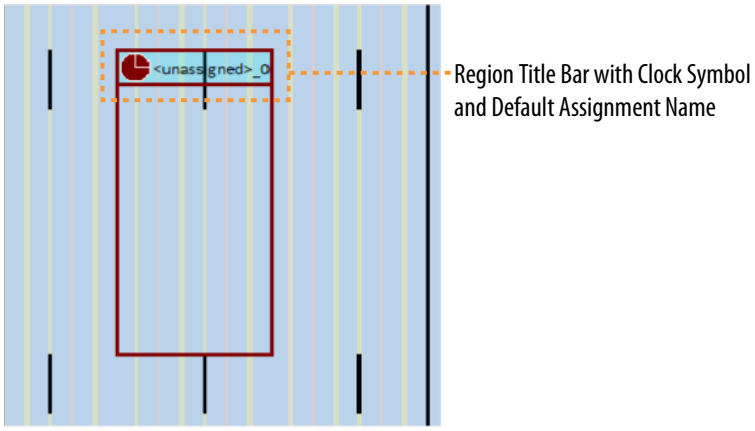


Figure 134. Newly Created Clock Region



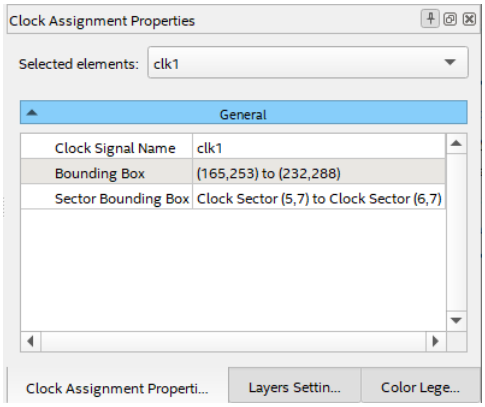
2. Click and drag the mouse on the Chip Planner floorplan to draw a clock region of your preferred location and size. The region that you draw snaps to the smallest clock sector capable of containing the region. Orange clock sector grids help visualize positioning of the clock region relative to clock sectors.
3. Assign a clock signal from the context menu. A clock symbol in the region title bar identifies a clock region. The clock region is unassigned until you assign a clock signal from the context menu.

6.6.1.1. Clock Assignment Properties

The **Clock Assignment Properties** pane displays properties of the selected clock assignment.

By default, the **Clock Assignment Properties** pane appears on a tab at the right side of the Chip Planner.

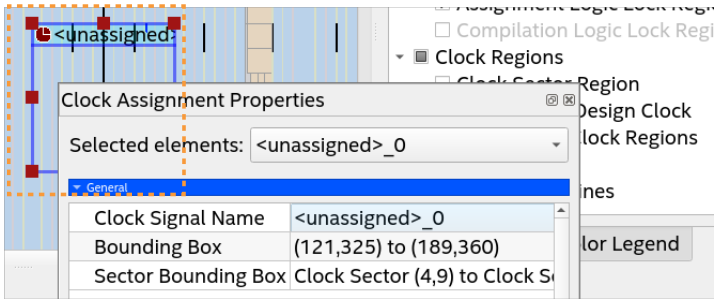
Figure 135. Clock Assignment Properties Pane



6.6.2. Resizing a Clock Assignment in Chip Planner

To resize an existing clock assignment in the Chip Planner, follow these steps:

Figure 136. Clock Assignment in Chip Planner



1. Select an existing clock assignment in the Chip Planner floorplan. Handles appear on each side of the region and at the corners.
2. Position the crosshairs over the handle of your choice and the resize mouse cursor appears.
3. Hold the left mouse button and drag the resize cursor to expand or shrink the boundary of the clock assignment. When you release the mouse button, the clock assignment boundaries snap to the nearest containing clock sector grid.

6.6.3. Moving a Clock Assignment in Chip Planner

To move a clock assignment in the Chip Planner, follow these steps:

1. Select the clock assignment in the Chip Planner floorplan.
2. Position the crosshairs over the clock assignment title bar and the move cursor appears.
3. Hold the left mouse button and drag the clock assignment to the new location.

6.6.4. Deleting a Clock Region Assignment in Chip Planner

To delete a clock region assignment in the Chip Planner, follow these steps:

1. Select the clock assignment that you want to delete in the Chip Planner floorplan.
2. Right-click the clock assignment title bar to display the context menu, or select **View** from the main menu bar.
3. Click **Clock Assignments ➤ Delete Clock Assignment**.
4. You are prompted to confirm that you want to delete the selected clock assignment. Click **Yes** to confirm the deletion.

The specified clock region assignment is deleted from the system.

6.6.5. Assigning a Clock Signal to a Clock Region in Chip Planner

To assign a clock signal to a clock region in the Chip Planner, follow these steps:

1. Right-click the clock assignment title bar to display the context menu, or select **View** from the main menu bar in the Chip Planner.
2. Click **Clock Assignments ➤ Set Clock Signal Name**.
3. In the **Set Clock Signal Name** dialog box, browse to or type the desired clock signal name.
4. Click **Ok**.

The system renames the clock assignment according to the name of the specified clock signal.

6.7. Scripting Support

You can run the commands and specify the settings described in this chapter as part of a Tcl script. You can also run some commands at a command prompt. The following topics describe the commands.

Related Information

Quartus Prime Pro Edition User Guide: Scripting

6.7.1. Creating Logic Lock Assignments with Tcl commands

The Quartus Prime software supports Tcl commands to create or modify Logic Lock assignments.

Note: Specify node names by using the full hierarchy path to the node.

Create or Modify a Placement Region

You can create the Logic Lock region from the GUI, or add the region directly to the QSF. The QSF entry contains the X/Y coordinates of the vertices and the Placement Region name.

The following assignment creates a new placement region with bounding box coordinates `X46 Y36 X65 Y49`:

```
set_instance_assignment -name PLACE_REGION "X46 Y36 X65 Y49" -to <node names>
```

- You can use the same command format to modify an existing assignment.
- To specify a non-rectangular or disjoint region, use a semicolon (`;`) as the delimiter between two or more bounding boxes.
- Assign multiple instances to the same region with multiple `PLACE_REGION` instance assignments.

Create or Modify a Routing Region

The following assignment creates a routing region with bounding box coordinates `X5 Y5 X30 Y30`:

```
set_instance_assignment -name ROUTE_REGION -to <node names> "X5 Y5 X30 Y30"
```

- You can use the same command format to modify an existing assignment.
- All instances with a routing region assignment must have a respective placement region; the routing region must fully contain the placement region.

Specify a Region as Reserved

The following assignment reserves an existing region:

```
set_instance_assignment -name <instance name> RESERVE_PLACE_REGION -to <node
names> ON
```

- You can only reserve placement regions.

Specify a Region as Core Only

By default, the Quartus Prime Pro Edition software includes pins in Logic Lock assignments. To specify a region as core only (that is, periphery logic in the instance that is not constrained), use the following assignment:

```
set_instance_assignment -name <instance name> CORE_ONLY_PLACE_REGION -to <node names> ON
```

Related Information

Defining Logic Lock Regions on page 167

6.7.2. Assigning Virtual Pins with a Tcl command

Use the following Tcl command to turn on the virtual pin setting for a pin called `my_pin`:

```
set_instance_assignment -name VIRTUAL_PIN ON -to my_pin
```

Related Information

Defining Virtual Pins on page 180

6.7.3. Logic Lock Region Assignment Examples

The following examples show the syntax of Logic Lock region assignments in the `.qsf` file. Optionally, you can enter these assignments in the Assignment Editor, the Logic Lock Regions Window, or the Chip Planner.

Example 1. Assign Rectangular Logic Lock Region

Assigns a rectangular Logic Lock region to a lower left corner location of (10,10), and an upper right corner of (20,20) inclusive.

```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
```

Example 2. Assign Non-Rectangular Logic Lock Region

Assigns instance with full hierarchical path "x|y|z" to non-rectangular L-shaped Logic Lock region. The software treats each set of four numbers as a new box.

```
set_instance_assignment –name PLACE_REGION –to x|y|z "X10 Y10 X20 Y50; X20 Y10 X50 Y20"
```

Example 3. Assign Subordinate Logic Lock Instances

By default, the Quartus Prime software constrains every child instance to the Logic Lock region of its parent. Any constraint to a child instance intersects with the constraint of its ancestors. For example, in the following example, all logic beneath "a|b|c|d" constrains to box (10,10), (15,15), and not (0,0), (15,15). This result occurs because the child constraint intersects with the parent constraint.

```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
set_instance_assignment –name PLACE_REGION –to a|b|c|d "X0 Y0 X15 Y15"
```

Example 4. Assign Multiple Logic Lock Instances

By default, a Logic Lock region constraint allows logic from other instances to share the same region. These assignments place instance c and instance g in the same location. This strategy is useful if instance c and instance g are heavily interacting.

```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
set_instance_assignment –name PLACE_REGION –to e|f|g "X10 Y10 X20 Y20"
```

Example 5. Assigned Reserved Logic Lock Regions

Optionally reserve an entire Logic Lock region for one instance and any of its subordinate instances.

```
set_instance_assignment –name PLACE_REGION –to a|b|c "X10 Y10 X20 Y20"
set_instance_assignment –name RESERVE_PLACE_REGION –to a|b|c ON

# The following assignment causes an error. The logic in e|f|g is not
# legally placeable anywhere:
# set_instance_assignment –name PLACE_REGION –to e|f|g "X10 Y10 X20 Y20"

# The following assignment does *not* cause an error, but is effectively
# constrained to the box (20,10), (30,20), since the (10,10),(20,20) box is
reserved
# for a|b|c
set_instance_assignment –name PLACE_REGION –to e|f|g "X10 Y10 X30 Y20"
```

6.8. Analyzing and Optimizing the Design Floorplan Revision History

The following revision history applies to this chapter:

Table 44. Document Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2023.12.04	23.4	• Added new *Location Assignment Optimization Guidelines* topic.
2023.08.01	23.2	• Replaced missing graphics in *Analyzing Connections for a Path*. • Replaced missing graphics in *Navigating with the Bird's Eye View*. • Replaced missing graphics in *Viewing Immediate Fan-In and Fan-Out in Chip Planner*. • Replaced missing graphics in *Show Delays*. • Replaced missing graphics in *Starting the Chip Planner*. • Replaced missing graphics in *Adding a New Shape to a Logic Lock Region*. • Replaced missing graphics in *Creating Clock Assignments in Chip Planner*. • Replaced missing graphics in *Viewing Fan-In and Fan-Out in Chip Planner*. • Replaced missing graphics in *Viewing Design Connectivity and Hierarchy*. • Replaced missing graphics in *Subtracting Shape from Logic Lock Region*.
2023.04.03	23.1	• Updated product family name to "Intel Agilex 7."

continued...

Document Version	Quartus Prime Version	Changes
2022.09.26	22.3	• Updated chapter for new Compilation Regions tab in the Logic Lock Regions window. This read-only tab displays the properties of any Logic Lock regions contained in a .qdb file in the current project.
2022.01.07	21.4	• Added Properties tab information to *Viewing Architecture-Specific Design Information* topic. • Added Design Assistant information to *Using Logic Lock Regions in the Chip Planner* topic. • Added new *Viewing the Location and Utilization of Device Resources in Chip Planner* topic. • Added new *Viewing Module Placement by Cross-Probing to Chip Planner* topic. • Added new *Finding Design Elements in the Chip Planner* topic. • Added new *Find In Options* topic.
2019.07.30	19.3.0	Added new *Using User-Defined Clock Regions in the Chip Planner* section.
2019.07.01	19.1.0	Added new "Snapping to a Region" topic that describes the **Snap Logic Lock Region to** option.
2019.04.01	19.1.0	• Added new "Viewing Selected Contents" topic that describes a new report listing selected design elements.
2018.09.24	18.1.0	• Added topic: *Viewing Clock Sector Utilization* • Added topic: *Viewing the Source and Destination of Placed Nodes*. • Renamed topic: *Generating Fan-In and Fan-Out Connections* to *Viewing Fan-In and Fan-Out Connections of Placed Resources*.
2018.05.07	18.0.0	• Added recommendations for using iterative methods for floorplanning.
2017.11.06	17.1.0	• Changed instances of *LogicLock Plus* to *Logic Lock*. • Added support for auto-sized Logic Lock regions. • Added support for empty Logic Lock regions. • Added topics: Considerations on Using Auto Sized Regions, Creating Partitions and Logic Lock Regions with the Design Partition Planner and Chip Planner.
2017.05.08	17.0.0	• Chapter reorganization and content update. • Added figures: Clock Regions, Path List in the Locate History Window, Show Physical Routing, Using the Add Rectangle Feature, Using the Subtract Rectangle Feature, Creating a Hole in a LogicLock Region, Noncontiguous LogicLock Region, Routing Regions, Logic Placed Outside of an Empty Region. • Updated figures: HSSI Channel Blocks, Highlight Routing, High-Speed and Low Power Tiles in an Arria 10 Device, Show Delays Highlight Routing, Viewing Assignments in the Chip Planner, LogicLock Plus Regions Window, Using the Merge LogicLock Plus Region Command. • Created topics: *Adding Rectangle to a LogicLock Plus Region*, *Subtracting Rectangle from a LogicLock Plus Region*. • Moved topic: *Viewing Critical Paths* to *Timing Closure and Optimization* chapter and renamed to *Critical Paths*. • Renamed topic: *Creating Non-Rectangular LogicLock Plus Regions* to *Merging LogicLock Plus Regions*. • Renamed topic: *Chip Planner Overview* to *Design Floorplan Analysis in the Chip Planner*. • Renamed chapter from *Analyzing and Optimizing the Design Floorplan with the Chip Planner* to *Analyzing and Optimizing the Design Floorplan*.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Added topic describing how to create a hole in a LogicLock Plus region.
2016.05.02	16.0.0	Updated information on creating LogicLock Plus regions.

continued...

Document Version	Quartus Prime Version	Changes
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Added information on how to use LogicLock regions.
2015.05.04	15.0.0	Added information about color coding of LogicLock regions.
2014.12.15	14.1.0	Updated description of Virtual Pins assignment to clarify that assigned input is not available.
June 2014	14.0.0	Updated format
November 2013	13.1.0	Removed HardCopy device information.
May 2013	13.0.0	Updated "Viewing Routing Congestion" section Updated references to Quartus UI controls for the Chip Planner
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	• Updated for the 11.0 release. Edited "LogicLock Regions" Updated "Viewing Routing Congestion" Updated "Locate History" Updated Figures 15-4, 15-9, 15-10, and 15-13 Added Figure 15-6
December 2010	10.1.0	• Updated for the 10.1 release.
July 2010	10.0.0	• Updated device support information • Removed references to Timing Closure Floorplan; removed "Design Analysis Using the Timing Closure Floorplan" section • Added links to online Help topics • Added "Using LogicLock Regions with the Design Partition Planner" section • Updated "Viewing Critical Paths" section • Updated several graphics • Updated format of Document revision History table
November 2009	9.1.0	• Updated supported device information throughout • Removed deprecated sections related to the Timing Closure Floorplan for older device families. (For information on using the Timing Closure Floorplan with older device families, refer to previous versions of the Quartus Prime Handbook, available in the Documentation Archive.) • Updated "Creating Nonrectangular LogicLock Regions" section • Added "Selected Elements Window" section • Updated table 12-1
May 2008	8.0.0	• Updated the following sections: "Chip Planner Tasks and Layers" "LogicLock Regions" "Back-Annotating LogicLock Regions" "LogicLock Regions in the Timing Closure Floorplan" • Added the following sections: "Reserve LogicLock Region" "Creating Nonrectangular LogicLock Regions" "Viewing Available Clock Networks in the Device" • Updated Table 10–1 • Removed the following sections: Reserve LogicLock Region Design Analysis Using the Timing Closure Floorplan

7. Using the ECO Compilation Flow

In a typical FPGA project development cycle, the specification of the programmable logic portion of the design can change during the design process. The Quartus Prime software supports these last-minute, targeted *engineering change orders* (ECOs), even after full compilation is complete.

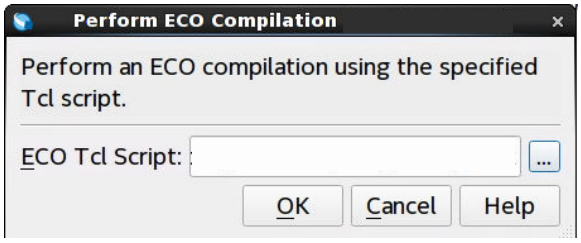
ECOs typically occur during the design verification stage. For example, during verification you may determine that the design requires a small change, such as a netlist connection change, correcting a LUT logic error, or placing a node in a new location. Implementing an ECO change, rather than changing RTL and fully recompiling the design, requires significantly less time, and changes only the affected logic.

You specify the ECO commands in a Tcl script using the `::quartus::eco` package.

Note: The Quartus Prime Pro Edition software supports ECOs for Stratix 10 and Agilex 7 devices only.

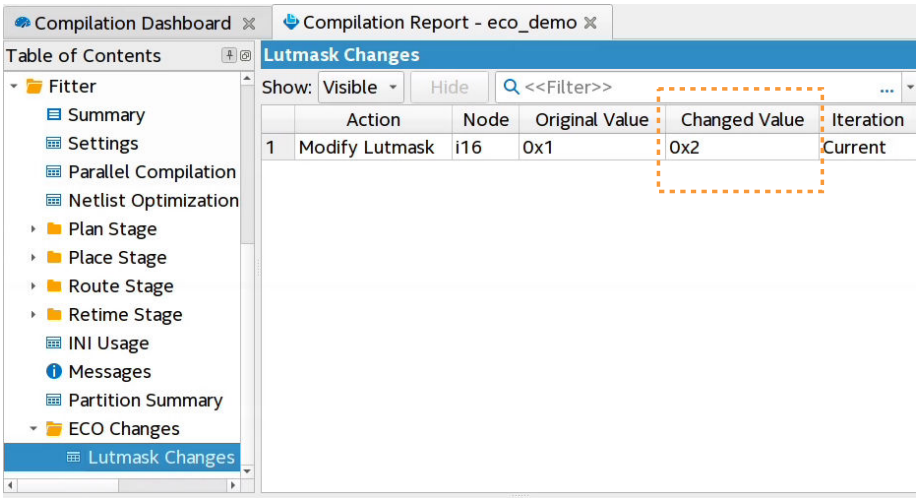
7.1. ECO Compilation Flow

1. Identify an ECO modification you want to make in a compiled design.
2. Determine if ECO commands support the change, by reviewing ECO Commands on page 197 and ECO Command Limitations on page 205.
3. Create a Tcl script, as ECO Tcl Script Example on page 195 shows.
4. Before running ECO compilation, click **Project ➤ Archive Project** and archive the compilation database and output file set.
5. Click **Processing ➤ Start ➤ Perform ECO Compilation**.



6. Specify the **ECO Tcl Script** file, and click **OK**. The Fitter processes the ECO commands and updates the finalized netlist. The Fitter generates an error if you specify any commands incorrectly. The changes apply when the Fitter processing completes.
7. View the ECO results in post-fit analysis tools, such as the Compilation Report, Timing Analyzer, Netlist Viewer, or Chip Planner. To view ECO changes in the Fitter report, click **Processing ➤ Compilation Report ➤ Fitter ➤ ECO Changes**.

Figure 137. Example of ECO Changes Report



As an alternative to the GUI methods, you can use the following commands to run the ECO Tcl scripts. If running from command line, any active Quartus Prime GUI application does not refresh. Close and reopen the project to refresh the GUI.

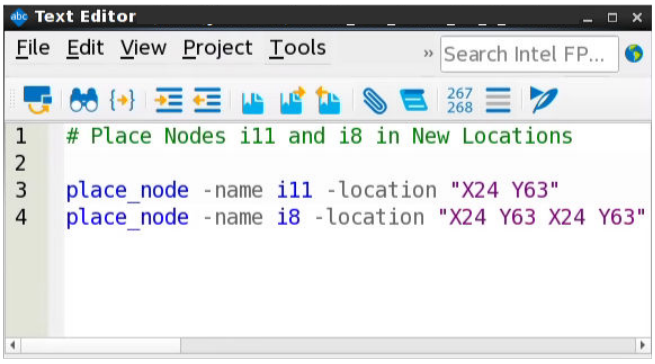
```
$ quartus_fit -s
load_package eco
project_open <project_name>
eco_load_design
...
eco_commit_design
project_close
```

Note: If you rerun the Fitter on a design after implementing an ECO, the Fitter overwrites the ECO changes. Update RTL, IP parameters, and recompile the design to permanently implement the ECO changes.

7.2. ECO Tcl Script Example

The following shows an example ECO Tcl script that places existing nodes in new locations:

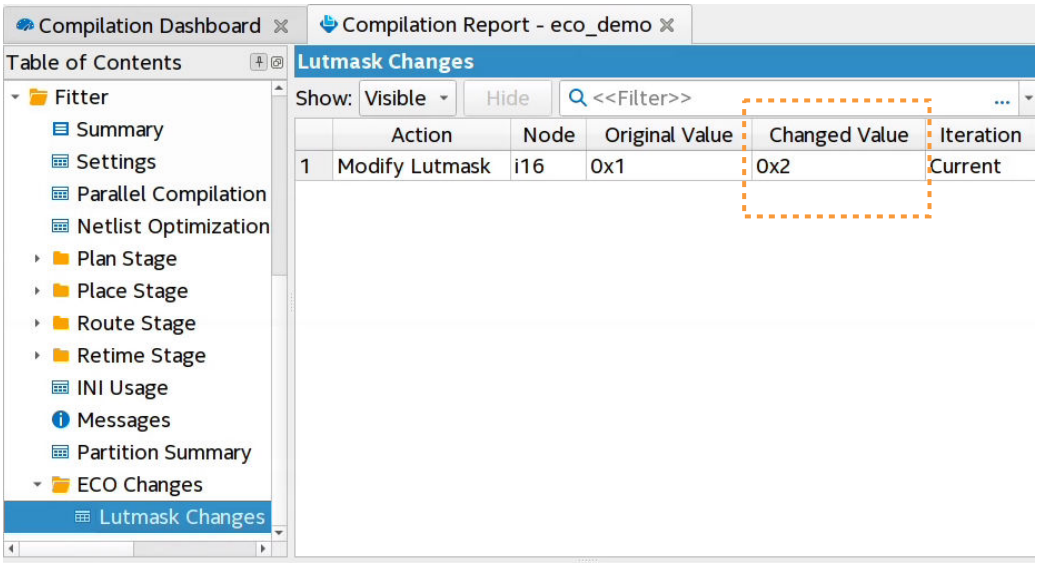
Figure 138. ECO Tcl Script Example



7.3. Viewing ECO Compilation Reports

The Compiler generates a report showing the details of each ECO compilation that you run successfully. You can view the report contents in the **ECO Changes** report under **Fitter** in the Compilation Report.

Figure 139. Example of ECO Changes Report



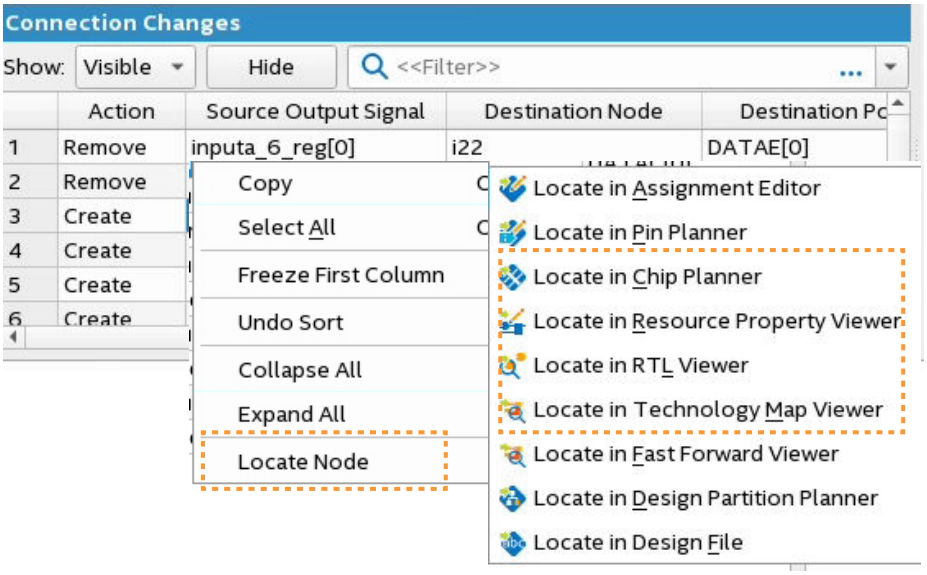
Alternatively, you can view this data in the generated `fit.eco` file. The Compiler organizes the report output according to the category of ECO change, such as Placement Changes. The table specifies the "Changes in Previous ECO Runs" and "Changes in Current ECO Run".

Figure 140. ECO Report Example

```
+-----------------------------------------+
; Placement Changes                       ;
+------------------+----------+-----------+
; Name             ; Location ; Iteration ;
+------------------+----------+-----------+
; GND~ECO_INSERTED ; X78_Y72  ; Previous  ;
; new_node         ; X78_Y72  ; Previous  ;
; new_wirelut      ; X78_Y72  ; Previous  ;
; new_wirelut      ; X24_Y24  ; Current   ;
+------------------+----------+-----------+
```

Locate nodes from the Fitter's ECO reports to confirm ECO changes.

Figure 141. Locate Node from Fitter ECO Reports



7.4. ECO Commands

The Quartus Prime Pro Edition software supports the following ECO commands:

Note: Check available arguments by running `<eco_command> -h|help|long_help`.

7.4.1. ECO Command Quick Reference

Table 45. ECO Command Quick Reference

ECO Change	ECO Commands
Route	`make_connection -from <src> -to <dst> -port <port>` `remove_connection -from <src> -to <dst> -port <port>`
Tie-Off	`make_connection –tieoff <VCC/GND> -to <node> -port <port>`
Lutmask	`modify_lutmask -to <node> [-eqn <lut equation>] [-mask 0x00]`
Slew Rate	`modify_io_slew_rate <value> -to <pin_name>`
Current Strength	`modify_io_current_strength <value> -to <pin_name>`
Delay Chains	`modify_io_delay_chain <value> -type <io_type> -to <pin_name>`
Update MIF	`update_mif_files`
IOPLL Ref Clock (Stratix 10 devices only)	`adjust_pll_refclk -to <pll name> -refclk <freq>`
Create New Node	`create_new_node -type <LUT|FF> -name <name>`
Remove Node	`remove_node -name <name>`
Place Node	`place_node -name <name> [-location <location>]`
Unplace Node	`unplace_node -name <name>`
Create Wirelut	`create_wirelut -from <src> -to <dst> -port <port> [-location <location>]`

7.4.2. make_connection

Description

Connects the source signal to the destination block port. If the port has an existing connection, the command removes the previous connection and connects it to the signal you specify. The actual routing change occurs implicitly when appropriate. You can locate node names by right-clicking a node in Netlist Viewer, and then clicking **Properties**.

`make_connection` also supports adding connections from and to Hyper-Registers. In the case of Hyper-Registers, the command first disconnects the destination port, before making a new connection. You can the run the `make_connection` command to specify a replacement signal source or destination.

If a port is shared among multiple RAM slice atoms, then the ECO Fitter automatically updates all relevant atoms, and reports them accordingly.

Usage

The following example connects `top|a_out` to the D input port of node `top|x`.

```
make_connection -from top|a_out -to top|x -port D
```

Arguments

from Output net of the source block of the new connection.

to Name of the destination block.

port The input port name of the destination block.

Options

tieoff To explicitly tie off an input port to VCC or GND.

VCC or GND

Example:

```
make_connection –tieoff VCC –to {node1} –port DATAA
```

to Name of the destination block.

port The input port name of the destination block.

7.4.3. remove_connection

Description

Disconnects the `src` signal from the destination block port. The actual routing change occurs implicitly. You can locate node names by right-clicking a node in Netlist Viewer, and then clicking **Properties**.

If a port is shared among multiple RAM slice atoms, then the ECO Fitter automatically updates all relevant atoms, and reports them accordingly.

Usage

The following example disconnects `top|a_out` from the D input port of node `top|x`, and set `top|x:D` to a disconnected state.

```
remove_connection -from top|a_out -to top|x -port D
```

Arguments

from Output net of the source block of the current connection.

to Name of the destination block.

port The input port name of the destination block.

7.4.4. modify_lutmask

Description

Modifies the lutmask of the matching destination node, with the lutmask value (`-mask`) in binary or hexadecimal, or with the equivalent lutmask value (`-eqn`) computed from specified logical equation.

Usage

The following example disconnects `top|a_out` from the D input port of node `top|x`, and set `top|x:D` to a disconnected state.

```
modify_lutmask -to top|lut_c -eqn {a&b&c}
modify_lutmask -to top|lut_a -mask 0xFF00FF00
modify_lutmask -to top|lut_b -mask 0b111111111001010
```

Arguments

eqn The logical equation of the inputs (`A, B, C, D, E, F`). The supported lexical tokens include `AND('&')`, `OR('|')`, `XOR('^')`, `NOT('!')`, `OPEN_BRACE('(')`, `CLOSE_BRACE(')')`. Specify `-mask` or `-eqn`

to Destination atom name.

mask The lutmask value to be modified in binary or hexadecimal format. Specify `-mask` or `-eqn`

Note: When you view the lutmask equations in the Resource Property Viewer, the equations display in terms of `F0/F1/F2/F3` LUTs for `A`, `B`, `C` and `D` inputs. For LUTs also using `E` or `F` inputs, you must combine these sub-functions using the connectivity that the ALM diagram shows for the `E` and `F` muxes.

7.4.5. adjust_pll_refclk

Description

Changes the IOPLL frequencies by modifying the input reference clock frequency. The following stipulations apply:

- Maintain the original refclk and outclk ratios.
- The IOPLLs you change cannot generate IP clocks.
- Cascaded IOPLLs must connect directly (no clock gates in between them).
- IOPLLs cannot be in 'nondedicated' compensation modes.
- For all IOPLLs, outclks duty cycle equals 50 and phase shift equals 0.
- No support for Agilex 7 devices.

Usage

The following example adjusts the `*pll_main*` IOPLL by modifying the input clock frequency to 100 MHz.

```
adjust_pll_refclk -to {*pll_main*} -refclk 100
```

Arguments

to Instance name of upstream IOPLL that you want to adjust. Escape any [or] characters in the target name.

refclk New refclk frequency value in MHz.

7.4.6. modify_io_slew_rate

Description

Implements the I/O pin slew setting rate that you specify for the I/O pin.

Usage

```
modify_io_slew_rate 1 -to top|ipin
```

Arguments

to Instance name of destination pin that you want to modify.

7.4.7. modify_io_current_strength

Description

Implements the change to the I/O pin current strength setting that you specify for the I/O pin.

Usage

```
modify_io_current_strength 3mA -to top|ipin
```

Arguments

to Instance name of the destination pin that you want to modify.

7.4.8. modify_io_delay_chain

Description

Implements the change to the delay chain settings that you specify for the I/O pin.

Usage

```
modify_io_delay_chain 3 -to top|ipin -type input
```

Arguments

type Specifies one of the following I/O types: `input`, `output`, oe, `io_12_lane_input`, `io_12_lane_input_strobe`

to Instance name of I/O pin that you want to modify the delay chain settings.

7.4.9. create_new_node

Description

Creates a LUT cell or flip-flop in the design netlist. The following use cases apply:

- Adding a gate to fix a logic bug.
- Adding a wire LUT to help add hold delay.
- Creates a new flip-flop or flip-flops where needed.

The name of the new node is hierarchical. Therefore, when creating node a|b|c|d, you must ensure that hierarchy a|b|c exists in the netlist. If the source or destination node lies under a partition, the new LUT inserts under that partition.

Note: This command does not support extended or arithmetic LUTs.

After creating the new node, you can run the following commands to connect, modify the lutmask, or place the new node:

1. Run `make_connection` to connect to the new LUT's DATA inputs and output port.
2. Run `modify_lutmask` to change the lutmask for the new LUT.
3. Run `place_node` to place (and subsequently route) the new LUT.

This flow ensures that all routing requirements are analyzed when determining a legal placement for the new node.

Usage

The following example creates a `new_lut` LUT node with input ports DATAA and DATAB, and with outputs connected accordingly. `modify_lutmask` the modifies the lutmask to perform A&B logic. `place_node` next places the new LUT. The connections route after node placement is complete.

```
create_new_node -name new_lut -type lut
make_connection -from src_a -to new_lut -port DATAA
make_connection -from src_b -to enew_lut -port DATAB
make_connection -from new_lut -to dst_reg -port D
modify_lutmask -to new_lut -eqn A&B
place_node -name new_lut
```

To connect to a new flip-flop node that you create, use the `make_connection` command to connect to the flip-flop data port (`D`), and control ports (`CLK`, `ENA`, `SCLR`, `CLRN`), and from its output `Q` port. You must place the new flip-flop node with the `place_node` command. The connections are automatically routed after the `place_node` command.

```
create_new_node -name my_ff -type ff
make_connection -from reg0 -to my_ff -port D
make_connection -from clk -to my_ff -port CLK
make_connection -from my_ff -to reg1 -port D
place_node -name my_ff -location “X10 Y10 X10 Y10”
```

Arguments

name Name of the new LUT of flip-flop node.

7.4.10. remove_node

Description

Removes a LUT cell or flip-flop from the design netlist.

Usage

The following example deletes a flip-flop node with the name `ff1`:

```
remove_node -name ff1
```

Arguments

name Name of the LUT of flip-flop node to delete.

7.4.11. place_node

Description

Places the node that you specify in a location that the ECO Fitter selects. Optionally, you can specify the `location` argument to assign a specific device region location. You can also run this command for nodes already placed by the Fitter.

`place_node` also supports placement of newly added or existing flip-flops. `place_node` does not support Hyper-Register locations.

Usage

The following examples show three placement cases. For `node1`, the ECO Fitter determines the placement location. For `node2`, the command specifies the exact LAB location constraint. For `node3`, the command specifies a placement region constraint.

```
place_node -name node1 # let ECO Fitter decide placement
place_node -name node2 -location FF_X20_Y60_N17 # place node at specific location
place_node -name node3 -location “X10 Y10 X20 Y20” # place node in region
place_node -name my_ff -location “X10 Y10 X10 Y10” # place flip-flop in region
```

Arguments

name Name of the node.

location Device region coordinates `(X1 Y1 X10 Y10)(X1 Y1) (FF_X20_Y60_N17)`.

7.4.12. unplace_node

Description

Unplaces the node that you specify. `unplace_node` supports moving larger clouds of logic. To simplify the process of moving a larger cloud of logic, such as an entire ALM, you can first unplace all of the nodes. The Fitter does not perform placement legality checking until you re-place the final ALM cell.

Usage

The following example unplaces a node with the name `ff1`:

```
unplace_node -name ff1
```

Arguments

name Name of the node to unplace.

7.4.13. create_wirelut

Description

Creates and inserts a wire LUT node in the connection that you specify. The ECO Fitter places the new LUT and routes the modified connections automatically. Optionally, you can specify the `location` argument to specify a particular device region location constraint.

The name of the new node is hierarchical. Therefore, when creating node `a|b|c|d`, you must ensure that hierarchy `a|b|c` exists in the netlist. If the source or destination node lies under a partition, the new wire LUT inserts under that partition.

`create_wirelut` also supports adding connections from and to Hyper-Registers. In the case of Hyper-Registers, the command first disconnects the destination port, before making a new connection. You can the run the `create_wirelut` command to specify a replacement signal source or destination.

If a port is shared among multiple atoms (for example, RAM), then the ECO Fitter automatically updates all relevant atoms, and reports them accordingly.

Usage

The following example creates the `my_wirelut` wire LUT, connects `my_wirelut` output to the D input port of `dest_node`, and connects the output of `src_output` to the input port of the wire LUT. Finally, the ECO Fitter places the new node within region `(20, 20)` to `(40, 40)` and routes automatically.

```
create_wirelut -name my_wirelut -from src_output -to dest_node \
     -port D -location "X20 Y20 X40 Y40"
```

Arguments

name Name of the node.

From Source of the connection.

To Name of destination node.

Port Input port name of destination node.

location Device region coordinates `(X1 Y1 X10 Y10)` `(X1 Y1)` `(FF_X20_Y60_N17)`.

7.5. ECO Command Limitations

The ECO commands have the following limitations due to connection dependencies within Intel FPGA devices.

- You cannot use ECO commands to modify dedicated connections.
- You cannot modify dedicated connections within a single ALM. This limitation applies to direct connections between LUT and flip-flop nodes.
- You can connect from or to a Hyper-Register. However, you cannot remove connections from or to a Hyper-Register because removing a connection from a Hyper-Register would leave the routing dangling. As an alternative, you can use `make_connection` to change a Hyper-Register connection immediately, without removing the previous connection first.
- Use of the `place_node` command with `location` arguments does not overwrite Partial Reconfiguration region constraints.
- If a LAB already has the maximum number of legal connections where a node is placed, the `place_node` or `make_connection` commands can fail, preventing the connection to the first placed node that cannot be legalized. You can then either move the original node to a different location, or move other nodes from the LAB to free up routing resources.
- The Fitter may fail to apply some I/O related ECO modifications, such as `modify_io_slew_rate`, `modify_io_current_strength`, and `modify_io_delay_chain`, if called using a command-line Tcl script or in interactive context. That is, any case that calls the `eco_load_design` command directly. To ensure all I/O modifications are applied successfully, use the standard ECO Tcl script approach this document describes.

The recommended order for creating and placing new LUTs or new flipflops is:

1. Create the node by using the `create_new_node` command.
2. Make connections to and from the node by using the `make_connection` command.
3. Update the lutmask by using the `modify_lutmask` command.
4. Place the node by using the `place_node` command.

This flow ensures that analysis includes all routing requirements when determining a legal placement for the new node. For example:

Create a new LUT in an exact location

```
set lut_name new_lut
create_new_node –name $lut_name –type lut
make_connection –from input1 –to $lut_name –port DATAA
make_connection –from input2 –to $lut_name –port DATAB
make_connection –from $lut_name –to output_dest –port DATAD
modify_lutmask –to $lut_name –eqn {A&B}
place_node –name $lut_name –location “X80 Y80 X85 Y95”
```

Create a new Flipflop in an exact location

```
set ff_name new_ff
create_new_node –name $ff_name –type ff
make_connection –from input1 –to $ff_name –port DATAA
make_connection –from input2 –to $ff_name –port DATAB
make_connection –from $ff_name –to output_dest –port DATAD
modify_lutmask –to $ff_name –eqn {A&B}
place_node –name $ff_name –location “X80 Y80 X85 Y95”
```

Note: To minimize issues with name matching caused by escaped characters, it can be useful to surround entity names with {} characters, instead of "". This technique is particularly useful if entity names contain backslashes or any other special characters.

7.6. Interactive ECO Fitting

The `quartus_fit` executable supports ECO changes in an interactive shell through `quartus_fit -s`.

In an interactive context, the ECO Fitter legalizes the changes, when appropriate. For example, for `make_connection` changes immediately after node creation, the Fitter does not attempt to route the connections immediately; rather, the Fitter waits until after the placement of the node prior to routing.

7.6.1. eco_load_design and eco_commit_design Commands

Description

- `eco_load_design`—loads the final netlist in the ECO context.
- `eco_commit_design`—commits the ECO modified netlist to disk while running in interactive mode.

Usage

The following example shows an interactive ECO session to modify the `lut_x` lutmask for project `top`. If the ECO modifications are legal, the `eco_commit_design` command commits the final netlist.

```
$ quartus_fit -s
>> load_package eco
>> project_open top
>> eco_load_design
>> modify_lutmask -to lut_x -eqn B&C # some ECO changes
>> eco_commit_design
>> project_close
```

7.7. Using the ECO Compilation Flow Revision History

The following revision history applies to this chapter:

Table 46. Document Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2020.09.28	20.3	• Revised chapter title to "Using the ECO Compilation Flow." • Added descriptions of new `unplace_node` and `delete_node` commands. • Described new support for placement of flip-flop nodes and exact locations in `place_node` command topic. • Described new support for creation of flip-flop nodes in `create_new_node` command topic. • Updated limitations in "ECO Command Limitations to remove obsolete limitations." • Revised wording of introduction. • Added report screenshots to "Viewing ECO Compilation Reports" topic.
2020.05.08	20.1	• Added descriptions of new `create_new_node`, `place_node`, and `create_wirelut` commands. • Referenced support for multi-node ECOs in `make_connection`, `remove_connection`, and `create_wirelut` command topics. • Referenced Support for ECO connections to Hyper-Registers in the `make_connection` topic. • Described updates to ECO reporting in "Viewing ECO Compilation Reports." • Updated limitations in "ECO Command Limitations." • Added ECO Command Quick Reference
2019.09.30	19.3.0	• Added information about `tieoff` option for `make_connection` command. • Added support for `modify_io_slew_rate` command. • Added support for `modify_io_current_strength` command. • Added support for `modify_io_delay_chain` command. • Added "Viewing ECO Compilation Reports" topic. • Added information about `num` option for `modify_lutmask` command. • Mentioned RTL Viewer for locating node names. • Added device support note.
2019.07.01	19.2.0	• First release of chapter.

8. Quartus Prime Pro Edition Design Optimization User Guide Archives

For the latest and previous versions of this user guide, refer to Quartus Prime Pro Edition User Guide: Design Optimization. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Programmer

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

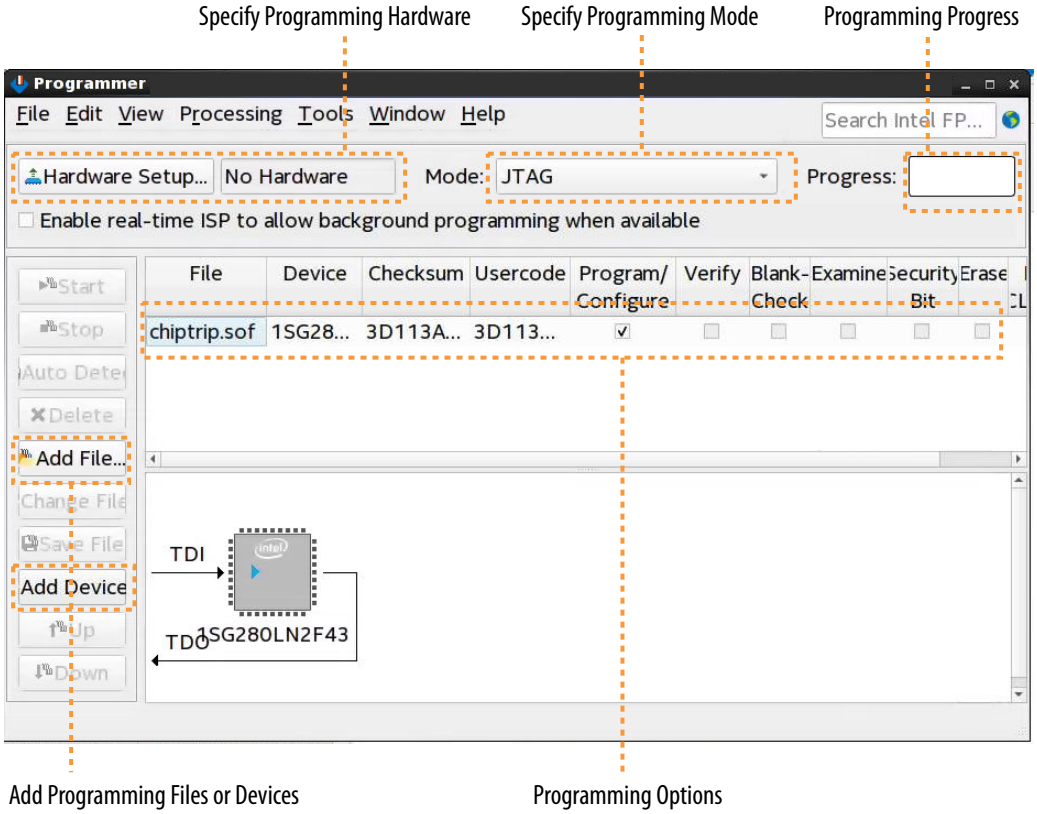
Answers to Top FAQs:

Contents

1. Quartus® Prime Programmer User Guide

The Quartus® Prime Programmer allows you to program and configure Intel® CPLD, FPGA, and configuration devices. Following full design compilation, you generate the primary device programming files in the Assembler, and then use the Programmer to load the programming file to a device. This user guide details Intel FPGA programming file generation and use of the Quartus Prime Programmer.

Figure 1. Quartus Prime Programmer



Related Information

- Generating Primary Device Programming Files on page 5
- Generating Secondary Programming Files on page 6
- Enabling Bitstream Security for Stratix 10 and Agilex 7 Devices on page 19
- Using the Quartus Prime Programmer on page 40
- Programming with Flash Loaders on page 51

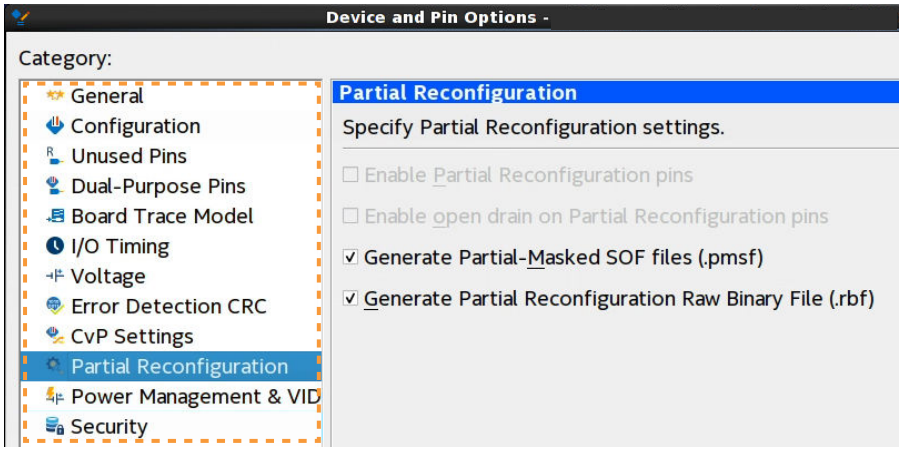
1.1. Generating Primary Device Programming Files

By default, the Compiler's Assembler module generates the primary device programming files at the end of full compilation. Alternatively, you can start the Assembler independently any time after design place and route to generate primary device programming files, such as SRAM Object Files (`.sof`) for configuration of Intel FPGAs.

Follow these steps to generate primary device programming files:

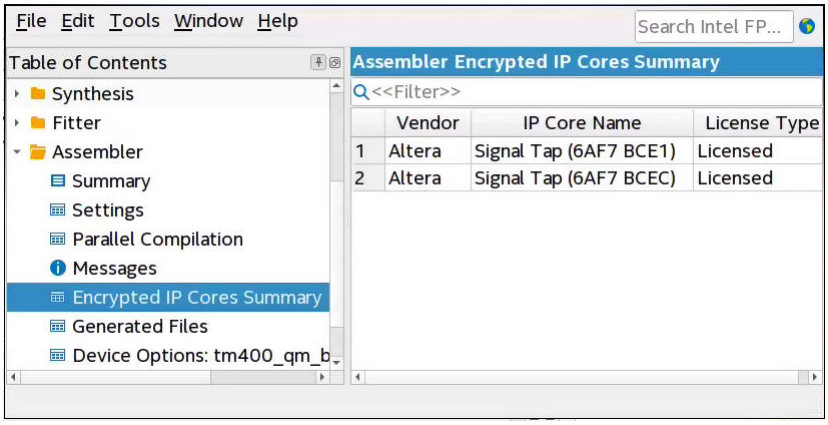
1. To specify programming options that enable features in the primary device programming file, such as **Configuration**, **Error Detection CRC**, and device **Security** options, click **Assignments ➤ Device ➤ Device & Pin Options**. Device & Pin Options Dialog Box on page 55 describes all options.

Figure 2. Device & Pin Options Dialog Box (Stratix® 10 Design)



2. To generate primary device programming files, click **Processing ➤ Start ➤ Start Assembler**, or double-click **Assembler** on the Compilation Dashboard. The Assembler generates the programming files according to the options you specify.
3. After running the Assembler, view detailed information about programming file generation, including the programming file Summary and Encrypted IP information in the Assembler report folder in the Compilation Report.

Figure 3. Assembler Reports



Note: Each successive release of the Quartus Prime software typically includes:

- Added support for new features in supported FPGA devices.
- Added support for new devices.
- Efficiency and performance improvements.
- Improvements to compilation time and resource use of the design software.

Due to these improvements, different versions of the Quartus Prime Pro Edition, Quartus Prime Standard Edition, and Quartus Prime Lite Edition software can produce different programming files from release to release.

1.2. Generating Secondary Programming Files

After generating primary device programming files, you can optionally generate one or more derivative programming files for alternative device configurations, such as flash programming, partial reconfiguration, remote system update, Configuration via Protocol (CvP), or hard processor system (HPS) core configuration.

You can use the **Programming File Generator** or **Convert Programming Files** dialog box to generate secondary programming files:

- The **Programming File Generator** supports advanced programming features and is optimized for Agilex™ 5, Agilex 7, Stratix® 10, MAX® 10, and Cyclone® 10 LP devices.
- The **Convert Programming Files** dialog box supports all devices released prior to Stratix 10 devices.

Table 1. Secondary Programming File Generators

	Programming File Generator	Convert Programming Files	
Device Support	• Agilex 5 • Agilex 7 • Stratix 10 • MAX 10 • Cyclone 10 LP	• Arria® 10 • Cyclone 10GX and LP • MAX 10	APEX20K, Arria II GX and GZ, Arria V, Cyclone, Cyclone II, Cyclone III and LS, Cyclone IV E and GX, Cyclone V, HardCopy® III, HardCopy II, HardCopy IV, MAX V, Stratix, Stratix II, Stratix III, Stratix IV, Stratix V

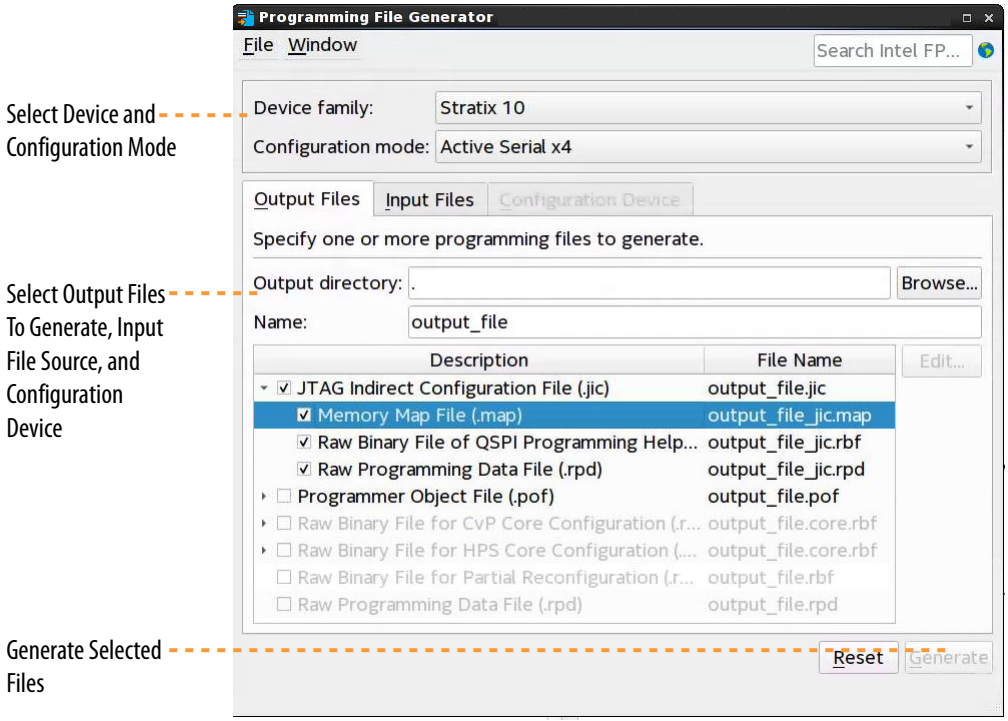
1.2.1. Generating Secondary Programming Files (Programming File Generator)

Follow these steps to generate secondary programming files for alternative device programming methods with the **Programming File Generator**.

1. Generate the primary programming files for your design, as Generating Primary Device Programming Files on page 5 describes.
2. Click **File ➤ Programming File Generator**.
3. For **Device family**, select your target device. The options available in the **Programming File Generator** change dynamically, according to your device and configuration mode selection.
4. For **Configuration mode**, select the target configuration mode for your device. Configuration Modes (Programming File Generator) on page 9 describes all modes.

5. On the **Output Files** tab, enable the checkbox for generation of the file you want to generate. The **Input Files** tab is now available. Secondary Programming Files (Programming File Generator) on page 10 describes all output files.
6. Specify the **Output directory** and **Name** for the file you generate. Output Files Tab Settings (Programming File Generator) on page 63 describes all options.

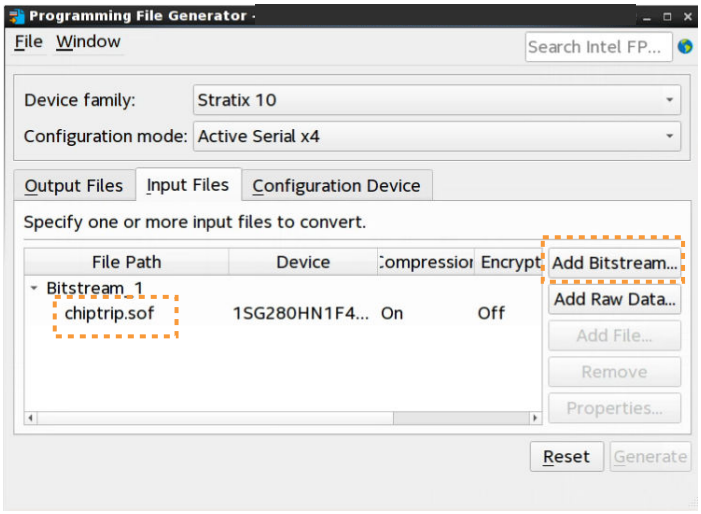
Figure 4. Programming File Generator



7. To specify a `.sof` file that contains the configuration bitstream data, on the **Input Files** tab, click **Add Bitstream**.
8. To include raw data, click **Add Raw Data** and specify a Hexadecimal (Intel-Format) file (`.hex`), Binary (`.bin`) file, or uncompressed ZIP file (`.zip`).

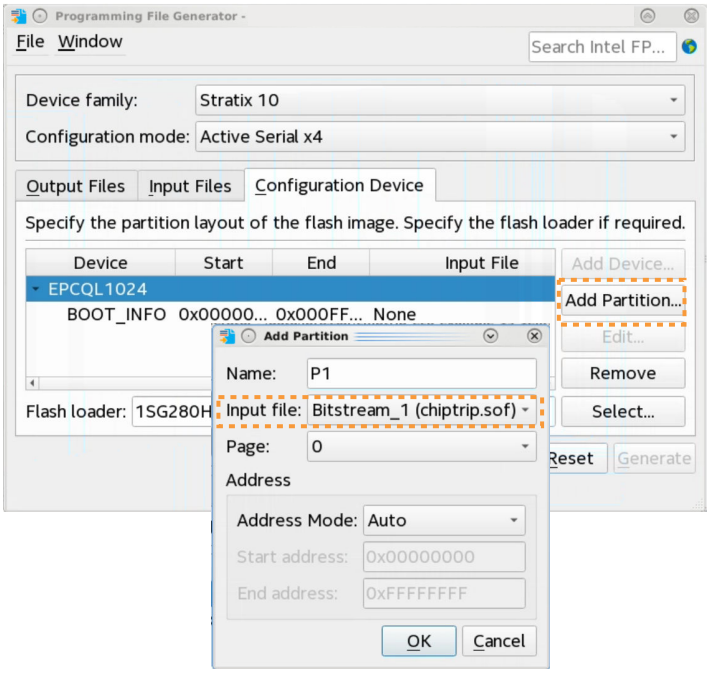
   Important: The ZIP file must be uncompressed. For example, in WinZip® file compression software, ensure that the compression mode is set to **No compression**.
9. To specify bitstream authentication or encryption security settings for the file, select the `.sof` and click **Properties**, as Enabling Bitstream Authentication (Programming File Generator) on page 20 describes.

Figure 5. Input Files Tab



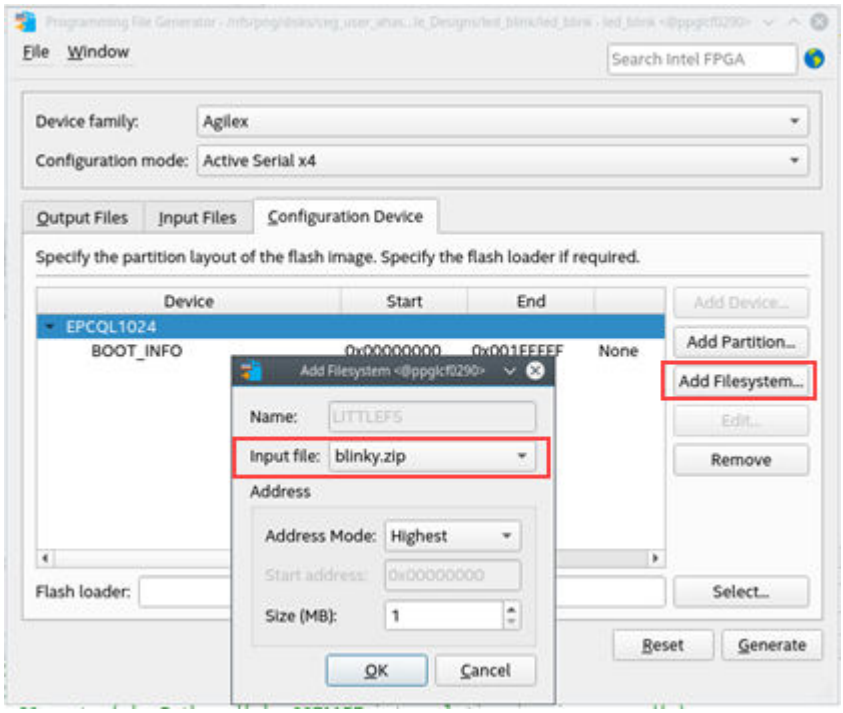
10. To create a flash memory partition and specify a `.sof` file that occupies the flash memory partition, click **Add Partition** on the **Configuration Device** tab. Add Partition Dialog Box (Programming File Generator) on page 66 describes all options.

Figure 6. Add Flash Partition



11. To create a file system partition and specify an uncompressed `.zip` file that contains the files to write to the file system partition, click **Add Filesystem** on the **Configuration Device** tab. Add Filesystem Dialog Box (Programming File Generator) on page 66 describes all options.

Figure 7. Add Flash Filesystem Partition



Important: The ZIP file must be uncompressed. For example, in WinZip file compression software, ensure that the compression mode is set to **No compression**.

12. To select a supported flash memory device and predefined programming flow, click **Add Device** on the **Configuration Device** tab. Alternatively, click **<<new device>>** to define a new flash memory device and programming flow. Configuration Device Tab Settings on page 65 describes all settings.
13. Click the **Select** button for **Flash Loader** and select the device that controls loading of the flash memory device. Select Devices (Flash Loader) Dialog Box on page 69 describes all settings.
14. After you specify all options in **Programming File Generator**, the **Generate** button enables. Click **Generate** to create the files.

1.2.1.1. Configuration Modes (Programming File Generator)

Select one of the following **Configuration modes** in **Programming File Generator** for generation of secondary programming files:

Table 2. Programming File Generator Configuration Modes

Programming Mode	Description	Supports Devices
Active Serial x4	For storing configuration data in a low-cost serial configuration device with non-volatile memory and four-pin interface. Serial configuration devices provide a serial interface to access the configuration data. During device	• Agilex 5 • Agilex 7 • Stratix 10

continued...

Programming Mode	Description	Supports Devices
	configuration, Stratix 10 devices read the configuration data through the serial interface, decompress the data if necessary, and configure their SRAM cells.	
AVST x8	The Avalon® streaming configuration mode uses an external host, such as a microprocessor or MAX 10 device. The external host controls the transfer of configuration data from an external storage such as flash memory to the FPGA. The design that controls the configuration process resides in the external host. You can use the PFL II IP core with an MAX 10 device as the host to read configuration data from a flash memory device that configures an Stratix 10 FPGA.	
AVST x16		
AVST x32		
1-Bit Passive Serial	An external controller passes configuration data to one or more FPGA devices via a serial data stream. The FPGA device is a slave device with a 5-wire interface to the external controller. The external controller can be an intelligent host such as a microcontroller or CPU.	Cyclone 10 LP
Active Serial	Stores configuration data in a low-cost serial configuration device with non-volatile memory and four-pin interface.	
Internal Configuration	Uses a `.pof` file for internal configuration of the MAX 10 device's Configuration Flash Memory (CFM) and User Flash Memory (UFM) via a download cable Quartus Prime Programmer.	MAX 10

1.2.1.2. Secondary Programming Files (Programming File Generator)

After generating primary device programming files, you can generate the following secondary device programming files with the **Programming File Generator** for alternative device configuration modes:

Table 3. Programming File Generator Output File Types

Programming File Type	Extension	Description
Hexadecimal (Intel-Format) Output File for SRAM	`.hexout`	An ASCII text file in Intel hexadecimal format that contains configuration data for programming a parallel data source, such as a configuration device or a mass storage device. The parallel data source in turn configures an SRAM-based Intel device.
JTAG Indirect Configuration File	`.jic`	Proprietary Intel FPGA file type that stores serial flash programming data for programming via Intel FPGA JTAG pins. This method only supports Active Serial configuration. Before programming the flash, the Programmer first configures the FPGA with the Serial Flash Helper Design.
Memory Map File	`.map`	A text file containing the byte addresses of pages and HEX data stored in the memory of the selected configuration device. The file stores the start and end addresses of the **Main Block Data** and **Bottom Boot Data** items, and the start and end addresses of pages within the **Main Block Data** item.
Programmer Object File	`.pof`	A binary file used by the Programmer to program a flash memory device via active serial header, or to program a flash memory device via the Parallel Flash Loader Intel FPGA IP.
Raw Binary File	`.rbf`	Configuration bitstream file for use with a third-party data source, partial reconfiguration, or HPS data source. Supports Passive Serial (PS) and Avalon-Streaming (AVST) modes.
Raw Binary File for CvP Core Configuration	`.rbf`	A binary file that containing logic that is programmed by configuration (CRAM) for CvP phase 2. The core bitstream is in `.rbf` format.

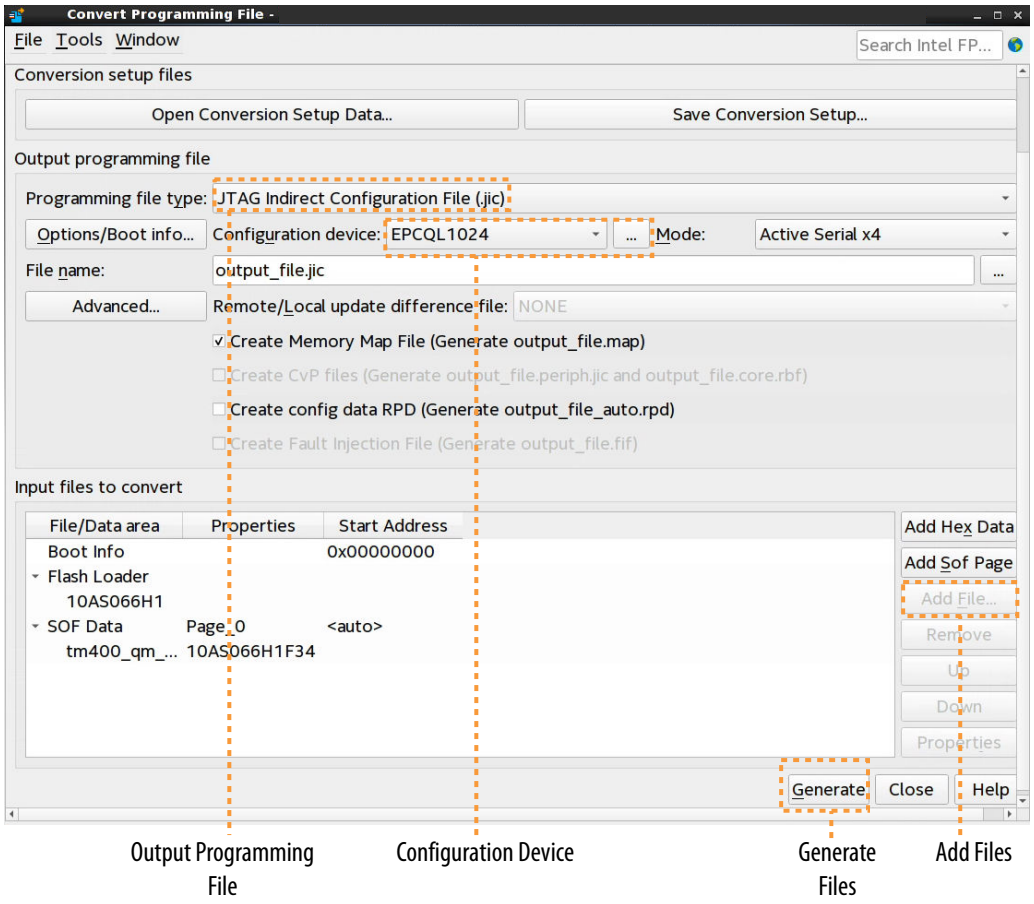
continued...

Programming File Type	Extension	Description
Raw Binary File for HPS Core Configuration	`.rbf`	A binary file that containing logic that is programmed by configuration (CRAM) for HPS configuration phase 2. The core bitstream is in `.rbf` format.
Raw Programming Data File	`.rpd`	Stores data for configuration with third-party programming hardware. You generate Raw Programming Data Files from a `.pof` or `.sof`. The `.rpd` file is a subset of a `.pof` or `.jic` that includes only device-specific binary programming data for Active Serial configuration scheme with EPCS or EPCQ serial configuration devices and remote system update.
Tabular Text File	`.ttf`	A TTF contains the decimal equivalent of a Raw Binary File (.rbf).

1.2.2. Generating Secondary Programming Files (Convert Programming File Dialog Box)

You can use the **Convert Programming File** dialog box to generate secondary programming files for alternative device programming methods. For example, generating the `.jic` file for flash programming, the `.rbf` file for partial reconfiguration, or the `.rpd` file for a third-party programmer configuration.

Figure 8. Convert Programming File Dialog Box

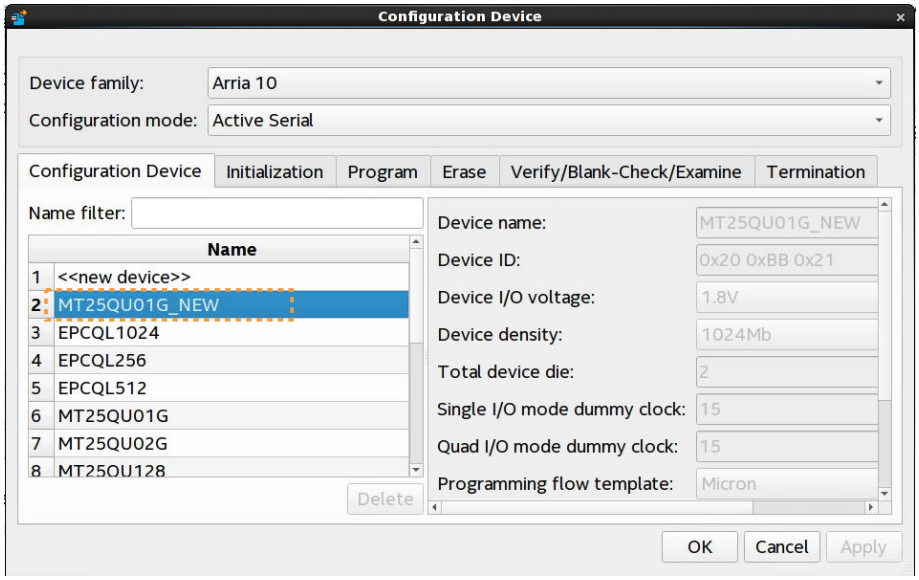


The options available in the **Convert Programming File** dialog box change dynamically, according to your device and configuration mode selection.

1. Generate the primary programming files for your design, as Generating Primary Device Programming Files on page 5 describes.
2. Click **File ➤ Convert Programming Files**.
3. Under **Output programming file**, select the **Programming file type** that you want to generate. Secondary Programming Files (Convert Programming Files) on page 13 describes all file options.
4. Specify the **File name** and output directory **(...)** for the file that you generate.
5. For the configuration **Mode**, select **Active Serial x4** or **Active Serial**. Configuration Modes (Convert Programming Files) on page 14 describes all modes.

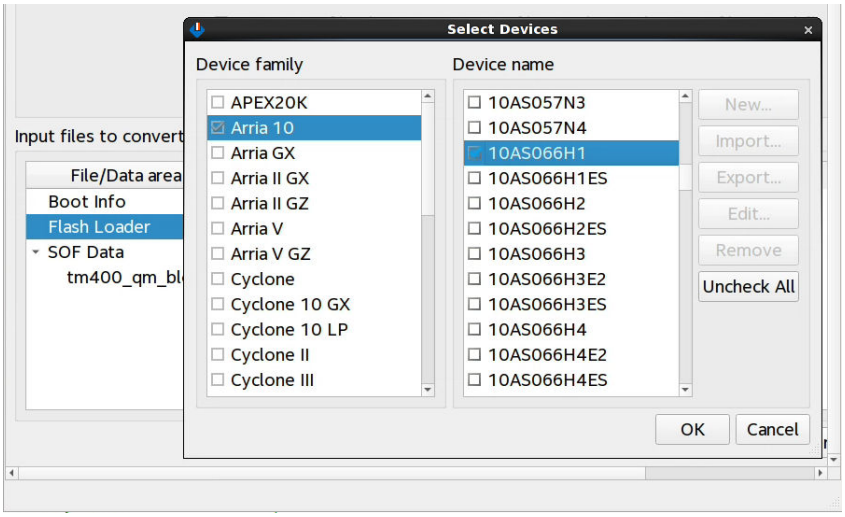
   Note: Stratix 10 devices support only **Active Serial x4**.
6. To specify the **Configuration device**, click the **(...)** button to select a supported flash memory device and predefined programming flow. When you select a predefined device, you cannot modify any setting. Alternatively, click **<<new device>>** to define a new flash memory device and programming flow. Configuration Device Tab Settings on page 65 describes all settings.

Figure 9. Configuration Device Dialog Box



7. Under **Input files to convert**, select the **SOF Data** item, and then click the **Add File** button. Specify the `.sof` file that contains the configuration bitstream data. To include raw data, click **Add Hex Data** and specify a `.hex` file.
8. To enable bitstream compression or encryption security settings, select the `.sof` file and click **Properties**, as Enabling Bitstream Encryption or Compression for Arria 10 and Cyclone 10 GX Devices on page 24 describes.
9. Select the **Flash Loader** text, and then click the **Add Device** button. Select the device that controls loading of the flash device.

Figure 10. Selecting the Flash Loader Device



10. After you specify all options in the **Convert Programming File** dialog box, click the **Generate** button to create the files.

1.2.2.1. Secondary Programming Files (Convert Programming Files)

After generating primary device programming files, you can generate the following secondary device programming files with the **Convert Programming Files** dialog box for alternative device configuration modes:

Table 4. Output File Types

Programming File Type	Extension	Description
CvP Files	`.jic/.rbf`	Files required for CvP configuration.
Hexadecimal (Intel-Format) Output File for SRAM	`.hexout`	An ASCII text file in Intel hexadecimal format that contains configuration data for programming a parallel data source, such as a configuration device or a mass storage device. The parallel data source in turn configures an SRAM-based Intel device.
JTAG Indirect Configuration File	`.jic`	Proprietary Intel FPGA file type that stores serial flash programming data for programming via Intel FPGA JTAG pins. This method only supports Active Serial configuration. Before programming the flash, the Programmer first configures the FPGA with the Serial Flash Helper Design.
Memory Map File	`.map`	Contains the byte addresses of pages and HEX data stored in the memory of the selected configuration device. The Map File stores the start and end addresses of the **Main Block Data** and **Bottom Boot Data** items, and the start and end addresses of pages within the **Main Block Data** item.
Partial-Masked SRAM Object Files	`.pmsf`	Contains the partial-mask bits for configuration of a PR region. The `.pmsf` file contains all the information for creating PR bitstreams.
Merged Mask Setting File	`.msf`	Contains the mask bits for the static region in a PR design.
Programmer Object File	`.pof`	A binary file that contains the data for programming non-volatile MAX 10, MAX V, MAX II, or flash memory devices that can configure Intel FPGA devices. A Programmer consists of a remote update

continued...

Programming File Type	Extension	Description
		enabled `.pof` and additional remote update enabled `.sof` that you used to program configuration devices in remote update configuration mode.
Raw Binary File	`.rbf`	Configuration bitstream file for use with a third-party data source, partial reconfiguration, or HPS data source. Supports Passive Serial (PS) and Avalon-Streaming (AVST) modes.
Raw Programming Data File	`.rpd`	Stores data for configuration with third-party programming hardware. You generate Raw Programming Data Files from a `.pof` or `.sof`. The `.rpd` file is a subset of a `.pof` or `.sof` that includes only device-specific binary programming data for Active Serial configuration scheme with EPCS or EPCQ serial configuration devices and remote system update. The `.rpd` file content has one bit swapped in comparison with the output file.
Tabular Text File	`.ttf`	A TTF contains the decimal equivalent of a Raw Binary File (.rbf).

1.2.2.2. Configuration Modes (Convert Programming Files)

Select one of the following **Configuration modes** in **Convert Programming Files** for generation of secondary programming files:

Table 5. Convert Programming Files Configuration Modes

Programming Mode	Description
1-Bit/2-Bit/4-Bit/8-Bit Passive Serial	An external controller passes configuration data to one or more FPGA devices via a serial data stream. The FPGA device is a slave device with a 5-wire interface to the external controller. The external controller can be an intelligent host such as a microcontroller or CPU, or the Quartus Prime Programmer, or an EPC2 or EPC16 configuration device.
Active Parallel	Supports configuration devices using commodity 16-bit parallel flash memories to control the configuration interface.
Active Serial	For storing configuration data in a low-cost serial configuration device with non-volatile memory. Serial configuration devices provide a serial interface to access the configuration data. During device configuration, the device reads the configuration data through the serial interface, decompresses the data if necessary, and configures their SRAM cells.
Active Serial x4	
AVST x8/x16/x32	The Avalon streaming configuration mode uses an external host, such as a microprocessor or MAX 10 device. The external host controls the transfer of configuration data from an external storage such as flash memory to the FPGA. The design that controls the configuration process resides in the external host. You can use the PFL II IP core with an MAX 10 device as the host to read configuration data from a flash memory device that configures an FPGA.
Passive Parallel Synchronous	An external controller, such as a CPU, loads the design data into a device via a common data bus. Data is latched by the device on the first rising edge of a CPU-driven clock signal. The next eight falling clock edges serialize this latched data within the device. The device latches the next 8-bit byte of data on every eighth rising edge of the clock signal until the device is completely configured.
Passive Parallel Asynchronous	An external controller, such as a CPU, loads the design data into a device via a common data bus. The device accepts a parallel byte of input data. Intelligent communication between the external controller and the device allows the external controller to configure the device.
Internal Configuration	Uses a `.pof` file for internal configuration of the MAX 10 device's Configuration Flash Memory (CFM) and User Flash Memory (UFM) via a download cable Quartus Prime Programmer.

1.2.2.3. Debugging the Configuration (Convert Programming Files)

Click the **Advanced** option in the **Convert Programming Files** dialog box to debug the file conversion configuration. Only choose advanced settings that apply to the design's target Intel FPGA device.

Changes in the **Advanced Options** dialog box affect `.pof`, `.jic`, `.rpd`, and `.rbf` file generation.

The following table describes the **Advanced Options** settings:

Table 6. Advanced Options Settings

Option Setting	Description	Values
Disable EPCS/EPCQ ID check	Directs the FPGA to skips the EPCS/EPCQ silicon ID verification. Applies to single and multi device AS configuration modes on all devices.	Default setting is ON (EPCS/EPCQ ID check is enabled).
Disable AS mode CONF_DONE error check	Directs the FPGA to skip the CONF_DONE error check. Applies to single- and multi-device (AS) configuration modes on all devices.	Default setting is OFF (AS mode CONF_DONE error check is enabled).
Program Length Count adjustment	Specifies the offset you can apply to the computed PLC of the entire bitstream. Applies to single- and multi-device (AS) configuration modes on all FPGA devices.	Integer (Default = 0)
Post-chain bitstream pad bytes	Specifies the number of pad bytes appended to the end of an entire bitstream.	If the bitstream of the last device is uncompressed, default value is 0. Otherwise, default is 2
Post-device bitstream pad bytes	Specifies the number of pad bytes appended to the end of the bitstream of a device. Applies to all single-device configuration modes on all FPGA devices.	Zero or positive integer. Default is 0
Bitslice Padding Value	Specifies the padding value used to prepare bitslice configuration bitstreams, such that all bitslice configuration chains simultaneously receive their final configuration data bit. Use only in 2, 4, and 8-bit PS configuration mode, when you use an EPC device with the decompression feature enabled. Applies to all FPGA devices that support enhanced configuration devices.	0 or 1 Default value is 1

The following table lists possible symptoms of a failing configuration, and describes the advanced options necessary for configuration debugging.

Failure Symptoms	Disable EPCS/ EPCQ ID Check	Disable AS Mode CONF_DONE Error Check	PLC Settings	Post-Chain Bitstream Pad Bytes	Post-Device Bitstream Pad Bytes	Bitslice Padding Value
Configuration failure occurs after a configuration cycle.	—	Yes	Yes	Yes (1)	Yes (2)	—
Decompression feature is enabled.	—	Yes	Yes	Yes (1)	Yes (2)	—
Encryption feature is enabled.	—	Yes	Yes	Yes (1)	Yes (2)	—
CONF_DONE stays low after a configuration cycle.	—	Yes	Yes (3)	Yes (1)	Yes (2)	—
CONF_DONE goes high momentarily after a configuration cycle.	—	Yes	Yes (4)	—	—	—
FPGA does not enter user mode even though CONF_DONE goes high.	—	—	—	Yes (1)	Yes (2)	—
Configuration failure occurs at the beginning of a configuration cycle.	Yes	—	—	—	—	—
EPCS128	Yes	—	—	—	—	—
Failure in .pof generation for EPC device using Quartus Prime Convert Programming File Utility when the decompression feature is enabled.	—	—	—	—	—	Yes

1.2.3. Generating Secondary Programming Files (Settings: Programming Files Dialog Box)

Follow these steps to generate .rbf files automatically as part of a standard compilation from the Quartus Prime GUI.

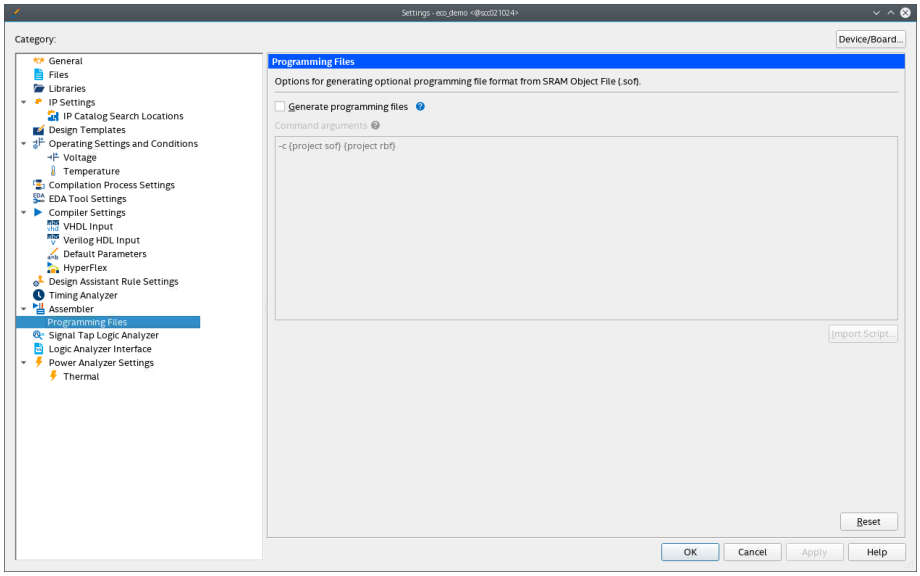
1. In Quartus Prime, select **Assignments ➤ Settings**.
2. In the **Settings** window, select **Assembler ➤ Programming Files**.

(1) Use only for multi-device chain

(2) Use only for single-device chain

(3) Start with positive offset to the PLC settings

(4) Start with negative offset to the PLC settings



3. In the **Programming Files** pane, select **Generate programming files**.

   Selecting this check box enables the default command arguments `-c .sof .rbf`.

4. (Optional) To override the default command arguments, enter your command in the **Command arguments** box or click **Import Script** to upload a script file.

   If the **Command arguments** box is not empty, only the command arguments in this box are run. The default command arguments are not run.

   Each argument line the **Command arguments** box is subject to the following conditions:

   - Lines must each begin with `-c` or `--convert`.
   - Lines must each contain at least two argument tokens.
   - Lines cannot contain pipes. For lines that contain pipes, only the first command argument before the first pipe in the command argument is executed.

   For example, `-c <project_name>.sof <project_name>.rbf`.

   If an argument line does not meet these conditions, you might see one of the following errors:

Figure 11. Error message when command argument lines do not begin with -c or --convert

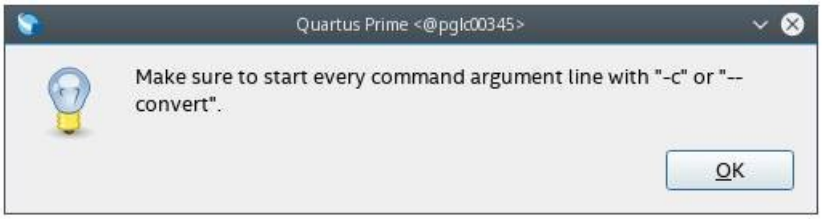


Figure 12. Error message when command argument line does not contain at least two arguments

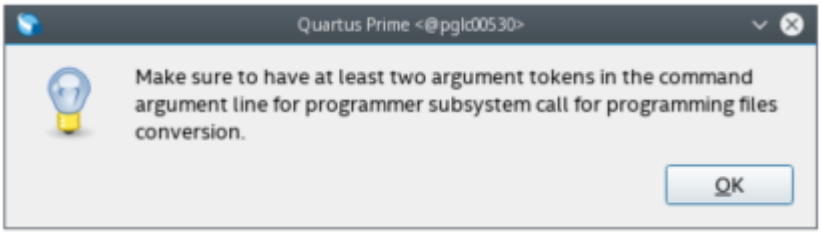
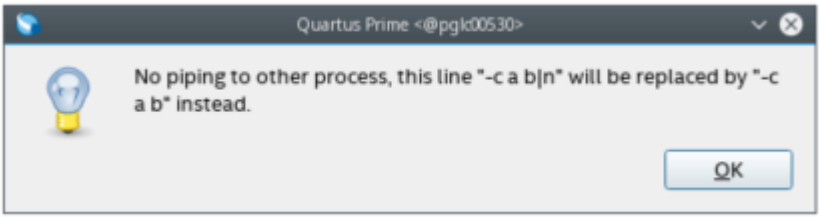


Figure 13. Error message when command argument line contains a pipe



5. Click **Apply** then click **OK**.

Ensure that the path to the `.sof` file is correct, otherwise you receive error messages similar to the following messages:

```
Error(22528): Programming files generation using command Error (19509):
Cannot locate file output_files/abc.sof. was unsuccessful. Check pfg.log for more
info.
```

A `pfg_commands.txt` file is generated and the QSF file is updated with the following settings:

```
GENERATE_PROGRAMMING_FILES=ON
CONVERT_PROGRAMMING_FILES_COMMANDS /<user-project-path>/pfg_commands.txt
```

1.3. Enabling Bitstream Security for Stratix 10 and Agilex 7 Devices

Stratix 10 and Agilex 7 devices provide flexible and robust security features to protect sensitive data, intellectual property, and device hardware from physical and remote attacks. The Stratix 10 and Agilex 7 device architectures support bitstream authentication and encryption security features. The Assembler applies bitstream compression automatically to reduce file size whenever you use authentication or encryption.

- **Bitstream Authentication**—verifies that the configuration bitstream and firmware are from a trusted source. Enable additional co-signing device firmware authentication to ensure that only signed firmware runs on the HPS or FPGA, and to authorize HPS JTAG debugging. Enable authentication security by specifying a first level signature chain file (`.qky`) for the **Quartus Key File** option (**Device and Pin Options** dialog box), as Enabling Bitstream Authentication (Programming File Generator) on page 20 describes.(5)
- **Bitstream Encryption**—protects proprietary or sensitive data from view or extraction in the configuration bitstream using an Advanced Encryption Standard (AES) 256-bit or 384-bit security key. Encryption also provides side-channel protection from non-intrusive attack. You can store the owner AES key in eFuses or BBRAM. Enable encryption by turning on the **Enable programming bitstream encryption** option (**Device and Pin Options** dialog box), as Enabling Bitstream Encryption (Programming File Generator) on page 23 describes.

Table 7. Stratix 10 and Agilex 7 Bitstream Authentication Files

Term	Description	Extension
First Level Signature Chain Key File	File you generate that specifies the root key (`.pem`) and one or more design signing keys (`.pem`) required to sign the bitstream and allow access to the FPGA when using authentication or encryption.	`.qky`
Root Key File	File you generate that anchors the first level signature chain to a known root key. The FPGA calculates the hash of the root entry and checks if it matches the expected hash. The Assembler appends the root key to the programming file and stores the key in eFuses.	`.qky`
Design Signing Key File	File you generate and append to the root key that authenticates the bitstream in the SDM to allow configuration of the device with the pending bitstream. Use separate design signing keys for the FPGA and HPS for highest security.	`.pem`
Firmware Co-signing Key File	Files provided in `<install>\devices\programmer\firmware` that includes the owner signature and firmware file that you use to sign the firmware to run on the FPGA or HPS.	`.zip`
Signed HPS Certificate File	Specifies a secure HPS debug certificate that permits access to the JTAG interface for HPS debugging. A secure HPS debug certificate is valid until you power down or reconfigure the device.	`.cert`

Note: Arria 10 and Cyclone 10 GX devices do not support bitstream authentication.

Related Information

- Stratix 10 Device Security User Guide
  For detailed information on generating device security keys.

(5) For Stratix 10 devices, bitstream authentication is available only for devices that include the AS (Advanced Security) ordering code suffix and all Stratix 10 DX devices.

- Agilex 7 Device Security User Guide
  For detailed information on generating device security keys.

1.3.1. Enabling Bitstream Authentication (Programming File Generator)

Bitstream authentication requires that you generate a first level signature chain (`.qky`) that includes the root key and one or more design signing keys. The root key enables the base security features and authenticates the design signing key through the public signature chain. The root key stores the SHA-256 or SHA-384 hash of the key in eFuses.

You can also optionally enable firmware co-signature capability to require signing the version of configuration firmware that runs on your device. The FPGA device then can load only authenticated firmware.

Note: For step-by-step first level signature chain key generation instructions, refer to one of the following guides:

- *Stratix 10 Device Security User Guide*
- *Agilex 7 Device Security User Guide*

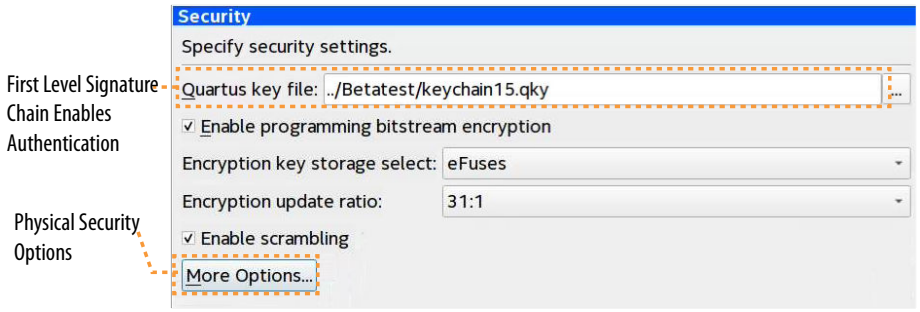
After you specify the `.qky` in Assembler settings, the Assembler appends the first level signature chain to the configuration `.sof` that you generate.

Use the **Programming File Generator** to generate the signed configuration bitstream for an `.sof` file. The JTAG Indirect Configuration File (`.jic`) and Raw Programming Data File (`.rpd`) formats are available for Active Serial (AS) configuration. The Programmer Object File (`.pof`) and Raw Binary File (`.rbf`) are available for Avalon Streaming configuration.

Follow these steps to enable bitstream authentication:

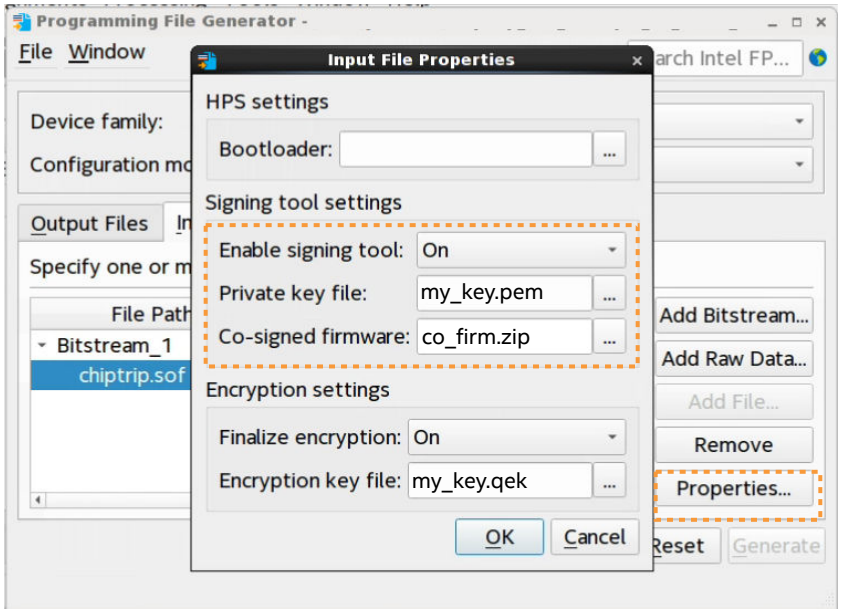
1. Generate a first level signature chain (`.qky`) that includes the root key and one or more design signing keys, as *Stratix 10 Device Security User Guide* and *Agilex 7 Device Security User Guide* describe.
2. To add the first level signature chain to a configuration bitstream, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Security**, and then specify the first level signature chain `.qky` for the **Quartus key file** option.
3. To enable more physical device security options, click the **More Options** button on the **Security** page. More Security Options Dialog Box on page 63 describes all options.

Figure 14. Security Tab (Device and Pin Options)



4. Generate primary device programing files in the Assembler, as Generating Primary Device Programming Files on page 5 describes. The primary device programming file now contains data to enable first level authentication.
5. To optionally enable co-signing device firmware authentication, generate a `.jic` or `.rbf` secondary programming file with the following options, as Generating Secondary Programming Files on page 6 describes:
   a. In **Programming File Generator**, click the **Properties** button. The **Input File Properties** dialog box appears.

Figure 15. Enabling Co-Signing Device Firmware Authentication (Stratix 10 Devices)



   b. Set **Enable signing tool** to **On**.
   c. For **Private key file**, specify a design signing key Privacy Enhanced Mail Certificates file (`.pem`) for firmware co-signing. This key can be separate from the FPGA design signing key.
   d. For **Co-signed firmware**, specify a Quartus Co-Signed Firmware file (`.zip`).
   e. Click **OK**.
6. Use the Programmer to configure the device with the `.jic` or `.rbf`.

Related Information

- Device & Pin Options Dialog Box on page 55
- Specifying Additional Physical Security Settings (Programming File Generator) on page 22
- Stratix 10 Device Security User Guide
  For detailed information on generating device security keys.
- Agilex 7 Device Security User Guide
  For detailed information on generating device security keys.

1.3.2. Specifying Additional Physical Security Settings (Programming File Generator)

Stratix 10 and Agilex 7 devices can store security and other configuration settings in eFuses. You can enable additional physical security settings in eFuses to extend the level of device security protection.

To specify additional physical device security settings, follow these steps:

1. Click **Assignments ➤ Device ➤ Device and Pin Options ➤ Security**.
2. On the **Security** tab, specify the First Level Signature Chain `.qky` file that contains the root key and one or more design signing keys for the **Quartus key file** setting.
3. Click the **More Options** button and specify any of the following:

Figure 16. More Security Options Dialog Box

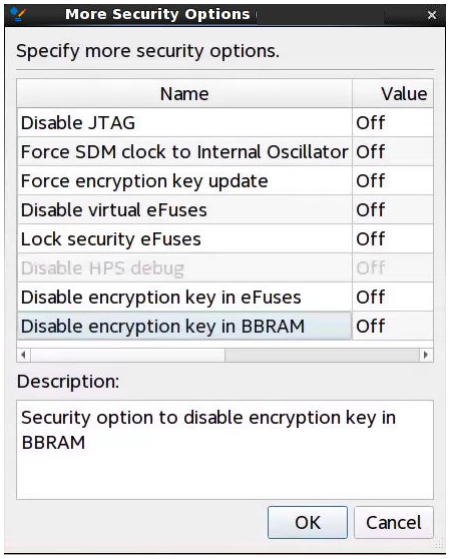


Table 8. More Security Options Dialog Box Settings

Option	Description	Values
Disable JTAG	Disables JTAG command and configuration of the device. Setting this eliminates JTAG as mode of attack, but also eliminates boundary scan functionality.	• **Off**—inactive • **On**—active until wipe of containing design • **On sticky**—active until next POR • **On check**—checks for corresponding blown fuse
Force SDM clock to internal oscillator	Disables an external clock source for the SDM. The SDM must use the internal oscillator. Using an internal oscillator is more secure than allowing an external clock source for configuration.	
Force encryption key update	Specifies that the encryption key must update by the frequency that you specify for the **Encryption update ratio** option. The default ration value is 31:1. Encryption supports up to 20 intermediate keys.	
Disable virtual eFuses	Disables the eFuse virtual programming capability.	
Lock security eFuses	Causes eFuse failure if the eFuse CRC does not match the calculated value.	

continued...

Option	Description	Values
Disable HPS debug	Disables debugging through the JTAG interface to access the HPS.	
Disable encryption key in eFuses	Specifies that the device cannot use an AES key stored in eFuses. Rather, you can provide an extra level of security by storing the AES key in BBRAM.	
Disable encryption key in BBRAM	Specifies that the device cannot use AES key stored in BBRAM. Rather, you can provide an extra level of security when you store the AES key in eFuses.	

4. Click **OK**.

Related Information

Enabling Bitstream Authentication (Programming File Generator) on page 20

1.3.3. Enabling Bitstream Encryption (Programming File Generator)

To enable bitstream encryption, you must first generate a first level signature chain (`.qky`) that enables encryption options in the GUI. Next, you generate the encrypted configuration bitstream in the Assembler. Finally, you generate a secondary programming file that specifies the AES **Encryption Key file** (`.qek`) for bitstream decryption.

Follow these steps to enable bitstream encryption:

1. Generate a First Level Signature Chain that includes the root key and one or more design signing keys, as *Stratix 10 Device Security User Guide* and *Agilex 7 Device Security User Guide* describe.
2. Click **Assignments ➤ Device ➤ Device and Pin Options ➤ Security**.
3. For the **Quartus key file** setting, specify the first level signature chain `.qky` that contains the root key and one or more design signing keys.
4. Turn on **Enable programming bitstream encryption**, and specify one or more of the following:

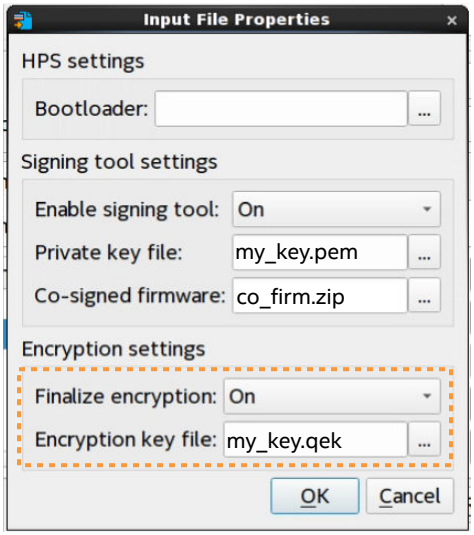
Table 9. Assembler Encryption Security Settings

Option	Description
Encryption key storage select	Specifies the location that stores the `.qek` key file. You can select either **Battery Backup RAM** or **eFuses** for storage.
Encryption update ratio	Specifies the ratio of configuration bits compared to the number of key updates required for bitstream decryption. You can select either **31:1** (the key must change 1 time every 31 bits) or **Disabled** (no update required). Encryption supports up to 20 intermediate keys.
Enable scrambling	Scrambles the configuration bitstream.
More Options	Opens the **More Security Options** dialog box for specifying additional physical security options.

5. Generate primary device programing files in the Assembler, as Generating Primary Device Programming Files on page 5 describes.
6. Generate a `.jic` or `.rbf` secondary programming file, as Generating Secondary Programming Files on page 6 describes:
   a. In the **Programming File Generator**, select the `.sof` file on the **Input Files** tab.

b. Click the **Properties** button. The **Input File Properties** dialog box appears.

Figure 17. Input File Properties



c. Set **Finalize encryption** to **On**.

d. Specify the AES 256-bit or 384-bit **Encryption key file** (`.qek`) to decrypt the bitstream in the SDM prior to device configuration.

7. Click **OK**.

Related Information

Input Files Tab Settings (Programming File Generator) on page 64

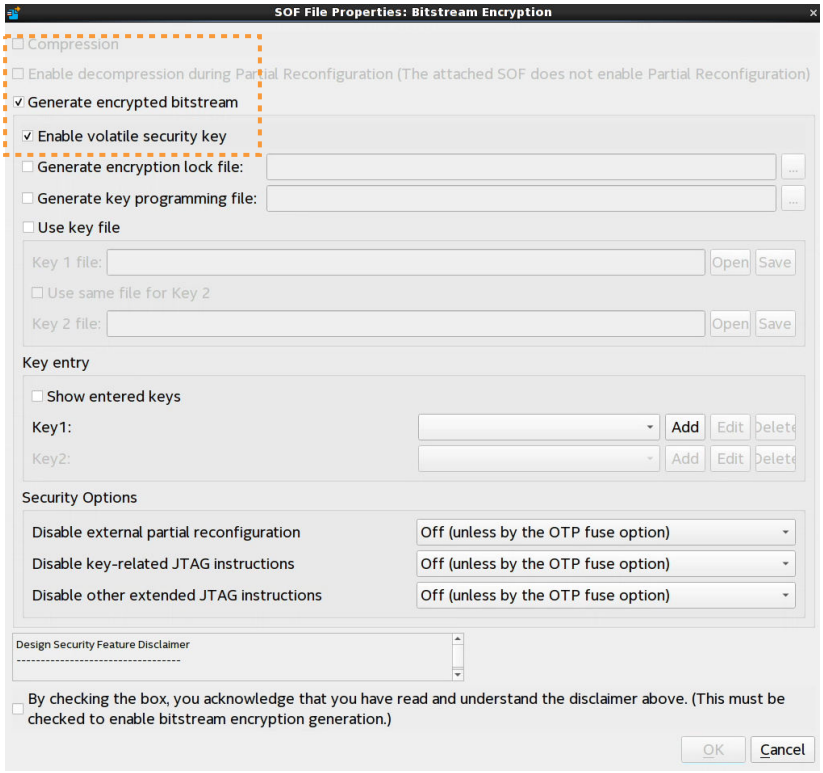
1.4. Enabling Bitstream Encryption or Compression for Arria 10 and Cyclone 10 GX Devices

You can optionally enable bitstream file encryption that requires a user-defined 256-bit security key to access the configuration bitstream. Alternatively, you can enable bitstream compression to reduces the size of your programming file to minimize file transfer and storage requirements. The compression reduces configuration file size by 30% to 55% (depending on the design). File compression and encryption options are mutually exclusive for Arria 10 and Cyclone 10 GX devices.

Follow these steps to enable bitstream file compression or encryption for Arria 10 and Cyclone 10 GX devices:

1. Generate a `.jic` file for flash programming, as this document describes.
2. In the **Convert Programming File** dialog box, select the `.sof` file under **Input files to convert**.
3. Click the **Properties** button. The **SOF File Properties: Bitstream Encryption** dialog box appears.

Figure 18. Enabling Bitstream Compression or Encryption (Intel Arria 10 and Intel Cyclone 10 GX Designs)



4. To enable compression, turn on the **Compression** option. All encryption options disable as these options are mutually exclusive.
5. To enable bitstream file encryption:
   a. Turn off the **Compression** option.
   b. Turn on the **Generate encrypted bitstream** option.
   c. Specify options for programming file key decryption, and **Security Options**, as Compression and Encryption Settings (Convert Programming File) on page 67 describes.
6. Click **OK**.

Related Information

- Arria 10 Core Fabric and General Purpose I/Os Handbook
  For detailed device security configuration steps.
- Cyclone 10 GX Core Fabric and General Purpose I/Os Handbook
  For detailed device security configuration steps.

1.5. Generating Programming Files for Partial Reconfiguration

The following sections describe generation of bitstream and other files for partial reconfiguration.

1.5.1. Generating PR Bitstream Files

For Agilex 7, Agilex 5, and Stratix 10 designs, the Assembler generates a configuration `.rbf` automatically at the end of compilation. For Arria 10 and Cyclone 10 GX designs, use any of the following methods to process the PR bitstreams and generate the Raw Binary File (`.rbf`) file for reconfiguration.

Generating PR Bitstreams During Compilation

Follow these steps to generate the `.rbf` file during compilation for Arria 10 and Cyclone 10 GX designs:

1. Add the following assignments to the revision `.qsf` to automatically generate the required PR bitstreams following compilation:

```
set_global_assignment -name GENERATE_PR_RBF_FILE ON
set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
```

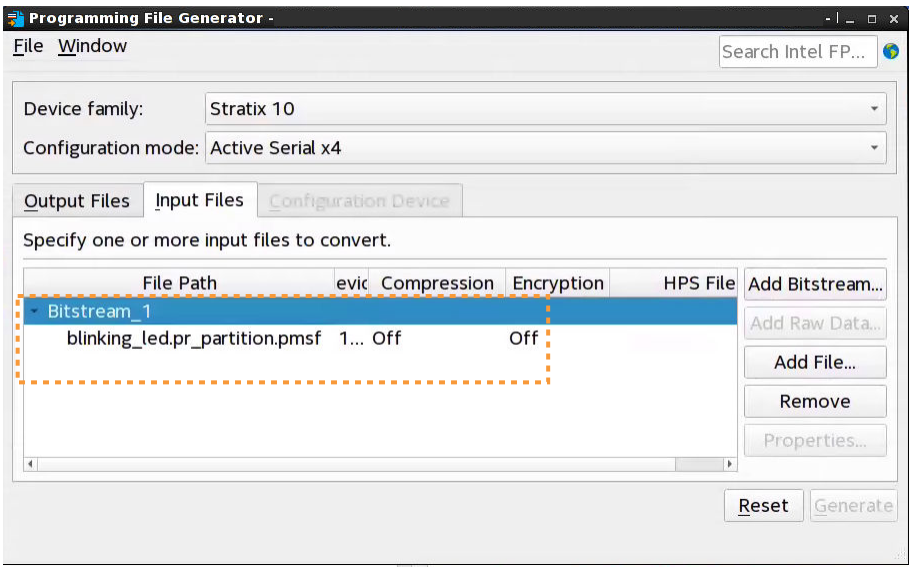
2. To compile the revision and generate the `.rbf`, click **Processing ➤ Start Compilation**.

Generating PR Bitstreams with Programming File Generator

Follow these steps to generate the `.rbf` for PR programming with the **Programming File Generator**:

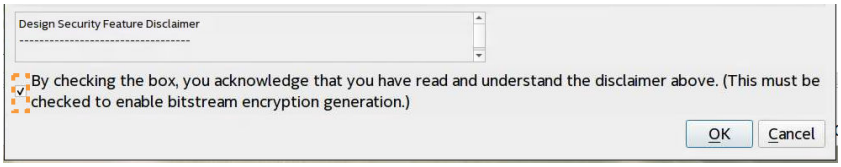
1. Click **File ➤ Programming File Generator**. The **Programming File Generator** appears.
2. Specify the target **Device family** and the **Configuration mode** for partial reconfiguration.
3. On the **Output File** tab, specify the **Output directory**, file **name**, and enable the **Raw Binary File for Partial Reconfiguration (.rbf)** file type.
4. To add the input `.pmsf` file to convert, click the **Input Files** tab, click **Add Bitstream**, and specify the `.pmsf` that you generated in the Assembler.

Figure 19. Adding Bitstream File



5. On the **Input Files** tab, select the bitstream `.pmsf` file and click **Properties**. Specify any of the following options for the `.rbf`:
   - **Enable compression**—generates compressed PR bitstream files to reduce file size.
   - **Enable encryption**—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (`.ekp`). You can also specify other **Security settings**.
   - If you turn on **Enable encryption**, you must also acknowledge the **Design Security Feature Disclaimer** by checking the box.

Figure 20. Design Security Feature Disclaimer



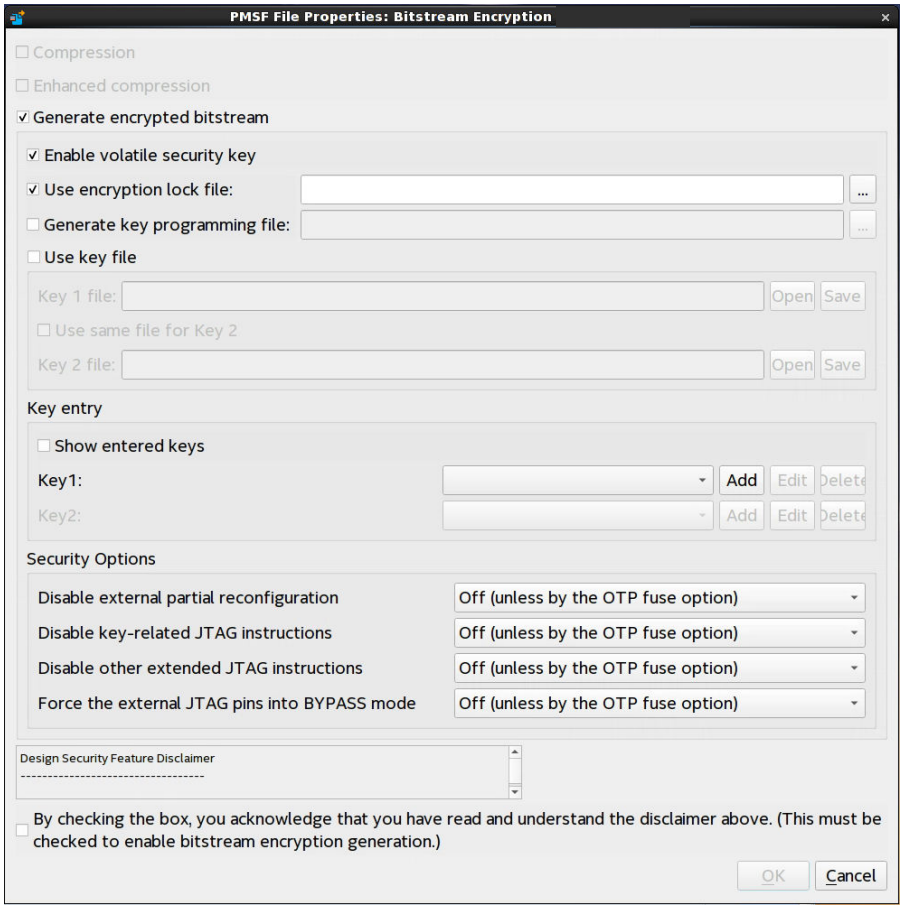
6. Click **OK**.
7. In **Programming File Generator**, click **Generate**. The PR bitstream files generate according to your specifications.

Generating PR Bitstreams with Convert Programming Files Dialog Box

Follow these steps to generate the `.rbf` with the **Convert Programming Files** dialog box:

1. Click **File ➤ Convert Programming Files**. The **Convert Programming Files** dialog box appears.
2. Specify the output file name and **Programming file type** as **Raw Binary File for Partial Reconfiguration (.rbf)**.
3. To add the input `.pmsf` file to convert, click **Add File**.
4. Select the newly added `.pmsf` file, and click **Properties**.
5. Enable or disable any of the following options and click **OK**:
   - **Compression**—enables compression on PR bitstream.
   - **Enhanced compression**—enables enhanced compression on PR bitstream.
   - **Generate encrypted bitstream**—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (`.ekp`). If you enable **Generate encrypted bitstream**, enable or disable the **Enable volatile security key**, **Use encryption lock file**, and **Generate key programming file** options.
6. Click **Generate**. The PR bitstream files generate according to your specifications.

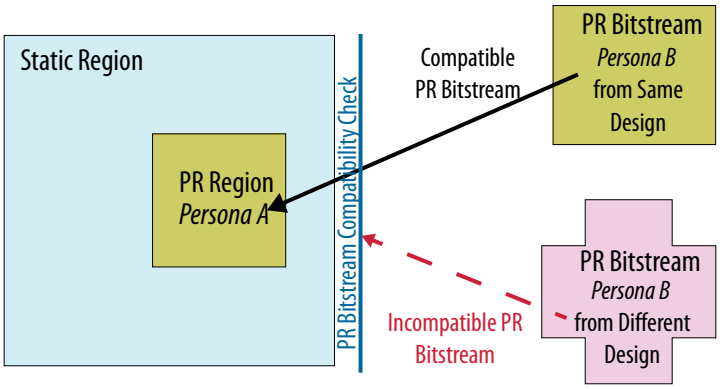
Figure 21. PMSF File Properties Bitstream Encryption



1.5.2. Partial Reconfiguration Bitstream Compatibility Checking

Partial reconfiguration bitstream compatibility checking verifies the compatibility of the reconfiguration bitstream to prevent configuration with an incompatible PR bitstream. The following sections describe PR bitstream compatibility check support.

Figure 22. PR Bitstream Compatibility Checking



PR Bitstream Compatibility Checking for Agilex 7, Agilex 5, and Stratix 10 Designs

For Agilex 7, Agilex 5, and Stratix 10 designs, PR bitstream compatibility checking is automatically enabled in the Compiler and in the Secure Device Manager (SDM) firmware by default. The following limitations apply to PR designs *if PR bitstream compatibility checking is enabled*:

- The firmware allows up to a total of 32 PR regions, irrespective of the number of hierarchical partial reconfiguration layers.
- Your PR design can have up to six hierarchical partial reconfiguration layers.
- Your PR design, when there is no hierarchy, can have up to 32 regions.
- Your PR design can have up to 15 child PR regions of any parent PR region (if it is hierarchical). Child PR regions count towards the total limit of 32 PR regions.

The Compiler generates an error if your PR design exceeds these limits when PR bitstream compatibility checking is enabled.

If you require more PR regions than this limitation allows, or otherwise want to disable PR bitstream compatibility checking , you can add the following assignment to the `.qsf` file:

```
set_global_assignment -name ENABLE_PR_POF_ID OFF
```

When you set this assignment to off, the limit of 32 total regions does not apply in the Compiler.

Note: If you require the PR bitstream authentication feature for your design, you must enable PR bitstream compatibility checking by setting the global assignment `ENABLE_PR_POF ID` to `ON`. The default setting is `ON`.

Arria 10 and Cyclone 10 GX PR Bitstream Compatibility Checking

For Arria 10 and Cyclone 10 GX designs, you enable or disable PR bitstream compatibility checking by turning on the **Enable bitstream compatibility check** option when instantiating the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP from the IP Catalog.

The PR IP verifies the partial reconfiguration PR Bitstream file (`.rbf`). When you enable the bitstream compatibility check, the PR `.pof` ID is encoded as the 71st word of the PR bitstream. If the PR IP detects an incompatible bitstream, then the PR IP stops the PR operation, and the status output reports an error.

When you turn on **Enable bitstream compatibility check**, the PR Controller IP core creates a **PR bitstream ID** and displays the bitstream ID in the configuration dialog box. For bitstream compatibility checking with hierarchical PR designs, refer to additional steps in *AN 806: Hierarchical Partial Reconfiguration Tutorial for Arria 10 GX FPGA Development Board*.

Related Information

AN 806: Hierarchical Partial Reconfiguration Tutorial for Intel Arria 10 GX FPGA Development Board

1.5.3. Raw Binary Programming File Byte Sequence Transmission Examples

The raw binary programming file (`.rbf`) file contains the device configuration data in little-endian raw binary format. The following example shows transmitting the `.rbf` byte sequence 02 1B EE 01 in x32 mode:

Table 10. Writing to the PR control block or SDM in x32 mode

In x32 mode, the first byte in the file is the least significant byte of the configuration double word, and the fourth byte is the most significant byte.

Double Word = 01EE1B02			
LSB: BYTE0 = 02	BYTE1 = 1B	BYTE2 = EE	MSB: BYTE3 = 01
D[7..0]	D[15..8]	D[23..16]	D[31..24]
0000 0010	0001 1011	1110 1110	0000 0001

1.5.4. Generating a Merged .pmsf File from Multiple .pmsf Files (Arria 10 and Cyclone 10 GX Designs)

Use a single merged `.rbf` file to reconfigure two PR regions simultaneously.

Note: This procedure supports only Arria 10 and Cyclone 10 GX devices. Agilex 7, Agilex 5, and Stratix 10 devices do not support merging `.pmsf` files.

To merge two or more `.pmsf` files:

1. Open the **Convert Programming Files** dialog box.
2. Specify the output file name and programming file type as **Merged Partial-Mask SRAM Object File (.pmsf)**.
3. In the **Input files to convert** dialog box, select **PMSF Data**.
4. To add input files, click **Add File**. You must specify two or more files for merging.
5. To generate the merged file, click **Generate**.

Alternatively, to merge two or more `.pmsf` files from the Quartus Prime shell, type the following command:

```
quartus_cpf --merge_pmsf=<number of merged files> <pmsf_input_file_1> \
     <pmsf_input_file_2> <pmsf_input_file_etc> <pmsf_output_file>
```

For example, to merge two `.pmsf` files, type the following command:

```
quartus_cpf --merge_pmsf=2 foo.pmsf bat.pmsf \
     combine.pmsf
```

After creating the merged `.pmsf`, generate a `.rbf`, as Generating PR Bitstream Files on page 26 describes.

1.6. Generating Programming Files for Intel FPGA Devices with Hard Processor Systems

When generating programming files for Intel FPGA devices with a Hard Processor System (HPS), you must first determine what boot flow you want to use for the device: HPS Boot First or FPGA Configuration First.

Depending on the boot flow that you want to use, follow the instructions in one of the following sections:

- Generating Programming Files for HPS Boot First Boot Flows on page 31
- Generating Programming Files for FPGA Configuration First Boot Flows on page 34

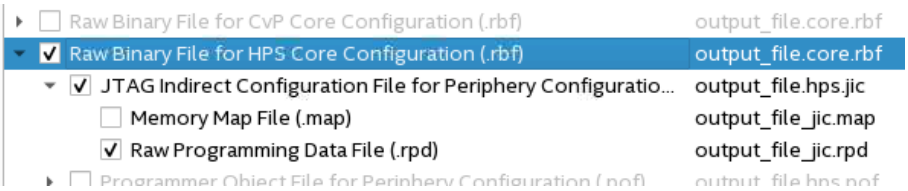
1.6.1. Generating Programming Files for HPS Boot First Boot Flows

In HPS Boot First boot flows, the HPS I/O and EMIF are configured and the HPS is booted before configuring the FPGA I/O and core.

Configuring the HPS I/O for the first time and then loading the HPS FSBL is called "Phase 1 configuration". The subsequent configuration of FPGA I/O and core by HPS is called "Phase 2 configuration".

To generate programming files for HPS Boot First boot flows:

1. Generate the primary programming files for your design, as Generating Primary Device Programming Files on page 5 describes.
2. Click **File ➤ Programming File Generator**.
3. For **Device family**, select your target device. The options available in the **Programming File Generator** change dynamically, according to your device and configuration mode selection.
4. For **Configuration mode**, select an Active Serial mode that your device supports. Configuration Modes (Programming File Generator) on page 9 describes all modes.
5. On the **Output Files** tab, select **Raw Binary File for HPS Core Configuration (.rbf)**, then select the following files:
   - **JTAG Indirect Configuration File for Periphery Configuration (.jic)**
   - **Raw Programming Data File (.rpd)**

   Secondary Programming Files (Programming File Generator) on page 10 describes all output files.
6. On the **Output Files** tab, select **Raw Programming Data File (.rpd)** and click **Edit**.
7. Optional: **Optional:** In the **RPD Properties** dialog box, set **Bit swap** to **On**.

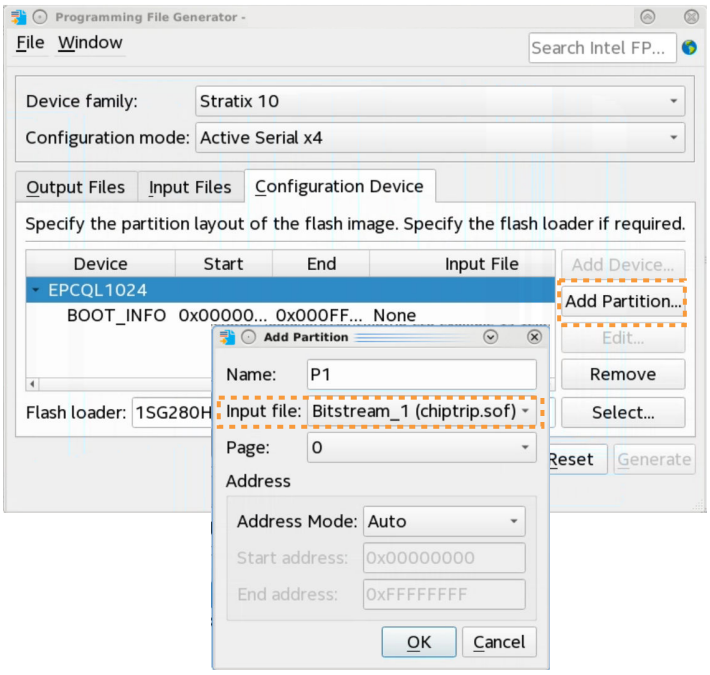
   Important: This step may or may not be required, depending on the external programmer that you use.

8. Specify the **Output directory** and **Name** for the file you generate. Output Files Tab Settings (Programming File Generator) on page 63 describes all options.
9. On the **Input Files** tab, click **Add Bitstream** to add your `.sof` file or files.
10. For each `.sof` file that you add, edit their properties as follows:
    a. Select the `.sof` file and click **Properties**.
    b. In the **Bootloader** field of **Input File Properties** dialog box, add the U-Boot First State Boot Loader (FSBL) file. Ensure that the file is an Intel-format hexadecimal (`.hex`) file.
11. To add other data, such as U-Boot Second Stage Boot Loader (SSBL) file or Phase 2 bitstreams:
    a. Click **Add Raw Data** and specify an Intel-format hexadecimal (`.hex`) file.
    b. Select the file you added and click **Properties**.
    c. In the **Input File Properties** dialog box, set the **Bit swap** field to **On**.

    Important: Your Phase 1 and Phase 2 bitstreams are subject to the following restrictions:
    - Your Phase 1 and Phase 2 bitstreams must be generated by the same version of Quartus Prime, including any applied patches or updates.
    - If your Phase 1 and Phase 2 bitstreams are generated from different Quartus Prime Pro Edition projects, review the following Knowledge Base article for additional steps that might be required:

      Why does FPGA configuration fail from Linux / u-boot fail on Stratix 10 10 SX devices when I use phase 2 bitstreams generated from different Quartus Projects?
12. To specify the `.sof` file that occupies the flash memory partition, click **Add Partition** on the **Configuration Device** tab. Add Partition Dialog Box (Programming File Generator) on page 66 describes all options.

Figure 23. Add Flash Partition



13. To select a supported flash memory device and predefined programming flow, click **Add Device** on the **Configuration Device** tab. Alternatively, click **<<new device>>** to define a new flash memory device and programming flow. Configuration Device Tab Settings on page 65 describes all settings.
14. Click the **Select** button for **Flash Loader** and select the device that controls loading of the flash memory device. Select Devices (Flash Loader) Dialog Box on page 69 describes all settings.
15. After you specify all options in **Programming File Generator**, the **Generate** button enables. Click **Generate** to create the files.
16. **Optional:** Export your settings to PFG setting file (`.pfg`) so that you can use these settings again with the `quartus_pfg` command line tool.

    For details, refer to quartus_pfg Command Line Tool on page 37.

Related Information

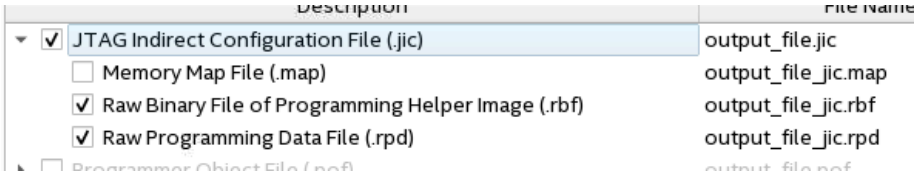
- Device Configuration User Guide: Agilex 5 FPGAs and SoCs
- Agilex 7 Configuration User Guide
- Stratix 10 Configuration User Guide
- Agilex 7 SoC FPGA Boot User Guide
- Stratix 10 SoC FPGA Boot User Guide
- Arria 10 SoC FPGA Boot User Guide
- Agilex 7 Hard Processor System Remote System Update User Guide
- Stratix 10 Hard Processor System Remote System Update User Guide

1.6.2. Generating Programming Files for FPGA Configuration First Boot Flows

In FPGA Configuration First boot flows, the FPGA core and periphery are configured first. After that, the HPS can optionally be booted.

To generate programming files for FPGA Configuration First boot flows

1. Generate the primary programming files for your design, as Generating Primary Device Programming Files on page 5 describes.
2. Click **File ➤ Programming File Generator**.
3. For **Device family**, select your target device. The options available in the **Programming File Generator** change dynamically, according to your device and configuration mode selection.
4. For **Configuration mode**, select an Active Serial mode that your device supports. Configuration Modes (Programming File Generator) on page 9 describes all modes.
5. On the **Output Files** tab, select **JTAG Indirect Configuration File (.jic)**, then select the following files:
   - **Raw Binary File of Programming Helper Image (.rbf)**
   - **Raw Programming Data File (.rpd)**



   Secondary Programming Files (Programming File Generator) on page 10 describes all output files.
6. On the **Output Files** tab, select **Raw Programming Data File (.rpd)** and click **Edit**.
7. Optional: **Optional:** In the **RPD Properties** dialog box, set **Bit swap** to **On**.

   Important: This step may or may not be required, depending on the external programmer that you use.
8. Specify the **Output directory** and **Name** for the file you generate. Output Files Tab Settings (Programming File Generator) on page 63 describes all options.
9. On the **Input Files** tab, click **Add Bitstream** to add your `.sof` file and then edit its properties:
   a. Select the `.sof` file and click **Properties**.
   b. In the **Bootloader** field of **Input File Properties** dialog box, add the U-Boot First State Boot Loader (FSBL) hex file (`.hex`).
10. To add other data, such as a U-Boot Second Stage Boot Loader (SSBL) file or Phase 2 bitstreams:
    a. Click **Add Raw Data** and specify an Intel-format hexadecimal (`.hex`) file.
    b. Select the file you added and click **Properties**.
    c. In the **Input File Properties** dialog box, set the **Bit swap** field to **On**.

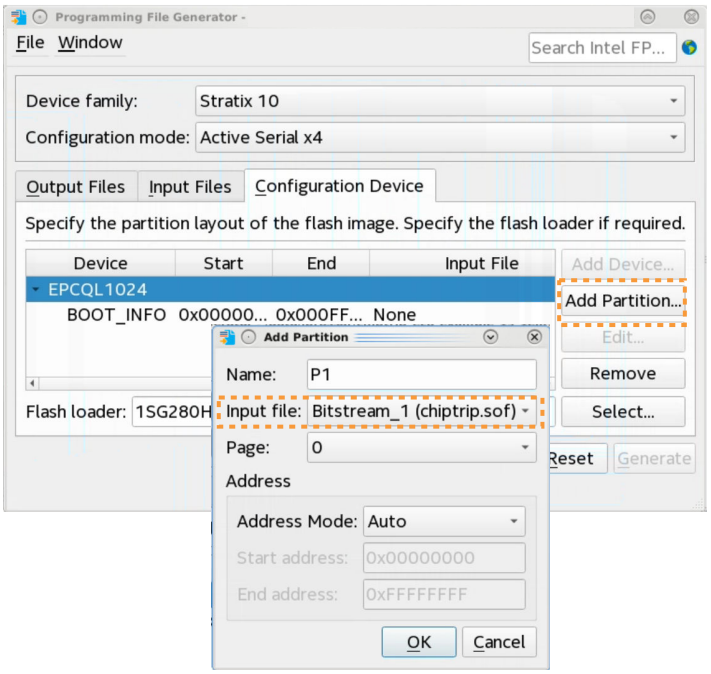
Important: Your Phase 1 and Phase 2 bitstreams are subject to the following restrictions:

- Your Phase 1 and Phase 2 bitstreams must be generated by the same version of Quartus Prime, including any applied patches or updates.
- If your Phase 1 and Phase 2 bitstreams are generated from different Quartus Prime Pro Edition projects, review the following Knowledge Base article for additional steps that might be required:

  Why does FPGA configuration fail from Linux / u-boot fail on Stratix 10 SX devices when I use phase 2 bitstreams generated from different Quartus Projects?

11. To specify the `.sof` file that occupies the flash memory partition, click **Add Partition** on the **Configuration Device** tab. Add Partition Dialog Box (Programming File Generator) on page 66 describes all options.

Figure 24. Add Flash Partition



12. To select a supported flash memory device and predefined programming flow, click **Add Device** on the **Configuration Device** tab. Alternatively, click **<<new device>>** to define a new flash memory device and programming flow. Configuration Device Tab Settings on page 65 describes all settings.
13. Click the **Select** button for **Flash Loader** and select the device that controls loading of the flash memory device. Select Devices (Flash Loader) Dialog Box on page 69 describes all settings.
14. After you specify all options in **Programming File Generator**, the **Generate** button enables. Click **Generate** to create the files.
15. Optional: **Optional:** Export your settings to a PFG setting file (`.pfg`) so that you can use these settings again with the `quartus_pfg` command line tool.

For details, refer to quartus_pfg Command Line Tool on page 37.

Related Information

- Device Configuration User Guide: Agilex 5 FPGAs and SoCs
- Agilex 7 Configuration User Guide
- Stratix 10 Configuration User Guide
- Agilex 7 SoC FPGA Boot User Guide
- Stratix 10 SoC FPGA Boot User Guide
- Arria 10 SoC FPGA Boot User Guide
- Agilex 7 Hard Processor System Remote System Update User Guide
- Stratix 10 Hard Processor System Remote System Update User Guide

1.7. Scripting Support

The Quartus Prime software allows generating programming files from the command line. You can incorporate these commands to scripted flows.

1.7.1. quartus_pfg Command Line Tool

The Programing File Generator is also available as the `quartus_pfg` executable. You can specify conversion settings in the command line or through a PFG setting file (`.pfg`). This ability is useful for advanced designs that require multiple images or multiple user data files (HEX/RBF), because you define the settings once in the GUI and then export for subsequent use in the command line.

By default, `quartus_pfg` converts files using the AVSTx8 configuration scheme. To use a different flow, specify the proper operation mode, as the following command shows:

```
quartus_pfg -c  -o device=MT25QU512 -o mode=ASX4 -o flash_loader=1SG280HN3S3 \
      project.sof project.jic
```

To export PFG settings to a `.pfg` file, click **File ➤ Save**. The Programming File Generator only saves settings that are consistent.

For more information about the `quartus_pfg` executable, type the following in the command line:

```
quartus_pfg --help
```

Differences Between GUI and Command Line Tool

The command line tool supports single image conversion only.

1.7.2. quartus_cpf Command Line Tool

The Convert Programming Files tool is also available as the `quartus_cpf` command line executable. You can specify conversion settings in the command line or with a conversion setup file (`.cof`).

For help with the `quartus_cpf` executable, type the following at the command line:

```
quartus_cpf --help
```

1.7.2.1. Generating a Partial-Mask SRAM Object File using a Mask Settings File and a SRAM Object File

- To generate a `.pmsf` file with the `quartus_cpf` executable, type the following in the command line:

  ```
  quartus_cpf -p <pr_revision.msf> <pr_revision.sof> <new_filename.pmsf>
  ```

Note: The `-p` option is available for designs targeting Arria 10 and Cyclone 10 GX device families.

Related Information

Quartus Prime Pro Edition User Guide: Partial Reconfiguration
In *Quartus Prime Pro Edition User Guide: Partial Reconfiguration*

1.8. Generating Programming Files Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Added support for Agilex 5 devices. • Updated the description of map file in *Secondary Programming Files (Programming File Generator)*.
2023.04.03	23.1	• Updated location of device information directory in *Enabling Bitstream Encryption or Compression for Arria 10 and Cyclone 10 GX Devices*. • Updated product family name to "Intel Agilex 7."
2022.09.26	22.3	• Updated *Generating Secondary Programming Files (Programming File Generator)*. • *Updated Generating Secondary Programming Files (Settings: Programming Files Dialog Box).* • Removed footnotes saying that security features are not available for Agilex 7 devices. For details about security features for Agilex 7 devices, refer to the *Agilex 7 Device Security User Guide*.
2021.10.04	21.3	• Added *Generating Secondary Programming Files (Settings: Programming Files Dialog Box)* topic.
2020.12.14	20.4	• Added new *Generating Programming Files for the HPS Flash Programmer* section.
2020.10.13	20.3	• Added more details about alternate configuration schemes to *quartus_pfg Command Line Tool* topic.
2020.05.08	19.4	• Added note about programming file differences to *Generating Primary Device Programming Files* topic.
2019.12.16	19.4	• Added programming file generation support for Agilex 7 devices. • Noted Agilex 7 security feature limitations.
2019.09.30	19.3	• Added new "Enabling Bitstream Security for Intel Stratix 10 Devices" topic. • Added new "Enabling Bitstream Authentication (Programming File Generator)" topic. • Added new "Specifying Additional Physical Security Settings (Programming File Generator)" topic. • Added new "Enabling Bitstream Encryption (Programming File Generator)" topic. • Updated name of "Authentication and Encryption" tab to "Security" tab. • Added footnote about programming file support for Agilex 7 devices. • Described new **More Security Settings** dialog box.
2019.06.10	19.1	• Added links to *Generic Flash Programmer User Guide*. • Added flash programming details to "Generating Secondary Programming Files" and created separate topics for **Programming File Generator** and **Convert Programming Files** dialog box. • Added new "Enabling Bitstream Encryption or Co-Signing (Programming File Generator)" topic. • Added new "Enabling Bitstream Compression or Encryption (Convert Programming File)" topic. • Updated screenshots for latest GUI.
2019.04.01	19.1	• Retitled and reorganized topics to improve flow of information. • Added "Programming File Generator Configuration Modes" topic. • Added "Convert Programming File Configuration Modes" topic. • Added "Generating Programming Files for Partial Reconfiguration." • Added "Generating PR Bitstreams Files."

continued...

Document Version	Quartus Prime Version	Changes
		• Added "Partial Reconfiguration Bitstream Compatibility Checking." • Added "Raw Binary Programming File Byte Sequence Transmission Examples." • Added "Generating a Merged .pmsf File from Multiple .pmsf Files."
2018.10.09	18.1	• Added MAX V to the list of devices that the Programming File Generator tool supports. • Added table : *Device Families that the Convert Programming Files Tool Supports*.
2018.09.24	18.1	• Added topic: *quartus_cpf Command Line Tool*. • Stated that the Convert Programming Files dialog box is a legacy tool that supports file conversion for older device families. • In topic: *Output File Types*, specified that the list includes file types generated by the Converting Programming Files tool.
2018.08.07	18.0	Reverted document title to *Programmer User Guide: Intel Quartus Prime Pro Edition*.
2018.06.27	18.0	• Created the new chapter with information from the *Programming Devices* chapter. • Included information about the Programming File Generator tool.

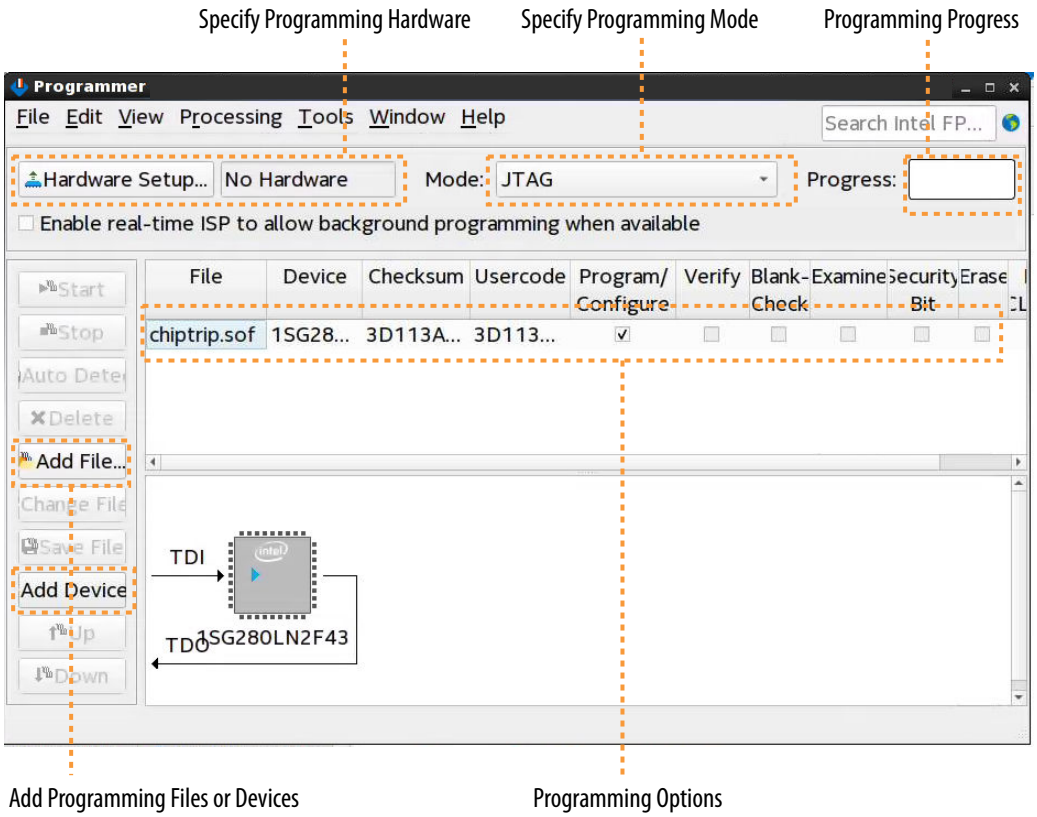
2. Using the Quartus Prime Programmer

Use the Quartus Prime Programmer and connected communication cable to program or configure Intel CPLD, FPGA, and configuration devices. This chapter describes how to setup and use the Quartus Prime Programmer.

2.1. Quartus Prime Programmer

Access the integrated Programmer by clicking **Tools ➤ Programmer** in the Quartus Prime software.

Figure 25. Quartus Prime Programmer



Prior to programming or configuration, you generate and specify the primary programming files, setup the programming hardware, and set the configuration mode in the Programmer.

2.2. Programming and Configuration Modes

The current version of the Quartus Prime Programmer supports the following programming and configuration modes in the Programmer's **Mode** list. Select a configuration mode to setup and run that type of programming or configuration.

Table 11. Programming and Configuration Modes

Programming or Configuration Mode	Description
JTAG	A configuration method that configures one or more devices through the Joint Test Action Group (JTAG) Boundary-Scan Test (BST) circuitry.
In-Socket Programming	Configuration device programming or testing via the Altera Programming Unit (APU).
Passive Serial	An external controller passes configuration data to one or more configuration devices via a serial data stream. The device is treated as a slave device with a 5-wire interface to the external controller. The external controller can be an intelligent host such as a microcontroller or CPU, or the Quartus Prime Programmer. The external controller can also be a serial configuration device.
Active Serial Programming	The active serial memory interface block loads design data into one or more devices. The active serial memory interface block controls the configuration process, and configures all of the devices in the chain using the configuration data stored in an EPCS1, EPCS4, EPCS16, EPCS64, EPCQ, EPCQL, and third-party QSPI serial configuration devices.

2.3. Basic Device Configuration Steps

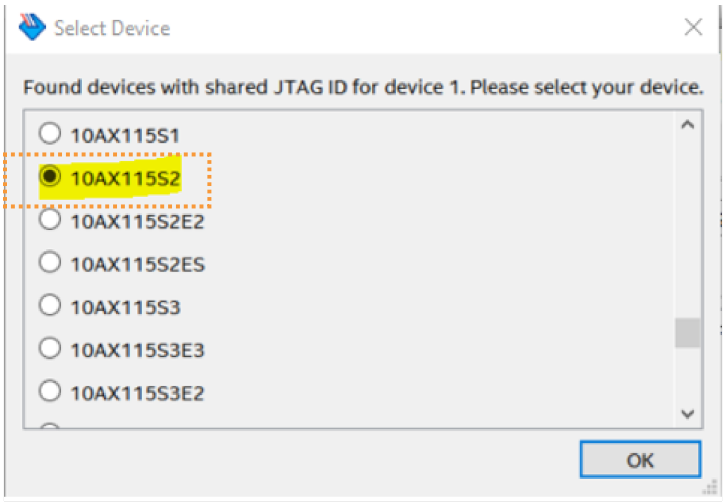
Basic FPGA Device Configuration over JTAG involves opening the Quartus Prime Programmer, connecting to a device on a development kit or board, and loading the configuration SRAM Object File (`.sof`) into the SRAM of the FPGA. The following steps describe the basic JTAG device configuration flow:

1. To run the Assembler to generate primary programming files, click **Processing ➤ Start ➤ Start Assembler**. The Compiler runs any prerequisite stages and generates programming files according to your specifications, as Generating Primary Device Programming Files on page 5 describes.
2. To open the Programmer, click **Tools ➤ Programmer**.
3. Connect the board cables. For JTAG device configuration, connect the JTAG USB cable to the board, and connect the power cable attached to the board to a power source.
4. Turn on power to the board.
5. In the Programmer, select **JTAG** for the programming **Mode**, as Programming and Configuration Modes on page 41 describes.
6. Click **Hardware Setup**. In the **Hardware** list, select connected programming hardware, as Specifying the Programming Hardware Setup on page 43 describes.

Figure 26. Hardware Setup

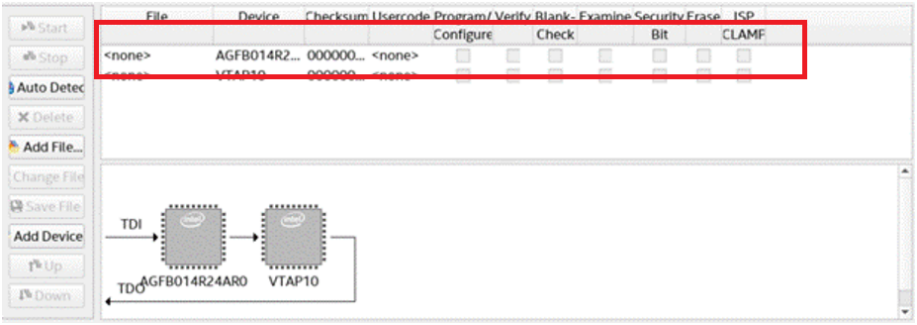
7. In the **Found Devices** list, select the device that matches your design and click **OK**.

Figure 27. Select Device



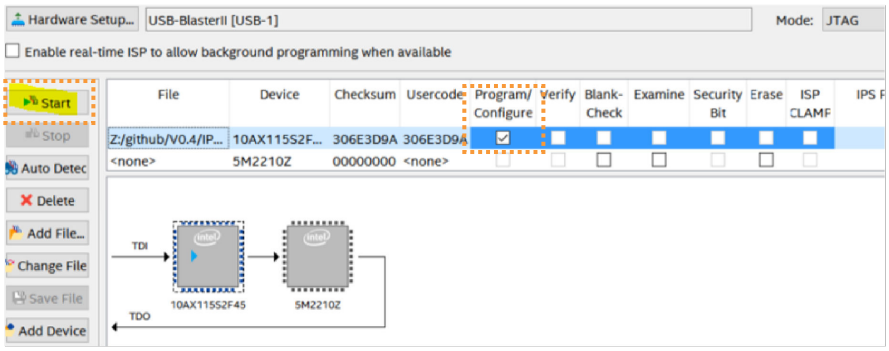
8. Right-click the row in the file list, and then click **Change File**.

Figure 28. Programmer Window



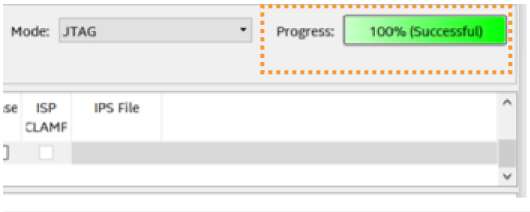
9. Browse to select the `.sof` file.
10. Enable the **Program/Configure** option for the row.

Figure 29. Program/Configure Option



11. Click **Start**. The progress bar reaches 100% when device configuration is complete. The device is now fully configured and in operation.

Figure 30. Programming Successful



Note: If device configuration fails, confirm that the device you select for configuration matches the device you specify during `.sof` file generation.

2.4. Specifying the Programming Hardware Setup

Before you can program or configure a device, you must specify an appropriate hardware setup. The Programmer's **Hardware Setup** dialog box allows you to add and remove programming hardware or JTAG servers from the current programming setup. You can specify a hardware setup for device programming or configuration, or configure a local JTAG server.

A JTAG server allows the Quartus Prime Programmer to access the JTAG programming hardware connected to a remote computer through the JTAG server of that computer. The JTAG server allows you to control the programming or configuration of devices from a single computer through other computers at remote locations. The JTAG server uses the TCP/IP communications protocol.

Selecting Device Programming Hardware

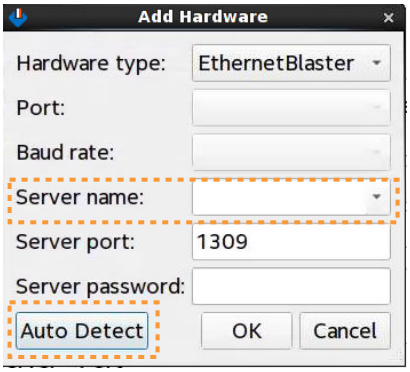
Follow these steps to select device programming hardware in the Programmer:

1. In the Programmer, click **Hardware Setup**.

Figure 31. Hardware Setup Dialog Box

2. To add new programming hardware, click **Add Hardware** on the **Hardware Settings** tab. In the **Add Hardware** dialog box, click **Auto Detect** to detect your programming hardware, or specify the properties of your programming hardware.

Figure 32. Add New Hardware



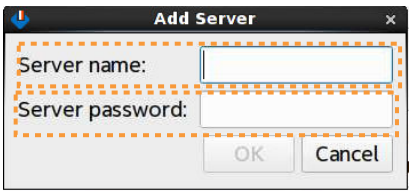
3. On the **Hardware Settings** tab, select your connected programming hardware in **Currently selected hardware**. This list is empty until you connect and add programming hardware to your system.
4. Enable or disable **Auto-adjust frequency at chain scanning** to automatically adjust the **Hardware frequency** according to the frequency at chain scanning.
5. Click **Close**. The setup appears as the current hardware setup.

Selecting a JTAG Server for Device Programming

Follow these steps to select a JTAG server for device programming in the Programmer:

1. In the Programmer, click **Hardware Setup**.
2. On the **JTAG Settings** tab, click **Add Server**. In the **JTAG Settings** dialog box, specify the **Server name** and **Server password**.

Figure 33. JTAG Settings



3. Under **JTAG Servers**, select the JTAG server that you want to access for programming.
4. Click **Close**. The setup appears as the current hardware setup.

2.4.1. JTAG Chain Debugger Tool

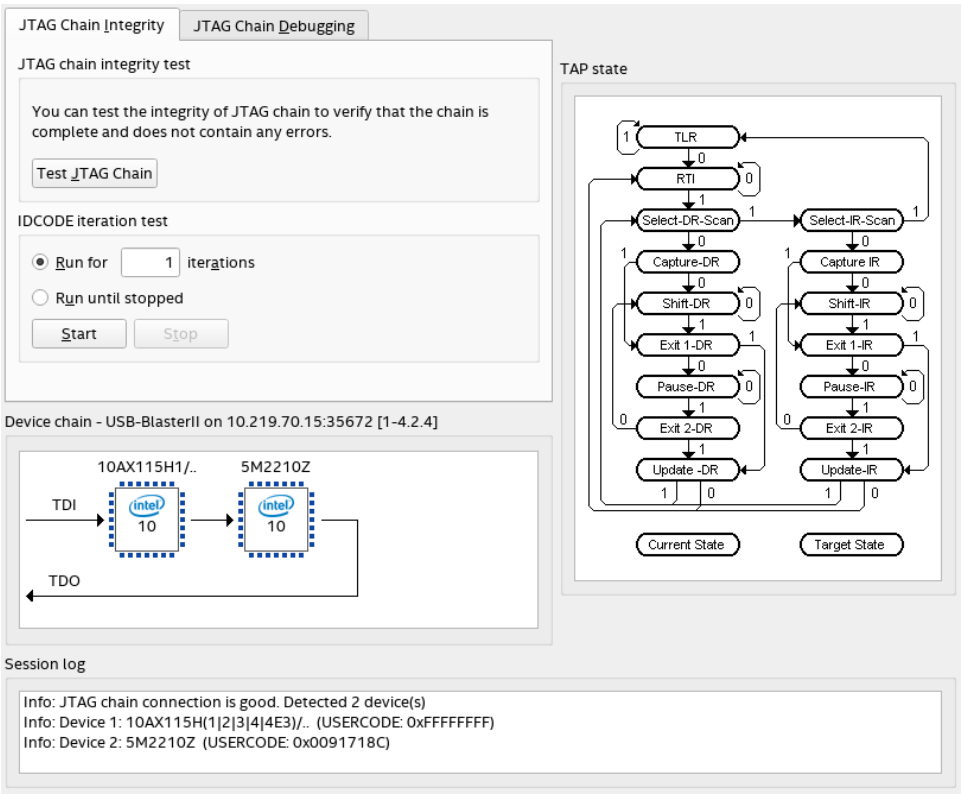
The JTAG Chain Debugger tool allows you to test the JTAG chain integrity and detect intermittent failures of the JTAG chain.

Access the tool by clicking **Tools ➤ JTAG Chain Debugger** in the Quartus Prime software.

Figure 34. JTAG Chain Debugger (Using an Arria 10 GX Development Kit)

The JTAG Chain Debugger has the following panes:

- **JTAG Chain Integrity** tab
- **JTAG Chain Debugging** tab
- **Device chain** pane
- **Tap state** pane
- **Session log** pane



The tool also allows you to shift in JTAG instructions and data through the JTAG interface, and step through the test access port (TAP) controller state machine for debugging purposes.

If the JTAG Chain Debugger Tool GUI is disabled when first you open it, enable the tool by selecting hardware:

1. In the JTAG Chain Debugger Tool GUI, select **Edit ➤ Hardware Setup**.
2. In the **Hardware Settings** tab of the **Hardware Setup** window, select a hardware device in the **Currently selected hardware** field.

If no hardware is available to select, add hardware by clicking **Add Hardware**.

Initially, the **Device chain** pane is empty. To populate the **Device chain** pane, right-click in the pane and select **Test JTAG chain**, or click **Test JTAG Chain** on the **JTAG Chain Integrity** tab.

When the **Device chain** pane is populated, you can activate one or all devices in the chain to run the test on the activated device or devices. To activate a device or all devices in the **Device chain** pane, the **JTAG Chain Debugging** tab must be selected.

Tests are run only on activated devices. Devices that are not activated are bypassed during testing.

You can save the contents of the **Session log** pane to repeat the session in the **JTAG Chain Debugging** tab, or you can clear the log:

- To save the log to replay later, right-click anywhere in the **Session log** pane and select **Save Session Log**.
- To clear the log, right-click anywhere in the **Session log** pane and select **Clear Session Log**.

Related Information

- Agilex 7 JTAG Boundary-Scan Testing User Guide
- Stratix 10 JTAG Boundary-Scan Testing User Guide
- JTAG Boundary-Scan Testing in Arria 10 Devices
- JTAG Boundary-Scan Testing in Cyclone 10 GX Devices

2.4.1.1. JTAG Chain Integrity Tab

Use the **JTAG Chain Integrity** tab to check the JTAG chain for potential failure, which can be caused by either a short circuit or an open JTAG circuit.

Figure 35. JTAG Chain Integrity Tab

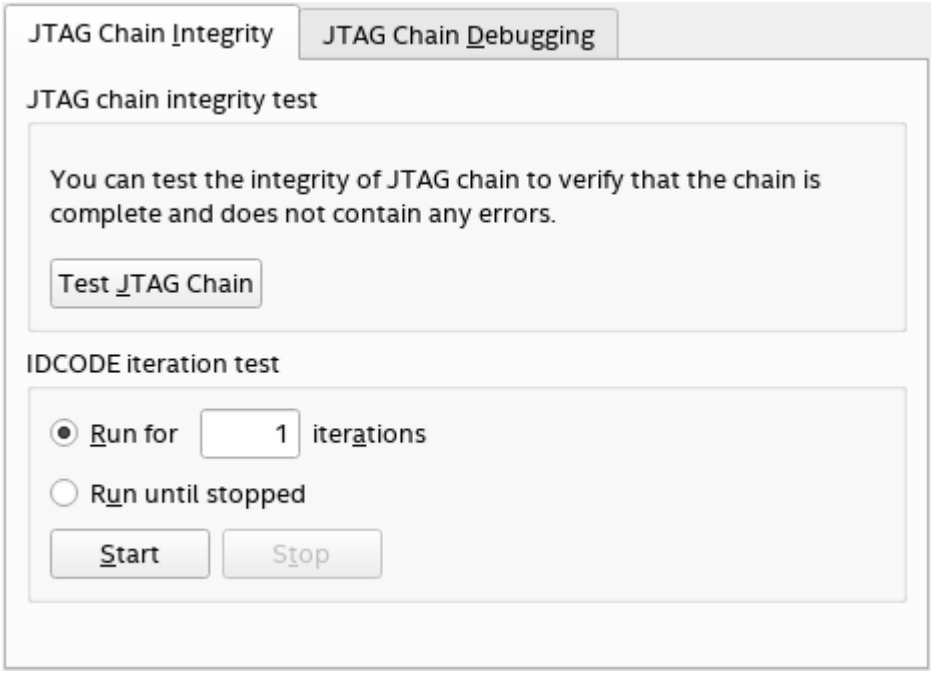


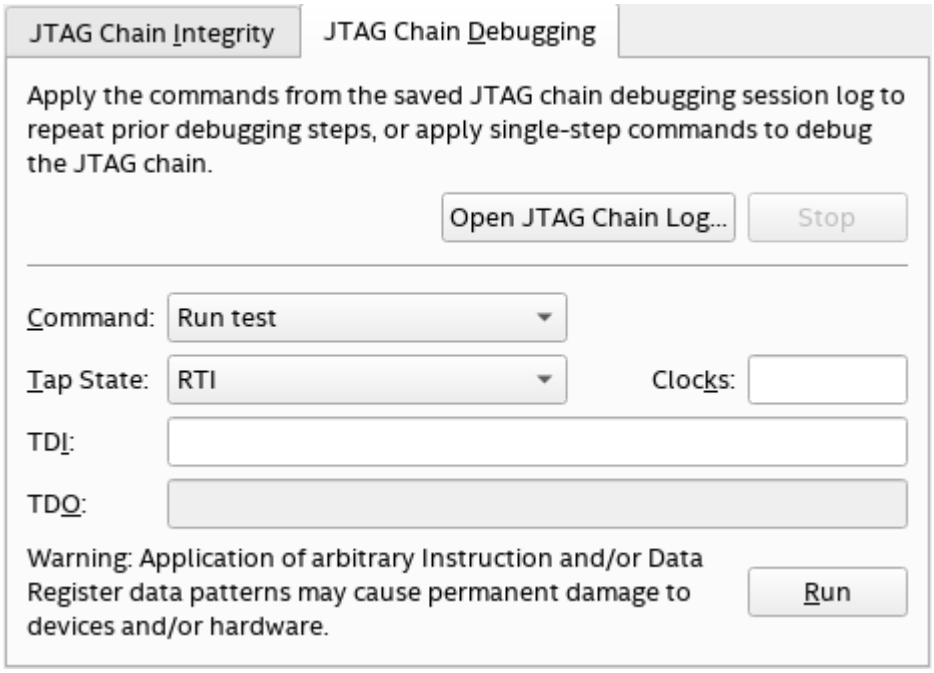
Table 12. JTAG Chain Integrity Tab

Pane	Description
JTAG chain integrity test	Run the Programmer **Auto Detect** feature, which detects devices in a chain and automatically adds supported devices to the **Device** list in the Programmer. The devices are added in the order in which they exist in the device chain. If the **Auto Detect** feature fails to detect the JTAG chain, click **Test JTAG Chain** to run a JTAG chain integrity test to analyze the failure.
IDCODE iteration test	Test for chain integrity and intermittent JTAG chain communication failure. This feature is turned off by default. • — **Run for**<*number*>**iterations**–specify the number of tests that you want to repeat. — **Run until stopped**–cycle the test repeatedly until you click **Stop**. The IDCODE iteration test cycles through Reset and Shift DR states to read IDCODEs from the device chain, and reports the following information: • When the IDCODEs read in each cycle of the test are the same, the result indicates that there are no inconsistencies or failures in the device chain. • When the IDCODEs read in each cycle are inconsistent, an error message is reported and the test terminates. Results from the IDCODE iteration test appear in text format in the **Session log** pane and in graphic format in the **Device chain** pane.

2.4.1.2. JTAG Chain Debugging Tab

Use the **JTAG Chain Debugging** tab to debug your JTAG chain by running commands to step through the state of the JTAG TAP controller. You can also repeat the commands from a previous session.

Figure 36. JTAG Chain Debugging Tab



When the **JTAG Chain Debugging** tab is active, you can use the **Device chain** pane to activate either a single device or all available devices:

- To activate a single device, click a device in the **Device chain** pane to select it and then right-click the device and select **Activate selected device**.
- To activate all available devices, right-click the **Device chain** pane background and select **Activate all devices**.

If the **Device chain** pane is empty, populate the **Device chain** pane by right-clicking in the pane and selecting **Test JTAG chain**. You can also click **Test JTAG Chain** on the **JTAG Chain Integrity** tab.

Table 13. JTAG Chain Debugging Tab

Task	Description
Playback the debugging sequence	Apply the commands from a saved JTAG debugging session log to repeat prior debugging steps: • **Open JTAG Chain Log**–browse to, open, and run a saved JTAG Chain Debugger session log. • **Stop**–stop the running of the commands from the saved session log.
Manually shift the JTAG sequence	Apply single-step commands:

continued...

Task	Description
	• **Command**–run one of the following commands for debugging the JTAG chain: — **Run test**–place the TAP controller in the specified state for a specified number of clocks. — **Scan Instruction Register**–specify a scan pattern that you can apply to the target instruction registers. — **Scan Data Register**–specify a scan pattern that you can apply to the target data registers. — **Goto State**–move the TAP controller from one stable state to another for further testing. You can also double-click a state in the **TAP state** pane to run this command. • **TAP state**–set the target state for the commands. The selected target state are reflected. • **Clocks**–specify the number of clocks to use when running a command. • **TDI**–specify the command data that is sent via TDI pins • **TDO**–shows command scan results read from TDO pins • **Run**–apply the selected **Command** and **Option** setting to the JTAG chain.

2.4.2. Editing the Details of an Unknown Device

When the Quartus Prime Programmer automatically detects devices with shared JTAG IDs, the Programmer prompts you to specify the device in the JTAG chain. If the Programmer does not prompt you to specify the device, you must manually add each device in the JTAG chain to the Programmer, and define the instruction register length of each device.

To edit the details of an unknown device, follow these steps:

1. Double-click the unknown device listed under the device column.
2. Click **Edit**.
3. Change the device **Name**.
4. Specify the **Instruction register length**.
5. Click **OK**.
6. Save the `.cdf` file.

2.4.3. Running JTAG Daemon with Linux

The JTAGD daemon is the Linux version of a JTAG server. The JTAGD daemon allows a remote machine to program or debug boards connected to a Linux host over the network. The JTAGD daemon also allows programs to share JTAG resources.

Running the JTAGD daemon prevents:

- The JTAGD server from exiting after two minutes of idleness.
- The JTAGD server from not accepting connections from remote machines, which might lead to an intermittent failure.

To run JTAGD as a daemon:

1. Create an `/etc/jtagd` directory.
2. Set the permissions of this directory and the files in the directory to allow read/write access.
3. Execute `jtagd` (with no arguments) from the `quartus/bin` directory.

The JTAGD daemon is now running and does not terminate when you log off.

2.5. Programming with Flash Loaders

Parallel and serial configuration devices do not support the JTAG programming interface. However, you can use a flash loader to program configuration devices in-system via the JTAG interface. The flash loader allows an FPGA to function as a bridge between the JTAG interface and the configuration device. The Quartus Prime software supports various parallel and serial flash loaders for programming bitstream storage and configuration via flash memory devices.

Refer to the following documents for step-by-step flash programming instructions.

Related Information

- Generic Serial Flash Interface Intel FPGA IP Core User Guide
- Intel Parallel Flash Loader IP Core User Guide
- Generic Flash Programmer User Guide
- Customizable Flash Programmer User Guide

2.5.1. Specifying Flash Partitions

Flash partitions allow you to store bitstreams or raw data.

Note: The **Programming File Generator** supports defining flash partitions only for `.jic` or `.pof` programming files.

To create flash partitions in the **Configuration Devices** tab:

1. Select the device and click **Add Partition**.
2. In the **Add Partition** dialog box, define the following parameters, and then click **OK**:

Table 14. Add Partition Dialog Box Settings

Setting	Description
Name	Name that you give to the partition.
Input file	Input file to program into the flash partition.
Page	Configuration devices can store multiple configuration bitstreams in flash memory, called pages. CFI configuration devices can store up to eight configuration bitstreams. Hyperflex® devices can store up to four configuration bitstreams, including the factory image. In Hyperflex devices, with the remote system update feature enabled, **Page** represents the parity.
Address Mode	The options are: • **Auto**—automatically allocates a block in the flash device to store the data. • **Block**—specify the start and end address of the flash partition. • **Start**—specify the start address of the partition. The tool assigns the end address of the partition based on the input data size.
Start address	Specifies the start address of the partition. Only enabled when **Address Mode** is **Block** or **Start**.
End address	Specifies the end address of the partition. Only enabled when **Address Mode** is **Block**.

The partition associated to the device appears in the device list.

3. If you want to change the parameters of a partition, click the partition and then click **Edit**.
4. If you want to remove a partition, click the partition and then click **Remove**.
5. After specifying the settings for all flash partitions, click **Generate.**

2.5.2. Full Erase of Flash Memory Sectors

When performing flash memory erase operations via JTAG and a `.jic` file, the Quartus Prime Programmer erases only the flash memory sectors that the `.jic` specifies.

For example, if you specify a `.jic` file containing only a 13.6Mbits FPGA image on an EPCQ64A device, the Programmer erases only the bottom 13.6Mbits, and does not erase the remaining 50.4Mbits of data.

To erase the entire flash memory device contents, do not specify a `.jic` file for flash programming. Rather, manually add the flash device to the associated FPGA device chain by following these steps:

1. In the Programmer, right-click the target FPGA device, and then click **Edit ➤ Attach Flash Device**.
2. Select the appropriate flash device from the list. The Factory Default Serial Flash Loader loads for the FPGA automatically.
3. In the Programmer, enable the **Erase** checkbox, and click **Start** to start the erase operation.

2.6. Verifying the Programming File Source with Project Hash

Quartus Prime programming files support the project hash property, which identifies the source project from which programming files generate.

During compilation, the Quartus Prime software generates a unique project hash, and embeds this hash value in the programming files (`.sof`). You can verify the source of programming files by matching the project and programming file hash values.

The project hash does not change for different builds of the Quartus Prime software, or when you install a software update. However, if you upgrade any IP with a different build or patch, the project hash changes.

2.6.1. Obtaining Project Hash for Arria 10 Devices

To obtain the project hash value of a `.sof` programming file for a design, use the `quartus_asm` command-line executable (`quartus_asm.exe` in Windows) with the `--project_hash` option.

```
quartus_asm --project_hash <sof-file>
```

Example 1. Output of Project Hash Command:

In this example, the programming file is `worm.sof`.

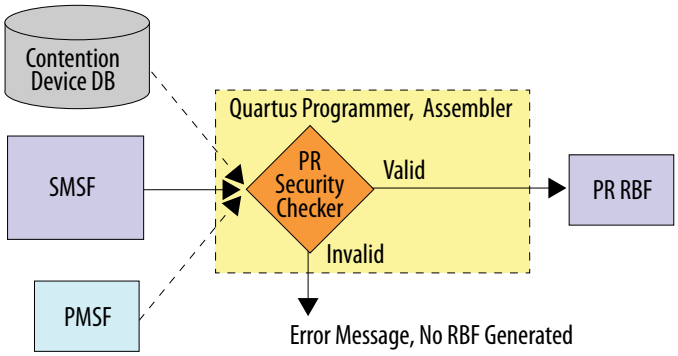
```
Info: *******************************************************************
Info: Running Quartus Prime Assembler
Info: Version 17.0.0 Build 288 04/12/2017 SJ Pro Edition
Info: Copyright (C) 2017  Intel Corporation. All rights reserved.
Info: Your use of Intel Corporation's design tools, logic functions
Info: and other software and tools, and its AMPP partner logic
Info: functions, and any output files from any of the foregoing
Info: (including device programming or simulation files), and any
Info: associated documentation or information are expressly subject
Info: to the terms and conditions of the Intel Program License
Info: Subscription Agreement, the Intel Quartus Prime License Agreement,
Info: the Intel MegaCore Function License Agreement, or other
Info: applicable license agreement, including, without limitation,
Info: that your use is for the sole purpose of programming logic
Info: devices manufactured by Intel and sold by Intel or its
Info: authorized distributors.  Please refer to the applicable
Info: agreement for further details.
Info: Processing started: Fri Apr 14 18:01:47 2017
Info: Command: quartus_asm -t project_hash.tcl worm.sof
Info: Quartus(args): worm.sof
0x1ffdc3f47c57bbe0075f6d4cb2cb9deb
Info (23030): Evaluation of Tcl script project_hash.tcl was successful
Info: Quartus Prime Assembler was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 1451 megabytes
Info: Processing ended: Fri Apr 14 18:01:56 2017
Info: Elapsed time: 00:00:09
Info: Total CPU time (on all processors): 00:00:04
```

2.7. Using PR Bitstream Security Verification (Stratix 10 Designs)

PR bitstream validation confirms that the persona does not use FPGA resources that are unauthorized by the `.smsf`.

Thereafter, the Programmer requires both the `.pmsf` and `.smsf` to generate the PR bitstream (`.rbf`) for this PR region, ensuring that the PR persona can only change bits that the persona owns. The Platform Owner can optionally release `.smsf` files to third-party Clients as part of the PR region collateral. The Platform Owner uses the `.smsf` to generate the PR bitstream from Client's `.pmsf` for this PR region.

Figure 37. PR Bitstream Security Validation in Programmer



The Platform Owner should follow these steps to license, enable, and use PR bitstream security verification:

1. Obtain the license file to enable generation of `.smsf` files for PR regions during base compilation, and to perform PR bitstream security verification during PR bitstream generation in the Programmer. To obtain the license, login or register for a My-Intel account, and then submit an Intel Premier Support case requesting the license key.
2. To add the license file to the Quartus Prime Pro Edition software, click **Tools ➤ License Setup** and specify the feature **License File**.
3. To enable PR security validation features, add the following line to the project `.qsf`:

```
set_global_assignment -name PR_SECURITY_VALIDATION on
```

4. Compile the base revision.
5. Following base compilation, view the Assembler reports to view the generated `.smsf` files required for bitstream generation for each PR region.
6. The Client provides the `.pmsf` to the Platform Owner.
7. The Platform Owner verifies the `.pmsf`, converts the `.pmsf` to `.rbf`, and configures the FPGA device with the `.rbf`.
8. The platform owner converts the `.pmsf` to a PR bitstream. Providing the `.smsf` file to `quartus_cmf` instructs the tool to validate the `.pmsf` against that `.smsf`, and then to generate a bitstream only if the files are compatible.

```
quartus_cpf -c –-smsf=<smsf_file> <pmsf_file> <output_file>
```

Related Information

Quartus Prime Pro Edition User Guide: Partial Reconfiguration
In *Quartus Prime Pro Edition User Guide: Partial Reconfiguration*

2.8. Stand-Alone Programmer

The free Stand-Alone Programmer is available and has the same full functionality as the Quartus Prime Programmer.

The Stand-Alone Programmer is useful when programming devices on a workstation that does not have an Quartus Prime software license. The Stand-Alone Programmer does not require a separate Quartus Prime software license. Download the Stand-Alone Programmer from the Download Center on the Intel website.

Related Information

Download Center for FPGAs

2.8.1. Stand-Alone Programmer Memory Consumption

The following operations increase memory usage in the Stand-Alone Programmer:

- Auto-detect
- Adding programming files to the flash memory
- Manually attaching the flash in the Programmer

In Windows systems, the Stand-Alone Programmer has the following memory limitations:

Table 15. Stand-Alone Programmer Memory Limitations

Application	Maximum Flash Device Size	Flash Device Operation Using PFL
64-bit Stand-Alone Programmer	Up to 2 Gb	Multiple Flash Device

2.9. Programmer Settings Reference

The following topics describe Quartus Prime settings that impact programming and programming file generation.

2.9.1. Device & Pin Options Dialog Box

The following tables describe **Device & Pin Option** settings that impact generation of primary and secondary programming files. To access these settings, click **Assignments ➤ Device ➤ Device & Pin Options**.

Table 16. General Device Options

Allow you to specify basic device configuration options that are independent of a specific configuration scheme. To access these settings, click **Assignments ➤ Device ➤ Device and Pin Options ➤ General**.

Option	Description
Options *Note:* Not supported for Agilex 7 or Stratix 10 devices.	• **Auto-restart configuration after error**—restarts the configuration process automatically if a data error is encountered. If this option is turned off, you must externally direct the device to restart the configuration process if an error occurs. This option is available for passive serial and active serial configuration schemes. • **Release clears before tri-states**—releases the clear signal on registered logic cells and I/O cells before releasing the output enable override on tri-state buffers. If this option is turned off, the output enable signals are released before the clear overrides are released. • **Enable user-supplied start-up clock (CLKUSR)**—uses a user-supplied clock on the CLKUSR pin for initialization. When turned off, external circuitry is required to provide the initialization clock on the DCLK pin in the Passive Serial and Passive Parallel Synchronous configuration schemes; in the Passive Parallel Asynchronous configuration scheme, the device uses an internal initialization clock. • **Enable device-wide reset (DEV_CLRn)**—enables the DEV_CLRn pin, which allows all registers of the device to be reset by an external source. If this option is turned off, the DEV_CLRn pin is disabled when the device operates in user mode and is available as a user I/O pin. • **Enable device-wide output enable (DEV_OE)**—enables the DEV_OE pin when the device is in user mode. If this option is turned on, all outputs on the chip operate normally. When the pin is disabled, all outputs are tri-stated. If this option is turned off, the DEV_OE pin is disabled when the device operates in user mode and is available as a user I/O pin. • **Enable INIT_DONE output**—enables the INIT_DONE pin, which allows you to externally monitor when initialization is complete and the device is in user mode. If this option is turned off, the INIT_DONE pin is disabled when the device operates in user mode and is available as a user I/O pin.

continued...

Option	Description
	• **Enable JTAG Pin Sharing**—enables the JTAG pin sharing feature. The JTAGEN pin is enables and becomes a dedicated input pin in user mode. JTAG pins (TDO, TCK, TDI, and TMS pins) are available as test pins when the JTAGEN pin is pull low. JTAG pins are dedicated when the JTAGEN pin is high. If this option is turned off, the JTAGEN pin is disabled when the device operates in user mode and is available as a user I/O pin. JTAG pins are retained as dedicated JTAG pins. • **Enable nCONFIG, nStatus, and CONF_DONE pins**—enables the major configuration pins, nCONFIG, nSTATUS, and CONF_DONE pin in user mode. If this option is turned off, the nCONFIG, nSTATUS, and CONF_DONE pins are disabled when the device operates in user mode and are available as user I/O pins. • **Enable OCT_DONE** —enables the OCT_DONE pin, which controls whether the INIT_DONE pin is gated by OCT_DONE pin. If this option is turned off, the INIT_DONE pin is not gated by the OCT_DONE pin. • **Enable security bit support**—enables the security bit support, which prevents data in a device from being obtained and used to program another device. This option is available for supported device (MAX II, and MAX V) families. • **Set unused TDS pins to GND**—sets the unused temperature sensing diode TSD pins, TEMPDIODEp and TEMPDIODEn to GND in the pin. By default, TSD pins are available for connection to an external temperature sensing device; however, you must manually connect the pins to GND if they are not connected. When turned on, this option updates the information in the .pin file and does not affect FPGA behavior. • **Enable CONFIG_SEL pin**—enables the BOOT_SEL pin in user mode. If this option is turned off, the BOOT_SEL pin is disabled when the device operates in user mode and is available as a user I/O pin. • **Enable nCEO pin**—enables the nCEO pin. This pin should be connected to the nCE of the succeeding device when multiple devices are being programmed. If this option is turned off, the nCEO pin is disabled when the device operates in user mode and is available as a user I/O pin. • **Enable autonomous PCIe HIP mode**—releases the PCIe HIP after periphery configuration, before device core configuration completes. This option only takes effect if CvP mode is disabled. • **Enable the HPS early release of HPS IO**—releases the HPS shared I/O bank after the IOCSR programming.
Auto usercode	Sets the JTAG user code to match the checksum value of the device programming file. The programming file is a .pof for non-volatile devices, or an .sof for SRAM-based devices. If you turn on this option, the **JTAG user code** option is not available.
JTAG user code	Specifies a hexadecimal number for the device selected for the current Compiler settings. The JTAG user code is an extension of the option register. This data can be read with the JTAG USERCODE instruction. If you turn on **Auto usercode**, this option is not available.
In-system programming clamp state	Allows you to specify the state that the pins take during in-system programming for used pins that do not have an in-system programming clamp state assignment. Unused pins and dedicated inputs must always be tri-stated for in-system programming. Used pins are tri-stated by default during in-system programming, which electrically isolates the device from other devices on the board. At times, however, in order to prevent system damage you may want to specify the logic level for used pins during in-system programming. The following settings are available:

continued...

Option	Description
	• **Tri-state**—the pins are tri-stated. • **High**—the pins drive VCCIO. • **Low**—the pins drive GND. • **Sample and Sustain**—the pins drive the level captured during the SAMPLE/PRELOAD JTAG instruction.
Configuration clock source	Specifies the clock source for device initialization (the duration between CONF_DONE signal went high and before INIT_DONE signal goes high). For AS x1 or AS x4 configuration mode, you can select either **Internal Oscillator** or **CLKUSR** pin only. The DCLK pin is an illegal option for AS mode. In 14 nm device families, only **Internal Oscillator** or **OSC_CLK_1** pins are available.
Device initialization clock source	Specifies the clock source for device initialization (the duration between CONF_DONE signal went high and before INIT_DONE signal goes high). For AS x1 or AS x4 configuration mode, you can select either **Internal Oscillator** or **CLKUSR** pin only. The DCLK pin is an illegal option for AS mode. In 14 nm device families, only **Internal Oscillator** or **OSC_CLK_1** pins are available.

Table 17. Configuration Options

Allow you to specify the configuration scheme, configuration device and pin options, serial clock source, and other options for subsequent device configuration with your programming bitstream. To access these settings, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Configuration**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Configuration scheme	Specifies the scheme of configuration for generation of appropriate primary and secondary programming files, such as **Active Serial x4**. Only options appropriate for the current **Configuration Scheme** are available.
Configuration Device	Allows you to specify options for an external configuration device that stores and loads configuration data. • **Configuration device I/O voltage**—specifies the VCCIO voltage of the configuration pins for the current configuration scheme of the target device. This option is available for supported device families. • **Force VCCIO voltage to be compatible with configuration I/O voltage**—forces the VCCIO voltage of the configuration pins to be the same as the configuration device I/O voltage. If you turn off this option, the VCCIO voltage of the configuration pins may vary depending on the I/O standards used in the I/O banks containing the configuration pins. This option is available for supported device families.
Configuration Pin Options	Enables or disables operation of specific device configuration pins for status monitoring, SEU error detection, CvP, and other configuration pin options.
Generate compressed bitstreams	Generates compressed bitstreams and enables bitstream decompression in the target device.
Active serial clock source	Specifies the configuration clock source for Active Serial programming. Options range from 12.5 MHz to 100 MHz.
VID Operation Mode	Enables Voltage Identification logic in the target device with selected operation mode. The available options are **PMBus Master** or **PMBus Slave**.
HPS/FPGA configuration order	For hard processor system (HPS) configuration, specifies the order of configuration between the HPS and FPGA. The options are **HPS First**, **After INIT_DONE**, and **When requested by FPGA**.
HPS debug access port	• **Disabled**—the HPS JTAG is not enabled. • **HPS Pins**—the HPS JTAG is routed to the HPS dedicated I/O. • **SDM Pins**—the HPS JTAG is chained to the FPGA JTAG.
Disable Register Power-Up Initialization	Specifies whether the Assembler generates a bit stream with register power-up initialization.

Table 18. Unused Pin Options

Allow you specify the reserve state of all the unused pins on the device. To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Unused Pins**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Reserve all unused pins	• **As input tri-stated**—the pins reserve as tri-state input pins. • **As output driving ground**—the pins reserve as output pins and drive the ground signal. • **As output driving an unspecified signal**—the pins reserve as output pins and drive any signal. • **As input tri-stated with bus-hold circuitry**—the pins reserve as tri-state input pins with bus-hold circuitry. • **As input tri-stated with weak pull-up**—the pins reserve as tri-state input pins with weak pull-up resistors.

Table 19. Dual-Purpose Pin Options

Allow you to specify whether the associated dual-purpose pin is reserved, and the reservation purpose. To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Dual-Purpose Pins**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Dual-purpose pins	• **Use as regular I/O**—the dual-purpose pin is not reserved. Rather the I/O pin is in in user mode. • **Use as programming pin**—the nCE0 pin is reserved as a dedicated programming pin. • **As input tri-stated**—the dual-purpose pin is reserved as an input pin. • **As output driving ground**—the dual-purpose pin is reserved as an output pin and drives the ground signal. • **As output driving an unspecified signal**—the dual-purpose pin is reserved as an output pin and drives any signal. • **Compiler configured**—the Compiler automatically selects the best reserve setting for the dual-purpose pin, considering the current configuration scheme, and whether the pins are only used for configuration. If your design uses the Active Parallel configuration scheme and the Programmer does not communicate directly with the parallel flash device in user mode, you should reserve all dual-purpose pins connected to the parallel flash device as Compiler configured.

Table 20. Board Trace Model Options

For Cyclone 10 GX designs only, allows you to specify the board trace, termination, and capacitive load parameters for each I/O standard. The board trace model parameters then apply to all output or bidirectional pins that you assign with the specified I/O standard. Board trace model parameters do not apply if you assign them to anything other than an output or bidirectional pin. You can create board trace model assignments for individual output or bidirectional pins in the Pin Planner. To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Board Trace Model**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
I/O standard	Specifies the supported I/O standard, such as **Differential 1.8-V SSTL Class II**.
Board trace model	Lists the board trace model parameters, with units, and values for the specified **I/O standard**. You can change the value of each parameter. The board trace model assignments apply to all output and bidirectional pins with the specified I/O standard assigned to them.

Table 21. I/O Timing Options

Allow you to specify the node at which output I/O timing terminates. To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ I/O Timing**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Default timing I/O endpoint	Specify **Near end** or **Far end**.

Table 22. Voltage Options

Allow you to specify the default I/O bank voltage for pins on the target device. Also displays the core voltage of the device or other internal voltage information. To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Voltage**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Default I/O standard	Specify **1.2 V**, **1.5 V**, **1.8 V**, **2.5 V**, **3.0 LVTTL**, or **3.0 LVCMOS**.

Table 23. Error Detection CRC Options

Allow you to specify whether to use error detection cyclic redundancy check (CRC) and the value by which you want to divide the error check frequency for the currently selected device. To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Error Detection CRC**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Enable Error Detection CRC_ERROR pin	Enables error detection CRC and CRC_ERROR pin usage for the targeted device. This check determines the validity of the programming data in the device. Any changes in the data while the device is in operation generates an error. *Note:* Not available for Agilex 7 or Stratix 10 devices.
Enable Open Drain on CRC Error pin	Sets the CRC ERROR pin as an open-drain pin. This action decouples the voltage level of the CRC ERROR pin from VCCIO voltage. When you turn on this option, you must connect a pull-up resistor to the CRC ERROR pin. *Note:* Not available for Agilex 7 or Stratix 10 devices.
Enable error detection check	Enables error detection CRC checking to verify the validity of programming data in the device, and reports any changes in the data while the device is in operation.
Minimum SEU interval	Specifies the minimum time interval between two checks of the same bit. Setting to 0 means check as frequently as possible. Setting to a large value saves power. The unit of interval is millisecond. The maximum allowed number of intervals is 10000.
Enable internal scrubbing	Specifies use of internal scrubbing to correct any detected single error or double adjacent error within the core configuration memory while the device is still running.
Generate SEU sensitivity map file (.smh)	Generates a Single Event Upset Sensitivity Map file. This file allows you to enable the Advanced SEU detection feature.
Allow SEU fault injection	Allows the injection of fault patterns to test for SEU.
Enable external scrubbing	Enables the Assembler to provide hashes of configuration RAM (CRAM) content for all sectors within the SRAM Object File (SOF) option. *Note:* Available only for Agilex 5 device.
seu_error_out signal behavior	Controls the behavior of the `seu_error_out signal`. Either correctable or uncorrectable and only uncorrectable are the supported options. *Note:* Available only for Agilex 5 device.

Table 24. CvP Settings

Specifies the configuration mode for Configuration via Protocol (CvP). To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ CvP Settings**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Configuration via protocol	In **Initialization and update** mode, the periphery image stores in an external configuration device and loads the image into the FPGA through a conventional configuration scheme. The core image stores in a host memory and loads into the FPGA through the PCIe link. In **Core initialization** mode, the periphery image stores in an external configuration device and loads into the FPGA through the conventional configuration scheme. The core image is stores in a host memory and is loads into the FPGA through the PCIe link. In **Core update** mode, the

continued...

Option	Description
	FPGA device is initialized after initial system power up by loading the full configuration image from the external local configuration device to the FPGA. You can use the PCIe link to perform one or more FPGA core image update through this mode. In the **Off** mode, CvP is turned off.
Enable CvP_CONFDONE pin	Indicates that the device finished core programming in Configuration via Protocol mode. If this option is turned off, the CvP_CONFDONE pin is disabled when the device operates in user mode and is available as a user I/O pin. *Note:* Not available for Agilex 7 or Stratix 10 devices.
Enable open drain on CvP_CONFDONE pin	Enables the open drain on the CvP_CONFDONE pin. *Note:* Not available for Agilex 7 or Stratix 10 devices.

Table 25. Partial Reconfiguration Options

Specifies generation of secondary programming files that partial reconfiguration requires. To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Partial Reconfiguration**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Enable partial reconfiguration pins	Allows you to enable the PR_REQUEST, PR_READY, PR_ERROR, PR_DONE, DCLK, and DATA[31..0] pins. These pins are needed to support partial reconfiguration (PR) with an external host. An external host uses the PR_REQUEST pin to request partial reconfiguration, the PR_READY pin to determine if the device is ready to receive programming data, the PR_ERROR pin to externally monitor programming errors, and the PR_DONE pin to indicate the device finished programming. If this option is turned off, these pins are not available as PR pins when the device operates in user mode and the dual-purpose programming pins are available as user I/O pins. *Note:* Not available for Agilex 7 or Stratix 10 devices.
Enable open drain on partial reconfiguration pins	Allows you to specify an open drain on the PR_READY, PR_ERROR, PR_DONE Partial Reconfiguration pins. *Note:* Not available for Agilex 7 or Stratix 10 devices.
Generate Partial-Masked SOF files	Generates a Partial-Masked SRAM Object file (.pmsf) containing both configuration data and region definitions that can be used to re-configure a device region. If this option is turned on, the .pmsf generates instead of a Mask Settings file (.msf).
Generate Partial Reconfiguration RBF	Generates a Partial Reconfiguration Raw Binary File (.rbf) containing configuration data that an intelligent external controller can use to reconfigure the portion of target device.

Table 26. Power Management & VID Options

For Stratix 10 and Agilex 7 devices only, specifies options for managing power, such as the bus speed mode and the address of the voltage regulator when in PMBus Master mode. To access, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Power Management & VID Options**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Bus speed mode	Specifies the bus speed mode (for example, **100 KHz** or **400 KHz**) in PMBus Master mode.
Slave device type	Specifies the slave device type when the target FPGA device is in PMBus master mode. Available options are **ED8401**, **EM21XX**, **EM22XX**, **ISL82XX**, **ISL69260**, **LTM4677**, and **Other**.
Device address in PMBus Slave mode	Specifies the starting 00 device address when in PMBus Slave mode.
PMBus device 0 slave address through **PMBus device 7 slave address**	Specifies 7-bit hexadecimal value (without leading prefix 0x). For example, 7F for the slave address of a voltage regulator when in PMBus Master mode. You must specify a non-zero address.
Voltage output format	Specifies the **Auto discovery**, **Direct format**, or **Linear format** output voltage format when in PMBus Master mode

continued...

Option	Description
Direct format coefficient (m,b,R)	Specifies direct format coefficient m, b, or R when in PMBus Master mode. Signed integer between -32768 and 32767. Coefficient m is the slope coefficient. Coefficient b is the offset. Coefficient R is the exponent. Refer to the PMBus device manufacturer product documentation for these values. You must set this parameter when output voltage format of PMBus device is **Direct format** or **Auto discovery** format. You must specify a non-zero address when the output voltage format of PMBus device is in **Direct format**.
Linear format N	Specifies linear format N when in PMBus Master mode. Signed integer between -16 and 15. This is the exponent for the mantissa for the output voltage related command when VOUT format is set to **Linear format**. Refer to the PMBus device manufacturer product documentation for these values. You must specify a non-zero value for **Linear format**.
Translated voltage value unit	Specifies the **Volts** or **Millivolts** output voltage format when in PMBus Master mode.
Enable PAGE command	The FPGA PMBus master uses PAGE command to set all output channels on registered regulator modules to respond to VOUT_COMMAND.
More Options	Opens the **Advanced Power Management & VID Options** dialog box for specifying the following additional options: • **Enable status_byte for polling**—Enables the device to query the voltage regulator status via STATUS WORD command. If any errors are found, the SEU_ERROR pin is asserted. The Error Message Queue (EMQ) includes the complete details about the error. For instructions about retrieving the EMQ, refer to the *SEU Mitigation User Guide* for your device. • **Disable VID for debug purpose only**—Use this option for debug purpose only. When you enable this option, you must set the device operating voltage to 0.8 V. VID is disabled, and device performance and functionality are not guaranteed. • **Diagnostic Boot**—Performs additional checks of the voltage regulator's configuration and operation. *Note:* This feature adds to the configuration time, so enable this option only during board bring-up operations.

Table 27. Assembler Security Settings

For Stratix 10 devices, specifies settings for programming bitstream authentication, encryption, scrambling, and other eFuse enabled security options. To access these settings, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Security**. Disabled options are unavailable for the current device or configuration mode.

Option	Description
Quartus Key File	Specifies the first level signature chain file (.qky) that you generate. This chain includes the root key (.pem) and one or more design signing keys (.pem) required to sign the bitstream and allow access to the FPGA when using authentication or encryption.
Encryption key storage select	Specifies the location that stores the .qek key file. You can select either **Battery Backup RAM** or **eFuses** for storage.
Encryption update ratio	Specifies the ratio of configuration bits compared to the number of key updates required for bitstream decryption. You can select either **31:1** (the key must change 1 time every 31 bits) or **Disabled** (no update required). Encryption supports up to 20 intermediate keys.
Enable scrambling	Scrambles the configuration bitstream.
More Options	Opens the **More Security Options** dialog box for specifying additional physical security options.

Table 28. Configuration PIN Dialog Box

For Stratix 10 devices, allows you to enable or disable specific configuration pins. For example, you can enable the CvP_CONFDONE pin, which indicates that the device finished core programming in Configuration via Protocol mode. To access these settings, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Configuration Pin Options**. Disabled options are unavailable for the current device or configuration mode.

Option	Values	Description
USE PWRMGT_SCL output	**SDM_1O0|SDM_IO14**	This is a required PMBus interface for the power management when the VID operation mode is the PMBus Master or PMBus Slave mode. Disable this pin for a non-SmartVID device. Intel recommends using the SDM_IO14 pin for this function.
Use PWRMGT_SDA output	**SDM_1O11|SDM_1O12| SDM_1O16**	This is a required PMBus interface for the power management when the VID operation mode is the PMBus Master or PMBus Slave mode. Disable this pin for a non-SmartVID device. Intel recommends using the SDM_IO11 pin for this function.
Use PWRMGT_ALERT output	**SDM_1O0|SDM_1O12**	This is a required PMBus interface for the power management that is used only in the PMBus Slave mode. Disable this pin for a non-SmartVID device. Intel recommends using the SDM_IO12 pin for this function.
USE CONF_DONE output	**SDM_100, SDM_1010 - SDM_1016**	Implement CONF_DONE using appropriate configuration pin resource.
USE INIT_DONE output	**SDM_100, SDM_1010 - SDM_1016**	Enables the INIT_DONE pin, which allows you to externally monitor when initialization is completed and the device is in user mode. If this option is turned off, the INIT_DONE pin is disabled when the device operates in user mode and is available as a user I/O pin.
USE CVPCONF_DONE output	**SDM_100, SDM_1010 - SDM_1016**	Enables the CVP_CONFDONE pin, which indicates that the device finished core programming in Configuration via Protocol mode. If this option is turned off, the CVP_CONFDONE pin is disabled when the device operates in user mode and is available as a user I/O pin.
USE SEU_ERROR output	**SDM_100, SDM_1010 - SDM_1016**	Enables the SEU_ERROR pin for use in single event upset error detection.
USE UIB CATTRIP output	**SDM_100, SDM_1010 - SDM_1016**	Enables UIB_CATTRIP output to indicate an extreme over-temperature conditioning resulted from UIB usage.
USE HPS cold nreset	**SDM_100, SDM_1010 - SDM_1016**	An optional reset input that cold resets only the HPS and is configured for bidirectional operation.
Direct to factory image	**SDM_100, SDM_1010 - SDM_1016**	If this pin asserted then device loads the factory image as the first image after boot without attempting to load any application image.
USE DATA LOCK output	**SDM_100, SDM_1010 - SDM_1016**	Output to indicate DIBs on both die in the same package is ready for data transfer.

Related Information

Enabling Bitstream Authentication (Programming File Generator) on page 20

2.9.2. More Security Options Dialog Box

Table 29. More Security Options Dialog Box

For Stratix 10 devices, specifies additional configuration bitstream physical security settings. To access these settings, click **Assignments ➤ Device ➤ Device and Pin Options ➤ Security ➤ More Settings** button. Disabled options are unavailable for the current device or configuration mode.

Option	Description	Values
Disable JTAG	Disables JTAG command and configuration of the device. Setting this eliminates JTAG as mode of attack, but also eliminates boundary scan functionality.	• **Off**—inactive • **On**—active until wipe of containing design • **On sticky**—active until next POR • **On check**—checks for corresponding blown fuse
Force SDM clock to internal oscillator	Disables an external clock source for the SDM. The SDM must use the internal oscillator. Using an internal oscillator is more secure than allowing an external clock source for configuration.	
Force encryption key update	Specifies that the encryption key must update by the frequency that you specify for the **Encryption update ratio** option. The default ration value is 31:1. Encryption supports up to 20 intermediate keys.	
Disable virtual eFuses	Disables the eFuse virtual programming capability.	
Lock security eFuses	Causes eFuse failure if the eFuse CRC does not match the calculated value.	
Disable HPS debug	Disables debugging through the JTAG interface to access the HPS.	
Disable encryption key in eFuses	Specifies that the device cannot use an AES key stored in eFuses. Rather, you can provides an extra level of security by storing the AES key in BBRAM.	
Disable encryption key in BBRAM	Specifies that the device cannot use AES key stored in BBRAM. Rather, you can provides an extra level of security when you store the AES key in eFuses.	

2.9.3. Output Files Tab Settings (Programming File Generator)

The **Output Files** tab allows you to specify the type of secondary programming file that you want to generate (output) with the **Programming File Generator**. The **Programming File Generator** converts a primary programming file (for example, `.sof`) into a programming file for alternative programming methods (for example, a `.jic` for flash programming). The **Output Files** tab and options change dynamically according to your selections.

The following output file options are available:

Table 30. Output File Options

Setting	Description
Device family	Specifies the FPGA device family you are targeting for configuration. **Programming File Generator** supports only Agilex 7, Stratix 10, MAX 10, and Cyclone 10 LP devices.
Configuration mode	Specifies the method of FPGA configuration, such as **Active Serial x4**, **AVST x8**, **AVST x16**, or **AVST x32**. Generic Flash Programmer supports only **Active Serial x4**.
Output directory and **Name**	Specifies the name and location of the file you generate. By default, this location is in the top-level project directory.
File Types	Allows you to enable the type of secondary programming file that you want to generate. Generic Flash Programmer supports only **JTAG Indirect Configuration File (.jic)**. The available options include: • **JTAG Indirect Configuration File (.jic)** • **Programmer Object File (.pof)** • **Raw Binary File for CvP Core Configuration (.rbf)** • **Raw Binary File for HPS Core Configuration (.rbf)** • **Raw Binary File for Partial Reconfiguration (.rbf)** • **Raw Programming Data File (.rpd)**

2.9.4. Input Files Tab Settings (Programming File Generator)

The **Input Files** tab allows you to specify the `.sof`, `.pmsf`, or `.rbf` file that contains the configuration bitstream data required to generate one or more secondary programming files. The **Input Files** tab and options change dynamically, according to your **Output Files** tab selections.

The following input file settings are available:

Table 31. Input File Settings

Setting	Description
Add Bitstream	Click this button to specify a `.sof`, `.pmsf`, or `.rbf` as input for generation of the secondary programming file you select in **Output Files**. Depending on the target device, the Quartus Prime software may allow you to add multiple SOF files.
Add Raw Data	Click this button to specify a `.hex` or `.bin` file that contains raw programming data as input for generation of the secondary programming file you select in **Output Files**.
Remove	Removes the file you select from the **Input Files** tab.
Properties	Displays the properties of the item you select in the **Input Files** tab.

Related Information

Enabling Bitstream Encryption (Programming File Generator) on page 23

2.9.5. Bitstream Co-Signing Security Settings (Programming File Generator)

Table 32. Input File Properties Dialog Box (Programming File Generator)

Allows you to specify options for bitstream authentication, co-signing, and encryption security. To access, select an `.sof` or `.rbf` in the **Input files** tab in the **Programming File Generator**, and click **Properties**.

Option	Description
Bootloader	Specifies an ASCII text file in Intel hexadecimal format that contains configuration data for programming a parallel data source, such as a configuration device or a mass storage device. The parallel data source in turn configures an SRAM-based Intel device
Enable signing tool	Enables the signing tool that checks for a required Privacy Enhanced Mail Certificates file (`.pem`) for the **Private key file**, and a Quartus Co-Signed Firmware file (`.zip`) for the **Co-signed firmware** option.
Private key file	Specifies the private `.pem` file required to sign the configuration bitstream when using the signing tool. If your `.pem` is password-protected, you are prompted to enter the password.
Co-signed firmware	Specifies the firmware source (`.zip`) required to include the signed firmware in the configuration bitstream.
Finalize encryption	Finalizes the configuration bitstream encryption.
Encryption key file	Specifies the Encryption Key File (`.qek`) required to decrypt the configuration bitstream file.

2.9.6. Configuration Device Tab Settings

The **Configuration Device** tab allows you to specify the properties of an existing or new flash memory device that you want to program. Click **Add Device** to select a programming template for a predefined flash memory, or to click **<<new device>>** and then define a new flash memory device.

The following settings are available:

Table 33. Configuration Device Tab Settings

Option	Description
Device name	Specify a unique name for the flash not already listed in the **Name** column. The **Name** must not contain any empty string (space) or special characters (except "_").
Device ID	Specify the 3-byte ID that the Programmer Auto-Detect operation uses to detect the flash programming device, such as `0x20 0xBB 0x21`.
Device I/O voltage	Specify **1.8V** or **3.0/3.3V** to match your memory device specification.
Device density	Select the total density that corresponds with your flash memory device size.
Total device die	Specify the total number of die for a stacked device (where applicable).
Single I/O mode dummy clock	Specify the Fast Read dummy clock cycle for flash device in single I/O protocol. The programming file generation uses this setting to determine if the configuration requires bit shifting to compensate for the actual dummy clock cycle during Active Serial configuration.
Quad I/O mode dummy clock	Specify the Fast Read dummy clock cycle for flash device in Quad I/O protocol. The programming file generation uses this setting to determine if the configuration requires bit shifting to compensate for the actual dummy clock cycle during Active Serial configuration.
Custom database directory	Specifies the location of the `.xml` file that preserves a flash memory device definition.

continued...

Option	Description
	Note: When you specify a non-default folder for the **Custom database directory** location, place the `.sof` and `.jic` files in the same folder as the `.xml` file to avoid missing a defined flash database or corruption of the `.jic` file.

2.9.7. Add Partition Dialog Box (Programming File Generator)

To open in the **Programming File Generator**, click the **Configuration Device** tab, select a device from the list, and click **Add Partition**.

Allows you to specify the attributes of a new partition. The following settings are available:

Table 34. Add Partition Dialog Box Settings

Setting	Description
Name	Name that you give to the partition.
Input file	Input file to program into the flash partition.
Page	Configuration devices can store multiple configuration bitstreams in flash memory, called pages. CFI configuration devices can store up to eight configuration bitstreams. Hyperflex devices can store up to four configuration bitstreams, including the factory image. In Hyperflex devices, with the remote system update feature enabled, **Page** represents the parity.
Address Mode	The options are: • **Auto**—automatically allocates a block in the flash device to store the data. • **Block**—specify the start and end address of the flash partition. • **Start**—specify the start address of the partition. The tool assigns the end address of the partition based on the input data size.
Start address	Specifies the start address of the partition. Only enabled when **Address Mode** is **Block** or **Start**.
End address	Specifies the end address of the partition. Only enabled when **Address Mode** is **Block**.

2.9.8. Add Filesystem Dialog Box (Programming File Generator)

To open in the **Programming File Generator**, click the **Configuration Device** tab, select a device from the list, and click **Add Filesystem**.

Allows you to specify the attributes of a new file system partition. The following settings are available:

Table 35. Add Filesystem Dialog Box Settings

Setting	Description
Name	Name given to the file system. This value is fixed and set to LITTLEFS. You cannot change this value.
Input file	Input ZIP file that contains files to write to the file system partition.
Address Mode	The options are as follows:

continued...

Setting	Description
	• **Highest** Allocate the start address of the file system partition at the highest address of the flash memory device. • **Start** Allocate the start address of the file system partition at the start address specified in the **Start address** field. • **Auto** Allocate the start address for the file system partition automatically.
Start address	Specifies the start address of the file system. Enabled only when **Address Mode** is **Start**.
Size (MB)	Specifies the size of the file system partition in MB. The default file system size is 1 MB. Enabled only when **Address Mode** is **Highest** or **Start**.

2.9.9. Convert Programming File Dialog Box

Allows you to convert or combine one or more secondary programming files that support alternative device configuration schemes, such as flash programming, partial reconfiguration, or remote system update.

Table 36. Convert Programming File Dialog Box Settings

Setting	Description
Programming file type	Allows you to specify a secondary programming file format for conversion of a primary programming file. The Generic Flash Programmer supports only the `.jic` file type.
Configuration device	Allows you to select a predefined or define a new configuration device. Click the (**...**) button to define a new device and programming flow.
Mode	Allows you to select the method of device configuration. The Generic Flash Programmer supports only the **Active Serial** or **Active Serial x4** modes.
Output file	Specifies the location of the files that **Convert Programming File** generates. By default this location is the top-level project directory.
Input files to convert	Specifies one or more primary programming files for conversion or combination into one or more secondary programming files for alternative programming methods.

2.9.10. Compression and Encryption Settings (Convert Programming File)

The compression and encryption settings allow you to specify options for compression and encryption key security for the device configuration SRAM Object File (`.sof`). To access these settings, select the `.sof` in the **Input files to convert** list in the **Convert Programming File** dialog box, and click **Properties**.

Table 37. SOF File Properties: Bitstream Encryption Dialog Box (Convert Programming Files)

Allows you to specify options for compression and encryption key security for the device configuration SRAM Object File (`.sof`). To access, select an `.sof` in the **Input files to convert** list in the **Convert Programming Files** dialog box, and click **Properties**.

Option	Description
Compression	Applies compression to the bitstream to reduce the size of your programming file. The Quartus Prime Assembler can generate a compressed bitstream image that reduces configuration file size by 30% to 55% (depending on the design). The FPGA device receives the compressed configuration bitstream, and then can decompress the data in real-time during configuration. This option is unavailable whenever **Generate encrypted bitstream** is enabled.
Enable decompression during partial reconfiguration	Enables the option bit for bitstream decompression during Partial Reconfiguration.
Generate encrypted bitstream	Generates an encrypted bitstream configuration image. You then generate and specify an encryption key file (`.ekp`) for device configuration. This option is unavailable whenever **Compression** is enabled.
Enable volatile security key	Allows you to encrypt the `.sof` file with volatile (enabled) or non-volatile (disabled) security key.
Generate encryption lock file	Specifies the name of the encryption lock file (`.elk`) that **Convert Programming Files** generates.
Generate key programming file	Specifies the name of the key programming file (`.key`) that **Convert Programming Files** generates.
Use key file	• **Key 1 file**—specifies the name of Key 1 `.key` file. • **Key 2 file**—specifies the name of Key 2 `.key` file.
Key entry	Specifies the keys for bitstream decryption.
Security options	The following options allow you to enable or disable features that impact device security for the configuration bitstream. • **Disable partial reconfiguration**—disables use of partial reconfiguration for the bitstream. • **Disable key-related JTAG instructions**—disables use of key-related JTAG instructions for the bitstream. • **Disable other extended JTAG instructions**—disables use of other JTAG instructions for the bitstream. • **Force the external JTAG pins into BYPASS mode**—forces the external JTAG pins into BYPASS mode. You can specify **Off**, **Turns On Until the Next Full Configuration**, **Turns on until the next Power-On-Reset event**, **Turns on by blowing the corresponding fuses**,
Design Security Feature Disclaimer	Acknowledges required acceptance of Design Security Disclaimer.

2.9.11. SOF Data Properties Dialog Box (Convert Programming File)

Allows you to define flash memory pages that store configuration data. To access from the **Convert Programming File** dialog box, click the **SOF Data** item and click the **Properties** button.

The following settings are available:

Table 38. SOF Data Properties Dialog Box Settings

Setting	Description
Pages	Configuration devices can store multiple configuration bitstreams in flash memory, called pages. CFI configuration devices can store up to eight configuration bitstreams. Some Intel FPGA devices can store multiple configuration bitstreams, including the factory image.
Address mode for selected pages	The options are: • **Auto**—automatically allocates a block in the flash device to store the data. • **Block**—specify the start and end address of the flash partition. • **Start**—specify the start address of the partition. The tool assigns the end address of the partition based on the input data size.
Start address	Specifies the start address of the partition. Only enabled when **Address Mode** is **Block** or **Start**.
End address	Specifies the end address of the partition. Only enabled when **Address Mode** is **Block**.

2.9.12. Select Devices (Flash Loader) Dialog Box

Allows you to select the device that controls loading of configuration data into a flash memory device. To access from the **Programming File Generator**, click the **Select** button for **Flash loader** in the **Configuration Device** tab. To access from the **Convert Programming File** dialog box, select the **Flash Loader** item and click **Add Device**.

The following settings are available:

Table 39. Flash Loader (Select Devices Dialog Box)

Option	Description
Device family	Specifies the family of the flash loader device.
Device name	Specifies the name of the flash loader device.

2.10. Scripting Support

In addition to the Quartus Prime Programmer GUI, you can access programmer functionality from the command line and from scripts with the Quartus Prime command-line executable `quartus_pgm.exe` (or `quartus_pgm` in Linux).

The following command programs a device from a Microsoft* Windows* command line:

```
quartus_pgm –c usbblasterII –m jtag –o bpv;design.pof
```

Where:

-c usbblasterII specifies the Intel FPGA Download Cable II

-m jtag specifies the JTAG programming mode

-o bpv represents the blank-check, program, and verify operations

design.pof	represents the `.pof` containing the design logic

The Programmer automatically executes the erase operation before programming the device.

In a Linux* terminal terminal window, use:

```
quartus_pgm –c usbblasterII –m jtag –o bpv\;design.pof
```

The following examples shows how to erase flash memory that is connected to the FPGA through an active serial interface from a Microsoft Windows command line:

```
quartus_pgm -c usbblasterII -m jtag -o ri;design.jic@1
```

Where:

-c usbblasterII	specifies the Intel FPGA Download Cable II
-m jtag	specifies the JTAG programming mode
-o ri	represents the serial flash loader program and erase operations
design.jic	represents the JTAG indirect configuration file (`.jic`)
@1	specifies the device number in the JTAG chain on which these operations are performed

In a Linux terminal terminal window, use:

```
quartus_pgm -c usbblasterII -m jtag -o ri\;design.jic@1
```

Related Information

Quartus Prime Scripting
In *Quartus Prime Help*

2.10.1. The jtagconfig Debugging Tool

You can use the `jtagconfig` command-line utility to check the devices in a JTAG chain and the user-defined devices. The `jtagconfig` command-line utility is similar to the auto detect operation in the Quartus Prime Programmer.

For more information about the `jtagconfig` utility, use the help available at the command prompt:

```
jtagconfig [–h | --help]
```

Note: The help switch does not reference the `-n` switch. The `jtagconfig -n` command shows each node for each JTAG device.

Related Information

Command Line Scripting
In *Quartus Prime Pro Edition User Guide: Scripting*

2.11. Using the Quartus Prime Programmer Revision History

Table 40. Document Revision History

Date	Quartus Prime Version	Changes
2024.04.01	24.1	• Made the following changes in *Device & Pin Options Dialog Box*: — Added ISL69260 to the list of slave device types and additional options to the *Power Management & VID Options* table. — Added Agilex 5 device-related options to the *Error Detection CRC Options* table.
2023.04.03	23.1	• Updated product family name to "Intel Agilex 7."
2022.09.26	22.3	• Added "Add Filesystem Dialog Box". • Removed footnotes saying that security features are not available for Agilex 7 devices. For details about security features for Agilex 7 devices, refer to the *Agilex 7 Device Security User Guide*.
2021.07.21	21.2	• Updated "JTAG Chain Debugger Tool". • Added "JTAG Chain Integrity". • Added "JTAG Chain Debugging".
2021.06.21	21.2	• Updated "Scripting Support" with an additional example.
2020.05.26	19.4.0	• Corrected descriptions of "Bus Speed Mode" and "Slave Device Type" power options.
2019.09.30	19.3.0	• Updated "Device & pin Options" topic to reflect new Security settings tab. • Updated "Configuration Device Tab" topic to reflect Custom database directory option. • Referenced compilation support for Agilex 7 devices. • Added "More Security Options Dialog Box" topic. • Added new "Bitstream Co-Signing Security Settings" topic. • Updated "SOF File Properties: Bitstream Encryption Dialog Box" topic. • Added new steps to "Full Erase of Flash Memory Sectors" topic.
2019.06.10	19.1.0	• Updated "Programming with Flash Loaders" topic to reflect new Generic Flash Programmer. • Removed references to obsolete 32-bit stand-alone Programmer. • Added "Erasing Flash Memory Sectors" topic describing complete erase of flash memory. • Added new "Programmer Settings Reference" section containing the following new GUI reference topics: — "Device & Pin Options Dialog Box" — "Input Files Tab Settings (Programming File Generator)" — "Output Files Tab Settings (Programming File Generator)" — "Configuration Device Tab Settings (Programming File Generator)" — "Add Partition Dialog Box (Programming File Generator)" — "Bitstream Compression, Authentication, and Encryption Settings (Programming File Generator)" — "Convert Programming Files Dialog Box" — "Bitstream Compression and Encryption Settings (Convert Programming File)" — "SOF Data Properties Dialog Box" — "Select Devices (Flash Loader) Dialog Box"
2019.04.01	19.1.0	• Added new "Using PR Bitstream Security Verification" topic. • Added new "Basic Device Configuration Steps" topic. • Added new "Programming and Configuration Modes" topic.

continued...

Date	Quartus Prime Version	Changes
		• Retitled and reorganized topics to improve flow of information. • Added enhanced diagram of Programmer to "Quartus Prime Programmer" topic. • Updated screenshots.
2018.10.09	18.1.0	• Created topic: *Stand-Alone Programmer Memory Limitations* from content in topic: *Stand-Alone Programmer*. • Removed outdated support information.
2018.08.07	18.0.0	Reverted document title to *Programmer User Guide: Quartus Prime Pro Edition*.
2018.06.27	18.0.0	• Moved information about programming file generator to new chapter: *Generating Programming Files*.
2018.05.07	18.0.0	• First release as part of the stand-alone *Programmer User Guide*
2017.05.08	17.0.0	• Added Project Hash feature.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of Quartus II to Quartus Prime software.
2015.05.04	15.0.0	Added Conversion Setup File (.cof) description and example.
December 2014	14.1.0	Updated the Scripting Support section to include a Linux command to program a device.
June 2014	14.0.0	• Added Running JTAG Daemon. • Removed Cyclone III and Stratix III devices references. • Removed MegaWizard Plug-In Manager references. • Updated Secondary Programming Files section to add notes about the Quartus II Programmer support for **.rbf** files.
November 2013	13.1.0	• Converted to DITA format. • Added JTAG Debug Mode for Partial Reconfiguration and Configuring Partial Reconfiguration Bitstream in JTAG Debug Mode sections.
November 2012	12.1.0	• Updated Table 18–3 on page 18–6, and Table 18–4 on page 18–8. • Added "Converting Programming Files for Partial Reconfiguration" on page 18–10, "Generating .pmsf using a .msf and a .sof" on page 18–10, "Generating .rbf for Partial Reconfiguration Using a .pmsf" on page 18–12, "Enable Decompression during Partial Reconfiguration Option" on page 18–14 • Updated "Scripting Support" on page 18–15.
June 2012	12.0.0	• Updated Table 18–5 on page 18–8. • Updated "Quartus II Programmer GUI" on page 18–3.
November 2011	11.1.0	• Updated "Configuration Modes" on page 18–5. • Added "Optional Programming or Configuration Files" on page 18–6. • Updated Table 18–2 on page 18–5.
May 2011	11.0.0	• Added links to Quartus II Help. • Updated "Hardware Setup" on page 21–4 and "JTAG Chain Debugger Tool" on page 21–4.
December 2010	10.1.0	• Changed to new document template. • Updated "JTAG Chain Debugger Example" on page 20–4. • Added links to Quartus II Help. • Reorganized chapter.

continued...

Date	Quartus Prime Version	Changes
July 2010	10.0.0	• Added links to Quartus II Help. • Deleted screen shots.
November 2009	9.1.0	No change to content.
March 2009	9.0.0	• Added a row to Table 21–4. • Changed references from "JTAG Chain Debug" to "JTAG Chain Debugger". • Updated figures.

3. Using the HPS Flash Programmer

The Quartus Prime software and Quartus Prime Programmer include the hard processor system (HPS) flash programmer. Hardware designs, such as HPS, incorporate flash memory on the board to store FPGA configuration data or HPS program data. The HPS flash programmer programs the data into a flash memory device connected to an Intel FPGA SoC. The programmer sends file contents over a download cable, such as the Intel FPGA Download Cable II, to the HPS and instructs the HPS to write the data to the flash memory.

The HPS flash programmer programs the following content types to flash memory:

- HPS software executable files — Many systems use flash memory to store non-volatile program code or firmware. HPS systems can boot from flash memory.

  Note: The HPS Flash Programmer is mainly intended to be used for programming the Preloader image to QSPI or NAND flash. Because of the low speed of operation, it is not recommended to be used for programming large files.
- FPGA configuration data — At system power-up, the FPGA configuration controller on the board or HPS read FPGA configuration data from the flash memory to program the FPGA. The configuration controller or HPS may be able to choose between multiple FPGA configuration files stored in flash memory.
- Other arbitrary data files — The HPS flash programmer programs a binary file to any location in a flash memory for any purpose. For example, a HPS program can use this data as a coefficient table or a sine lookup table.

The HPS flash programmer programs the following memory types:

- Quad serial peripheral interface (QSPI) Flash
- Open NAND Flash Interface (ONFI) compliant NAND Flash

3.1. Supported Devices

Table 41. HPS Flash Programmer-supported Devices

	Cyclone V SoC, Arria V SoC	Arria 10 SoC	Stratix 10 SoC, Agilex 7 F-Series/I-Series/M-Series SoC	Agilex 5 E-Series/D-Series SoC
HPS Flash Programmer	Supported	Supported	Not Supported *	Not Supported *

Note: For Stratix 10 SoC, Agilex 7 SoC, or Agilex 5 SoC devices, you must program the flash connected to HPS. You have the following options:

- Use a bus switch to route the QSPI signals to an external master that does the programming.
- Use software running on HPS to do the programming. For example, you can load U-Boot with an Arm* debugger or System Console, and then use it to program the flash.

Note: On Stratix 10 SoC, Agilex 7 SoC, or Agilex 5 SoC devices, the HPS can access the QSPI flash memory connected to SDM. This flash is programmed using the Quartus Prime Programmer tool that is part of the Quartus Prime Pro Edition software. For additional information about the use of the Quartus Prime Programmer, refer to your device's boot user guides.

Related Information

- Hard Processor System Booting User Guide: Agilex 5 SoCs
- Agilex 7 SoC FPGA Boot User Guide
- Stratix 10 SoC FPGA Boot User Guide
- Arria 10 SoC FPGA Boot User Guide
- HPS SoC Boot Guide - Cyclone V SoC Development Kit

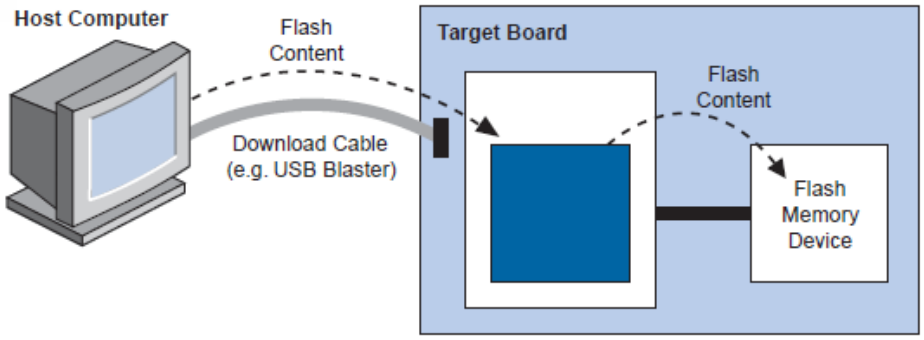
3.2. HPS Flash Programmer Command-Line Utility

Run the HPS flash programmer from the command line. The HPS flash programmer is located in the <*Quartus Prime installation directory*>/quartus/bin directory.

3.3. How the HPS Flash Programmer Works

The HPS flash programmer is divided into a host and a target. The host portion runs on your computer and sends flash programming files and programming instructions over a download cable to the target. The target portion is the HPS in the SoC. The target accepts the programming data flash content and required information about the target flash memory device sent by the host. The target writes the data to the flash memory device.

Figure 38. HPS Flash Programmer



The HPS flash programmer determines the type of flash to program by sampling the boot select (BSEL) pins during cold reset; you do not need to specify the type of flash to program.

3.4. Using the Flash Programmer from the Command Line

3.4.1. HPS Flash Programmer

The HPS flash programmer utility can erase, blank-check, program, verify, and examine the flash. The utility accepts a Binary File with a required "**.bin**" extension.

The HPS flash programmer command-line syntax is:

```
quartus_hps <options> <file.bin>
```

Note: The HPS flash programmer uses byte addressing.

Table 42. HPS Flash Programmer Parameters

Option	Short Option	Required	Description
`--addr`	`-a`	Yes (if the start address is not 0)	This option specifies the start address of the operation to be performed.
`--cable`	`-c`	Yes	This option specifies what download cable to use. To obtain the list of programming cables, run the command "jtagconfig". It lists the available cables, like in the following example: `jtagconfig` • Intel FPGA Download Cable [USB-0] • Intel FPGA Download Cable [USB-1] • Intel FPGA Download Cable [USB-2] The "`-c`" parameter can be the number of the programming cable, or its name. The following are valid examples for the above case: • `-c 1` • `-c` "Intel FPGA Download Cable [USB-2]"
`--device`	`-d`	Yes (if there are multiple HPS devices in the chain)	This option specifies the index of the HPS device. The tool automatically detects the chain and determine the position of the HPS device; however, if there are multiple HPS devices in the chain, the targeted device index must be specified.
`--boot`	N/A	Yes	Option to reconfigure the HPS IOCSR and PINMUX before starting flash programming. For the Quartus Prime HPS Flash Programmer options: • Warm/cold reset HPS (BootROM) so that BootROM can reconfigure the setting. FPGA (for Arria 10) is `nconfig`. • Explicitly configure dedicated I/O and PINMUX Available options: • 1 - Set Breakpoint to halt CPU, warm reset HPS [not recommended][6] • 2 - Set Watchpoint to halt CPU, warm reset HPS[7]

continued...

[6] Warm reset HPS for BootROM to configure dedicated IO and PINMUX, and use a breakpoint to halt CPU.

Option	Short Option	Required	Description
			• 3 - Explicitly configure dedicated IO and PINMUX(8) • 16 - Cold reset HPS(9) Cyclone V and Arria 10 support values: 1, 2 and 16. Arria 10 supports values: 2, 3 and 16 *Note:* For the first three options, add up integer of 16, so that the HPS cold reset is performed Available options for a cold reset before the flow: • 17 - Cold reset HPS + breakpoint(10) • 18 - Cold reset HPS + watchpoint(11) • 19 - Cold reset HPS + configure dedicated IO and PINMUX(12)
`--exit_xip`	N/A	Yes (if the QSPI flash device has been put into XIP mode)	This option exits the QSPI flash device from XIP mode. A non-zero value has to be specified for the argument. For example, `quartus_hps -c <cable> -o <operation> --exit_xip=0x80`.
`--operation`	`-o`	Yes	This option specifies the operation to be performed. The following operations are supported: • I—Read IDCODE of SOC device and discover Access Port • S—Read Silicon ID of the flash • E—Erase flash • B—Blank-check flash • P—Program flash • V—Verify flash • EB—Erase and blank-check flash • BP—Program <*BlankCheck*> flash • PV—Program and verify flash • BPV—Program (blank-check) and verify flash • X—Examine flash *Note:* The program begins with erasing the flash operation before programming the flash by default.
`--size`	`-s`	No	This option specifies the number of bytes of data to be performed by the operation. `size` is optional.

continued...

(7) Warm reset HPS for BootROM to configure dedicated IO and PINMUX, and use a watchpoint to halt CPU.

(8) Explicitly configure dedicated IO and PINMUX.

(9) Bit-wise on top of the flow, if you set the bit, the tool will perform cold reset first

(10) Cold reset HPS for BootROM to configure dedicated IO and PINMUX, and use a breakpoint to halt CPU.

(11) Cold reset HPS for BootROM to configure dedicated IO and PINMUX, and use a watchpoint to halt CPU.

(12) Cold reset HPS, then explicitly configure dedicated IO and PINMUX.

Option	Short Option	Required	Description
Note: The following options: `repeat` and `interval` must be used together and are optional. The HPS BOOT flow supports up to four images where each image is identical. These options duplicate the operation data; therefore you do not need embedded software to create a large file containing duplicate images.			
`--repeat`	`-t`	No	`repeat`—specifies the number of duplicate images for the operation to perform.
`--interval`	`-i`	No	`interval`—specifies the repeated address. The default value is 64 kilobytes (KB).

3.4.2. HPS Flash Programmer Command Line Examples

Type `quartus_hps --help` to obtain information about usage. You can also type `quartus_hps --help=<option>` to obtain more details about each option. For example "quartus_hps --help=o".

Example 2. Program File to Address 0 of Flash

`quartus_hps –c 1 –o P input.bin` programs the input file (**input.bin**) into the flash, starting at flash address 0 using a cable 1.

Example 3. Program First 500 Bytes of File to Flash (Decimal)

`quartus_hps –c 1 –o PV –a 1024 –s 500 input.bin` programs the first 500 bytes of the input file (**input.bin**) into the flash, starting at flash address 1024, followed by a verification using a cable 1.

Note: Without the prefix "0x" for the flash address, the tool assumes it is decimal.

Example 4. Program First 500 Bytes of File to Flash (Hexadecimal)

`quartus_hps –c 1 –o PV –a 0x400 –s 500 input.bin` programs the first 500 bytes of the input file (**input.bin**) into the flash, starting at flash address 1024, followed by a verification using a cable *1*.

Note: With the prefix 0x, the tool assumes it is hexadecimal.

Example 5. Program File to Flash Repeating Twice at Every 1 MB

`quartus_hps –c 1 –o BPV –t 2 –i 0x100000 input.bin` programs the input file (**input.bin**) into the flash, using a cable 1. The operation repeats itself twice at every 1 megabyte (MB) of the flash address. Before the program operation, the tool ensures the flash is blank. After the program operation, the tool verifies the data programmed.

Example 6. Erase Flash on the Flash Addresses

`quartus_hps –c 1 –o EB input.bin` erases the flash on the flash addresses where the input file (**input.bin**) resides, followed by a blank-check using a cable 1.

Example 7. Erase Full Chip

`quartus_hps –c 1 –o E` erases the full chip, using a cable 1. When no input file (**input.bin**) is specified, it erases all the flash contents.

Example 8. Erase Specified Memory Contents of Flash

`quartus_hps –c 1 –o E –a 0x100000 –s 0x400000` erases specified memory contents of the flash. For example, 4 MB worth of memory content residing in the flash address, starting at 1 MB, are erased using a cable 1.

Example 9. Examine Data from Flash

`quartus_hps –c 1 –o X –a 0x98679 –s 56789 output.bin` examines 56789 bytes of data from the flash with a 0x98679 flash start address, using a cable 1.

3.5. Supported Memory Devices

Table 43. QSPI Flash

Flash Device	Manufacturer	Device ID	DIE #	Density (Mb)	Voltage
M25P40	Micron	0x132020	1	4	3.3
N25Q064	Micron	0x17BA20	1	64	3.3
N25Q128	Micron	0x18BA20	1	128	3.3
N25Q128	Micron	0x18BB20	1	128	1.8
N25Q256	Micron	0x19BA20	1	256	3.3
N25Q256	Micron	0x19BB20	1	256	1.8
MT25QL512	Micron	0x20BA20	1	512	3.3
N25Q512	Micron	0x20BA20	2	512	3.3
MT25QU512	Micron	0x20BB20	1	512	1.8
N25Q512A	Micron	0x20BB20	2	512	1.8
N25Q00AA	Micron	0x21BA20	4	1024	3.3
MT25QU01G	Micron	0x21BB20	2	1024	1.8
N25Q00AA	Micron	0x21BB20	4	1024	1.8
MT25QL02G	Micron	0x22BA20	4	2048	3.3
MT25QU02G	Micron	0x22BB20	4	2048	1.8
S25FL128S	Cypress	0x182001	1	128 (64KB Sectors)	3.3
S25FL128S	Cypress	0x182001	1	128 (256KB Sectors)	3.3
S25FL256S	Cypress	0x190201	1	256 (64KB Sectors)	3.3
S25FL256S	Cypress	0x190201	1	256 (256KB Sectors)	3.3
S25FL512S	Cypress	0x200201	1	512	3.3
MX25L12835E	Macronix	0x1820C2	1	128	3.3
MX25L25635	Macronix	0x1920C2	1	256	3.3
MX66L51235F	Macronix	0x1A20C2	1	512	3.3

continued...

Flash Device	Manufacturer	Device ID	DIE #	Density (Mb)	Voltage
MX66L1G45	Macronix	0x1B20C2	1	1024	3.3
MX66U51235	Macronix	0x3A25C2	1	512	1.8
GD25Q127C	GigaDevice	0x1840C8	1	128	3.3

Table 44. ONFI Compliant NAND Flash

Manufacturer	MFC ID	Device ID	Density (Gb)
Micron	0x2C	0x68	32
Micron	0x2C	0x48	16
Micron	0x2C	0xA1	8
Micron	0x2C	0xF1	8

Note: The above table contains just examples of supported devices. The HPS Flash Programmer supports all ONFI compliant NAND flash devices that are supported by the HPS QSPI Flash Controller.

Note: The HPS Flash Programmer supports Arria 10 devices. For more information, refer to the "Supported Flash Devices for Arria 10" web page.

Related Information

Supported Flash Devices for Arria 10 SoCs

3.6. HPS Flash Programmer User Guide Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Added *Supported Devices* topic.
2023.04.03	23.1	• Updated product family name to "Intel Agilex 7."
2020.12.14	20.4	Moved HPS Flash Programmer User Guide from *Intel SoC FPGA Embedded Development Suite User Guide* to *Intel Quartus Prime Pro Edition User Guide: Programmer*.

Document Version	Changes
2020.08.07	Maintenance release
2020.05.29	Added descriptions for the bit-wise values not displayed in help output
2019.12.20	Supported with Quartus Prime Standard Edition version 19.1 and Quartus Prime Pro Edition version 19.3 • Maintenance release
2019.05.16	*HPS Flash Programmer*: Documented --boot=18 for QSPI programming
2018.09.24	Added supported memory devices in the "QSPI Flash" table; and updated voltages for several flash devices.
2018.06.18	• Updated chapter to include support for Stratix 10 • Updated chapter to include support for Quartus Prime Pro Edition
2017.05.08	• Intel FPGA rebranding • Rebranded paths and tools for the Standard and Professional versions

continued...

Document Version	Changes
2016.11.07	Maintenance release
2016.05.27	Maintenance release
2016.02.17	Added QSPI Flash part number to the QSPI Flash table in the "Supported Memory Devices" chapter
2015.08.06	Added Arria 10 support

683039 | 2024.04.01

Send Feedback

A. Quartus Prime Pro Edition User Guide: Programmer Document Archive

For the latest and previous versions of this user guide, refer to Quartus Prime Pro Edition User Guide: Programmer. If a software version is not listed, the reference manual for the previous software version applies.

B. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Intel® Quartus® Prime Pro Edition User Guide

Block-Based Design

Updated for Intel® Quartus® Prime Design Suite: **23.3**

This document is part of a collection - Intel® Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

1. Block-Based Design Flows

The Intel® Quartus® Prime Pro Edition software supports block-based design flows, also known as modular or hierarchical design flows.

You can designate a design block as a design partition in order to reuse the block. A design partition is a logical, named, hierarchical boundary assignment that you can apply to a design instance. Block-based design flows enable the following:

- Design block reuse—export and reuse of design blocks in other projects

You can reuse design blocks with the same periphery configuration, share a synthesized design block with another designer, or replicate placed and routed IP in another project. Design, implement, and verify core or periphery blocks once, and then reuse those blocks multiple times across different projects that use the same device.

1.1. Block-Based Design Terminology

This document refers to the following terms to explain block-based design methods:

Table 1. Block-Based Design Terminology

Term	Description
Black Box File	RTL source file that contains only port and module or entity definitions, without any logic. Include parameters or generics passed to the module or entity to ensure that the configuration matches the implementation in the Consumer project.
Block	Logic that comprises a hierarchical design instance, typically represented by a Verilog module or VHDL entity. You designate a design block as a design partition to empty or export the block.
Consumer	A Consumer can reuse a design partition that a Developer exports as a Partition Database File (`.qdb`) from another project.
Core Partition	A design partition that contains only FPGA resources for the implementation of core logic, such as LUTs, registers, M20K memory blocks, and DSPs. A core partition cannot include periphery resources.
Design Partition	A logical, named, hierarchical boundary assignment that you can apply to a design instance. Creating a partition creates a logical boundary and prevents logic optimization and merging with parent or child partitions. Design partitions facilitate incremental block-based compilation and design block reuse by logically separating instances.
Developer	A Developer creates and exports a design partition as a `.qdb` for use in a Consumer project.
Floorplanning	Planning the physical layout of FPGA device resources. The manual process of assigning the logical design hierarchy and periphery to physical regions in the device and I/O.

continued...

Term	Description
Logic Lock Region Constraints	Constrains the placement and routing of logic to a specific region in the target device. Specify the region origin, height, width, and any of the following options: • **Reserved**—prevents the Fitter from placing non-member logic within the region. • **Core-Only**—applies the constraint only to core logic in the region, and does not include periphery logic in the region. • **Routing Region**—restricts the routing of connections between region members to the specified area. The routing region is non-exclusive. Other resources in the parent or sibling hierarchy levels can use that routing area. You can restrict the routing region to areas equal to or larger than the Logic Lock region, up to the entire chip. The default routing region is the entire chip. • **Size/State**—fixes the size and locks the state of the region. The **Fixed/Locked** option defines a region of fixed size and locked location. The **Auto/Floating** option defines a region with a floating location that automatically sizes to the logic.
Project	The Intel Quartus Prime software organizes the source files, settings, and constraints within a project of one or more revisions. The Intel Quartus Prime Project File (`.qpf`) stores the project name and references each project revision that you create.
Root Partition	The Intel Quartus Prime software automatically creates a top-level "root_partition" with a hierarchy path of |for each project revision. The root partition includes all device periphery resources (such as I/O, HSSIO, memory interfaces, and PCIe*) and associated core resources. You can export and reuse periphery resources by exporting the root partition and reserving a region for subsequent development (the reserved core) by a Consumer.
Snapshot	A snapshot is a view of the design after a compilation stage. The Intel Quartus Prime Compiler generates a snapshot of the compilation database after each compilation stage. You can export a specific snapshot for incremental block-based compilation, design block reuse, and team based designs.

1.2. Block-Based Design Overview

This section provides an overview of design block reuse and incremental block-based compilation flows. Design Block Reuse Flows on page 15 and Incremental Block-Based Compilation Flow on page 25 describe the step-by-step details of these block-based flows.

1.2.1. Design Block Reuse Overview

In design block reuse flows, you export a core or root partition for reuse in another project that targets the same Intel FPGA device family. You can share one of the following compilation snapshots for a partition across projects or with other designers:

- Synthesized snapshot
- Final snapshot

Core partition reuse enables preservation and export of compilation results for a core partition. Reuse of the core partition allows an IP developer to create and optimize an IP once and share it across multiple projects.

Root partition reuse enables preservation and export of compilation results for a top-level (or root) partition that describes the device periphery, along with associated core logic. Reuse of the periphery allows a board developer to create and optimize a platform design with device periphery logic once, and then share that root partition

with other board users who create custom core logic. The periphery resources include all the hardened IP in the device periphery, such as general purpose I/O, PLLs, high-speed transceivers, PCIe, and external memory interfaces.

Team members can work on different partitions separately, and then bring them together later, facilitating a team-based design environment. A team lead integrates the partitions in the system and provides guidance to ensure that each partition uses the appropriate device resource and achieves design requirements during the full design integration. A Developer initially creates and exports a block as a partition in one Intel Quartus Prime project. Subsequently, a *Consumer* reuses the partition in a different project.(1) To avoid resource conflicts, floorplanning is essential when reusing final snapshot partitions.

Related Information

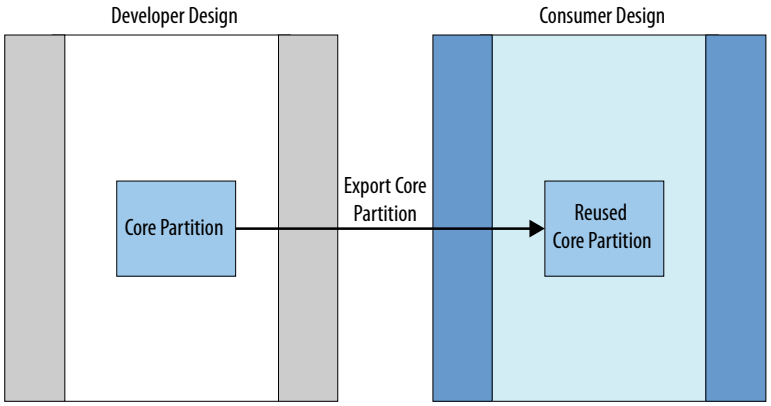
Design Block Reuse Flows on page 15

1.2.1.1. Design Block Reuse Examples

Core Partition Reuse Example

In a typical core partition reuse example, a Developer exports a core partition that already meets design requirements. The Developer optimizes and exports the block, and then the Consumer can simply reuse the block without requiring re-optimization in the Consumer project that targets the same device family.

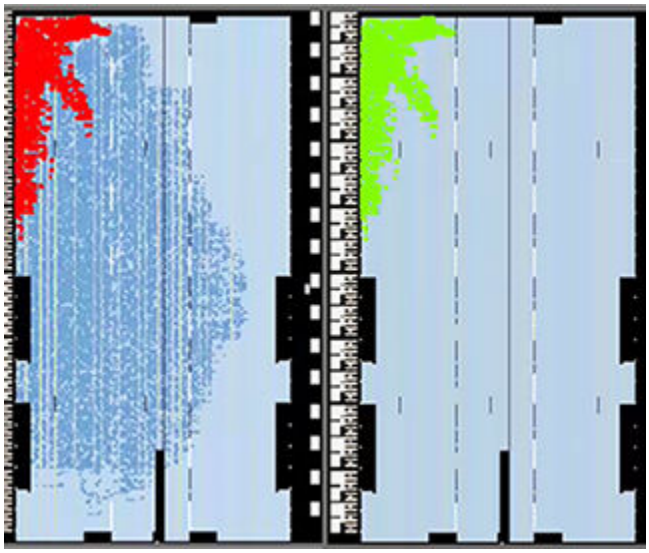
Figure 1. Core Partition Reuse Example



You can export a core block with unique characteristics that you want to retain, and then replicate that functionality or physical implementation in other projects. In the following figure, a Developer reuses the red-colored partition in the floorplan in another project shown in green in the floorplan on the right.

(1) For brevity, this document uses Developer to indicate the person or project that originates a reusable block, and uses Consumer to indicate the person or project that consumes a reusable block.

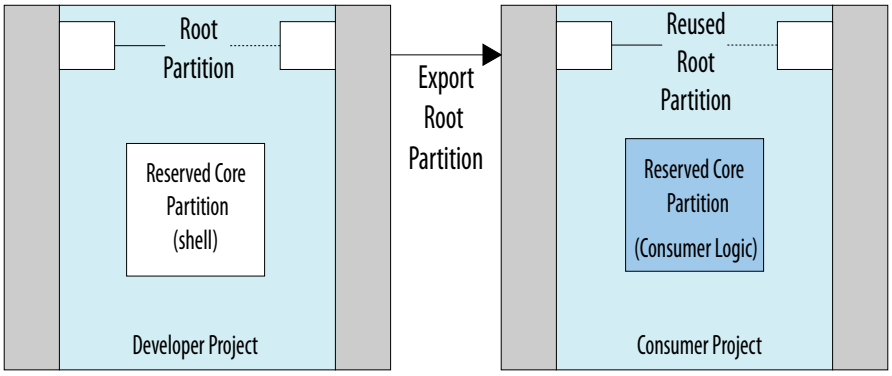
Figure 2. IP Replication and Physical Implementation



Root Partition Reuse Example

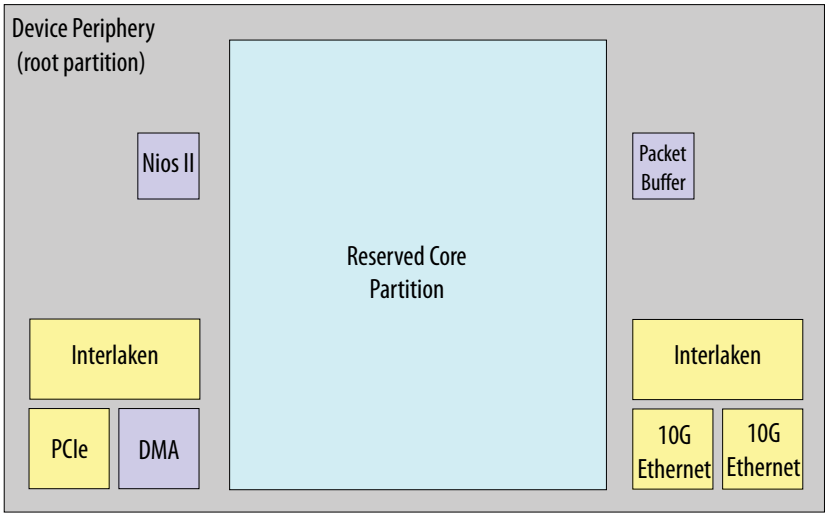
In a typical root partition reuse example, a Developer defines a root partition that includes periphery and core resources that are appropriate for reuse in other projects. An example of this scenario is reuse of the periphery for a development kit that can be reused by multiple Developers and projects.

Figure 3. Root Partition Reuse Example



In root partition reuse, each project must target the same Intel FPGA device, must have the same interfaces, and use the same version of the Intel Quartus Prime Pro Edition software. The following example shows reuse of an optimized root partition that contains various periphery interfaces. Only the reserved core partition that contains custom logic changes between Consumer projects.

Figure 4. Root Partition Reuse Example



You can reuse the root partition with multiple other boards. The root partition Developer creates a design that meets expected design requirements with all the required resources, while also reserving a region (the reserved core) for Consumer development. The Developer then exports the root partition as a `.qdb` file and passes the `.qdb` to the Consumers of the partition.

The Consumer reuses the root partition, and adds their own RTL for the reserved core. This flow allows for development on several different boards with a common root partition. Reusing the root partition saves the Consumer development time, because the root partition is pre-optimized by the Developer.

1.2.2. Incremental Block-Based Compilation Overview

You can use incremental block-based compilation to reduce overall compile time through design abstraction with empty design partitions.

You can specify a design partition as *Empty* to represent parts of your design that are incomplete or missing. Setting a partition to **Empty** can reduce the total compilation time if the Compiler does not process design logic associated with the empty partition.

Empty partitions allow you to:

- Set aside incomplete portions of the design, mark them as **Empty**, and complete the design incrementally.
- Designate complete portions of the design as **Empty** to focus all subsequent Compiler efforts on uncompleted portions of the design

1.3. Design Methodologies Overview

Block-based design flows support top-down, bottom-up, and team-based design methodologies. The following sections describe these methods.

1.3.1. Top-Down Design Methodology Overview

In a top-down design methodology, you plan the design at the top level, and then split the design into lower-level design blocks. Different developers or IP providers can create and verify HDL code for the lower-level design blocks separately, but one team lead manages the implementation project for the entire design.

In the top-down methodology, you can use core block reuse to preserve the core logic by exporting the `.qdb` file. Another developer working on the same project can then reuse the core logic `.qdb` file.

You can similarly preserve the root partition by exporting the root `.qdb` file, using the root partition reuse flow, and then reusing that root partition in the same project.

1.3.2. Bottom-Up Design Methodology Overview

In a bottom-up design methodology, you create lower-level design blocks independently of one another, and then integrate the blocks at the top-level.

To implement a bottom-up design, individual developers or IP providers can complete the placement and routing optimization of their design in separate projects, and then reuse lower-level blocks in the top-level project. This methodology can be useful for team-based design flows with developers in other locations, or when third-parties create design blocks.

However, when developing design blocks independently in a bottom-up design flow, individual developers may not have all the information about the overall design, or understand how their block connects with other blocks. The absence of this information can lead to problems during system integration, such as difficulties with timing closure, or resource conflicts. To reduce such difficulties, plan the design at the top level, whether optimizing within a single project, or optimizing blocks independently in separate projects, for subsequent top-level integration.

Teams that use a bottom-up design method can optimize placement and routing of design partitions independently. However, the following drawbacks can occur when optimizing the design partitions in separate projects:

- Achieving timing closure for the full design may be more difficult if you compile partitions independently without information about other partitions in the design. Avoiding this problem requires careful timing budgeting and observance of design rules, such as always registering the ports at the module boundaries.
- The design requires resource planning and allocation to avoid resource conflicts and overuse. Floorplanning with Logic Lock regions can help you avoid resource conflicts while developing each part independently in a separate Intel Quartus Prime project.
- Maintaining consistency of assignments and timing constraints is more difficult if you use separate Intel Quartus Prime projects. The team lead must ensure that the assignments and constraints of the top-level design, and those developers define in the separate projects and reuse at the top-level, are consistent.

Partitions that you develop independently all must share a common set of resources. To minimize issues that can arise when sharing a common set of resources between different partitions, create the partitions within in a single project, or in copies of the top-level project, with full design-level constraints, to ensure that resources do not overlap. Correct use of partitions and Logic Lock regions can help to minimize issues that can arise when integrating into the top-level design.

If a developer has no information about the top-level design, the team lead must at least provide a specific Intel FPGA device part number, along with any required physical timing constraints. The developer can then create and export the partition from a separate project. When a developer lacks information, the developer should overconstrain or create additional timing margin on the critical paths. The technique helps to reduce the chance of timing problems when integrating the partitions with other blocks.

You can use the bottom-up design methodology in conjunction with the core partition reuse flow to independently develop and export a core partition `.qdb` for reuse by a the team lead.

Related Information

Bottom-Up Design Considerations on page 28

1.3.3. Team-Based Design Methodology Overview

In a team-based design methodology, a team lead sets up the top-level project and constraints (including the top-level clock, I/O, and inter-partition constraints), and determines which portions of the design that other team members develop.

Team-based design combines a top-down methodology (where all developers must be aware of the top-level project structure and constraints), with elements of bottom-up flows (where developers work separately on lower-level blocks and integrate them into the top-level). In the top-down methodology, you can preserve the partitions and provides a `.qdb` file to other team members working on the same project.

The project lead must ensure that the top-level project contains all the interfaces for the design blocks that other team leaders add later. Each team member then develops their portion of the design, and may specify other constraints specific to their design partition. The team members implement the individual design blocks in the context of the top-level design, to avoid integration issues later. As the project nears completion, the team lead then integrates partitions from team members into the top-level project, accounting for any new constraints for the imported partitions.

Individual team members can optionally work on a copy of the same top-level project. The team member creates a partition for their respective design block, compiles the design, and then exports the partition. The team lead then integrates each design partition into the top-level design.

To simplify full design optimization, allow full-chip placement and routing of the partition at the top-level. Export and reuse only the synthesized snapshot, unless the top-level design requires optimized post-fit results.

Related Information

Setting-Up Team-Based Designs on page 26

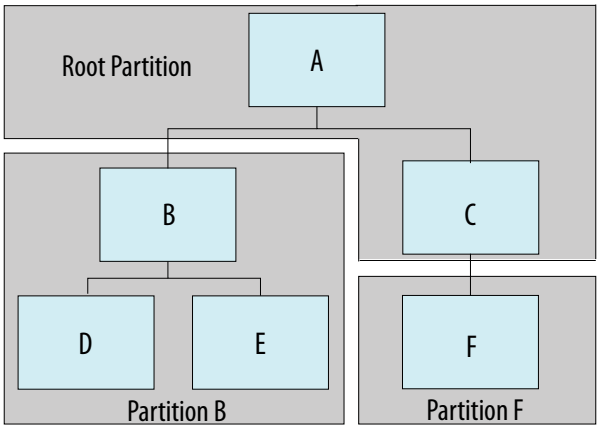
1.4. Design Partitioning

To use block-based design flows, you must first create design partitions from your design's hierarchical instances. The Compiler then treats the design partitions separately to allow the block-based functionality.

The Intel Quartus Prime software automatically creates a top-level (|) "root_partition" for each project revision. The root partition contains all the periphery resources, and may also include core resources. When you export the root partition for reuse, the exported partition excludes all logic in reserved core partitions. To export and reuse periphery elements, you export the root partition.

When you create partitions, every hierarchy within that partition becomes part of the parent partition. The partition creates a logical boundary that prevents merging or optimizing between partitions. The following *Design Partitions in Design Hierarchy* diagram illustrates design partition relationships and boundaries.

Figure 5. Design Partitions in Design Hierarchy



Note:

- Instances B and F are design partitions.
- Partition B includes sub-instances D and E.
- The root partition contains the top-level instance A and instance C, because C is unassigned to any partition.

Design partitions facilitate incremental block-based compilation and design block reuse by logically separating instances. This logical separation allows the Compiler to synthesize and optimize each partition separately from the other parts of the design. The logical separation also prevents Compiler optimizations across partition boundaries.

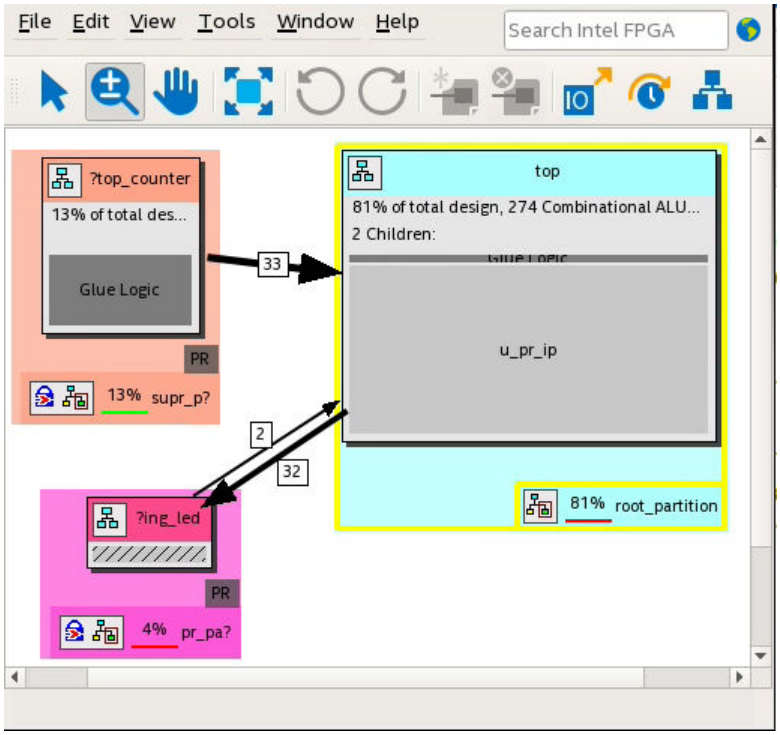
Block-based design requires that you plan and structure the source code and design hierarchy to ensure proper logic grouping for optimization. Implementing the correct logic grouping is easiest early in the design cycle.

Creating or removing a design partition changes the synthesis and subsequent physical implementation and quality of results. When planning the design hierarchy, be aware of the size and scope of each partition, and the possibility of different parts of the design changing during development. Separate logic that changes frequently from the fixed parts of the design.

Group design blocks in your design hierarchy so that highly-connected blocks have a shared level of design hierarchy for assignment to one partition. Structuring your design hierarchy appropriately reduces the required number of partition boundaries, and allows maximum optimization within the partition.

The Design Partition Planner (**Tools ➤ Design Partition Planner**) helps you to visualize and refine a design's partitioning scheme by showing timing information, relative connectivity densities, and the physical placement of partitions. You can locate partitions in other viewers, or modify or delete partitions in the Design Partition Planner.

Figure 6. Design Partition Planner



Consider creating each design entity that represents a partition instance in a separate source file. This approach helps you correlate which partitions require recompilation when you make source code changes. As you make design changes, you can designate partitions as empty or preserved (using the `.qdb` file) to instruct the Compiler which partitions to recompile from source code, as Design Abstraction on page 25 describes.

If your design has timing-critical partitions that are changing through the design flow, or partitions exported from another project, use design floorplan assignments to constrain the placement of the affected partitions. A properly partitioned and floor-planned design enables partitions to meet top-level design requirements when you integrate the partitions with the rest of your design. Poorly planned partitions or floorplan assignments negatively impact design area utilization and performance, thereby increasing the difficulty of timing closure.

The following design partition guidelines help ensure the most effective and efficient results. Block-based design flows add steps and requirements to the design process, but can provide significant benefits in design productivity.

Related Information

Step 1: Developer: Create a Design Partition on page 16

1.4.1. Planning Partitions for Periphery IP, Clocks, and PLLs

Use the following guidelines to plan partitions for periphery IP, clocks, and PLLs:

Planning Partitions for Periphery IP

- Plan the design periphery to segregate and implement periphery resources in the root partition. Ensure that IP blocks that utilize both core and periphery resources (such as transceiver and external memory interface Intel FPGA IP) are part of the root partition.
- When creating design partitions for an existing design, remove all periphery resources from any entity you want to designate as a core partition. Also, tunnel any periphery resource ports to the top level of the design. Implement the periphery resource in the root partition.
- You cannot designate instances that use periphery resources as separate partitions. In addition, you cannot split an Intel FPGA IP core into more than one partition.
- The Intel Quartus Prime software generates an error if you include periphery interface Intel FPGA IP cores in any partition other than the top-level root partition.
- You must include Intel FPGA IP cores for the Hybrid Memory Cube (HBM) or Hard Processor System (HPS) in the root partition.

Planning Partitions for Clocks and PLLs

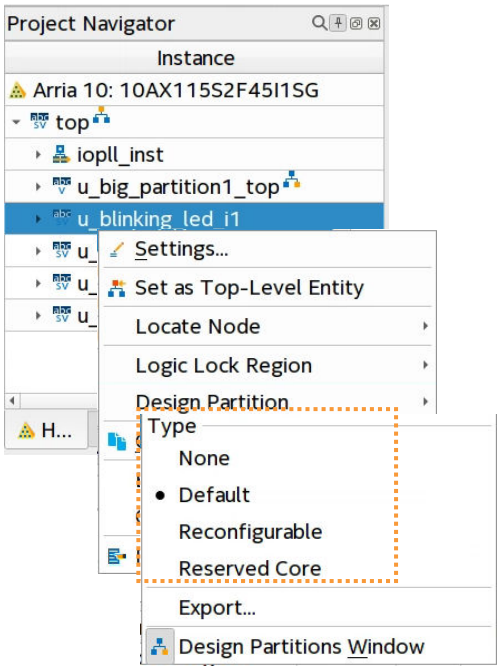
- Plan clocking structures to retain all PLLs and corresponding clocking logic in the root partition. This technique allows the Compiler to control PLLs in the root partition, if necessary.
- Consider creating a design block for all clocking logic that you instantiate in the top-level of the design. This technique ensures that the Compiler groups clocking logic together, and that the Compiler treats clocking logic as part of the root partition. Clock routing resources belong to the root partition, but the Compiler does not preserve routing resources with a partition.
- Include any signal that you want to drive globally in the root partition, rather than the core partition. Signals (such as clocks or resets) that you generate inside core partitions cannot drive to global networks without a clock buffer in the root partition.
- To support existing Intel Arria® 10 designs, the Compiler allows I/O PLLs in core partitions. However, creating a partition boundary prevents such PLLs from merging with other PLLs. The design may use more PLLs without this merging, and may have suboptimal clocking architecture.

1.4.2. Creating Design Partitions

Follow these steps to create and modify design partitions:

1. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
2. In the Project Navigator, right-click an instance in the **Hierarchy** tab, point to **Design Partition**, and click a design partition **Type**. A design partition icon appears next to each instance you assign.

Figure 7. Creating a Design Partition from the Project Hierarchy

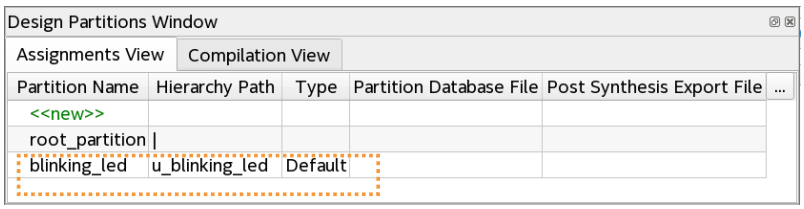


This setting corresponds to the following assignment in the `.qsf`:

```
set_instance_assignment -name PARTITION <name> \
     -to <partition hierarchical path>
```

3. To view and edit all design partitions in the project, click **Assignments ➤ Design Partitions Window**.

Figure 8. Design Partitions Window



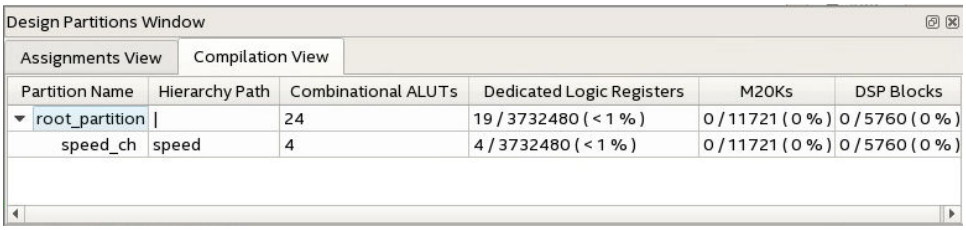
4. Specify the properties of the design partition in the Design Partitions Window. The following settings are available:

Table 2. Design Partition Settings

Option	Description
Partition Name	Specifies the partition name. Each partition name must be unique and consist of only alphanumeric characters. The Intel Quartus Prime software automatically creates a top-level (|) "root_partition" for each project revision.
Hierarchy Path	Specifies the hierarchy path of the entity instance that you assign to the partition. You specify this value in the **Create New Partition** dialog box. The root partition hierarchy path is |.
Type	Double-click to specify one of the following partition types that control how the Compiler processes and implements the partition: • **Default**—Identifies a standard partition. The Compiler processes the partition using the associated design source files. • **Reconfigurable**—Identifies a reconfigurable partition in a partial reconfiguration flow. Specify the **Reconfigurable** type to preserve synthesis results, while allowing refit of the partition in the PR flow. • **Reserved Core**—Identifies a partition in a block-based design flow that is reserved for core development by a Consumer reusing the device periphery.
Empty	Specifies an empty partition that the Compiler skips. This setting is incompatible with the **Reserved Core** and **Partition Database File** settings for the same partition.
Partition Database File	Specifies a Partition Database File (.qdb) that the Compiler uses during compilation of the partition. You export the .qdb for the stage of compilation that you want to reuse (synthesized or final). Assign the .qdb to a partition to reuse those results in another context.
Entity Re-binding	• PR Flow—specifies the entity that replaces the default persona in each implementation revision. • Root Partition Reuse Flow —specifies the entity that replaces the reserved core logic in the consumer project.
Color	Specifies the color-coding of the partition in the Chip Planner and Design Partition Planner displays.
Post Synthesis Export File	Automatically exports post-synthesis compilation results for the partition to the specified .qdb file each time Analysis & Synthesis runs. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.
Post Final Export File	Automatically exports post-final compilation results for the partition to the specified .qdb file each time the final stage of the Fitter runs. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.

Following compilation, you can view details about design partition implementation in the **Compilation View** tab of the Design Partitions Window. The synthesis and Fitter reports provide additional information about .qdb file assignments.

Figure 9. Design Partition Window Compilation View Tab



Note: You can only preserve paths inside a partition, and cannot preserve the paths crossing from one partition to another. Although you cannot merge partitions together, you can create a RTL wrapper to wrap modules that you want to group into a partition, and then assign a design partition to the RTL wrapper.

1.4.3. Design Partition Guidelines

Creating a design partition creates a logical hierarchical boundary around that instance. This partition boundary can limit the Compiler's ability to merge the partition's logic with other parts of the design. A partition boundary can also prevent optimization that reduces cell and interconnect delay, thereby reducing design performance. To minimize these effects, follow these general design partition guidelines:

- Register partition boundary ports. This practice can reduce unnecessary long delays by confining register-to-register timing paths to a single partition for optimization. This technique also minimizes the effect of the physical placement for boundary logic that the Compiler might place without knowledge of other partitions.
- Minimize the timing-critical paths passing in or out of design partitions. For timing critical-paths that cross partition boundaries, rework the partition boundaries to avoid these paths. Isolate timing-critical logic inside a single partition, so the Compiler can effectively optimize each partition independently.
- Avoid creating a large number of small partitions throughout the design. Excessive partitioning can impact performance by preventing design optimizations.
- Avoid grouping unrelated logic into a large partition. If you are working to optimize an independent block of your design, assigning that block as a small partition provides you more flexibility during optimization.

1.5. Design Block Reuse Flows

Design block reuse allows you to preserve a design partition as an exported `.qdb` file, and reuse this partition in another project. Reuse of core or root partitions involves partitioning and constraining the block prior to compilation, and then exporting the block for reuse in another project. Effective design block reuse requires planning to ensure that the source code and design hierarchy support the physical partitioning of device resources that these flows require.

- **Core partition reuse**—allows reuse of synthesized or final snapshots of a core partition. A core partition can include only core resources (LUTs, registers, M20K memory blocks, and DSPs).
- **Root partition reuse**—allows reuse of a synthesized or final snapshot of a root partition. A root partition includes periphery resources (including I/O, HSSIO, PCIe, PLLs), as well as any associated core resources, while leaving a core partition open for subsequent development.

At a high level, the core and root partition reuse flows are similar. Both flows preserve and reuse a design partition as a `.qdb` file. The Developer defines, compiles, and preserves the block in the Developer project, and the Consumer reuses the block in one or more Consumer projects.

The following sections describe the core and root partition reuse flows in detail.

Related Information

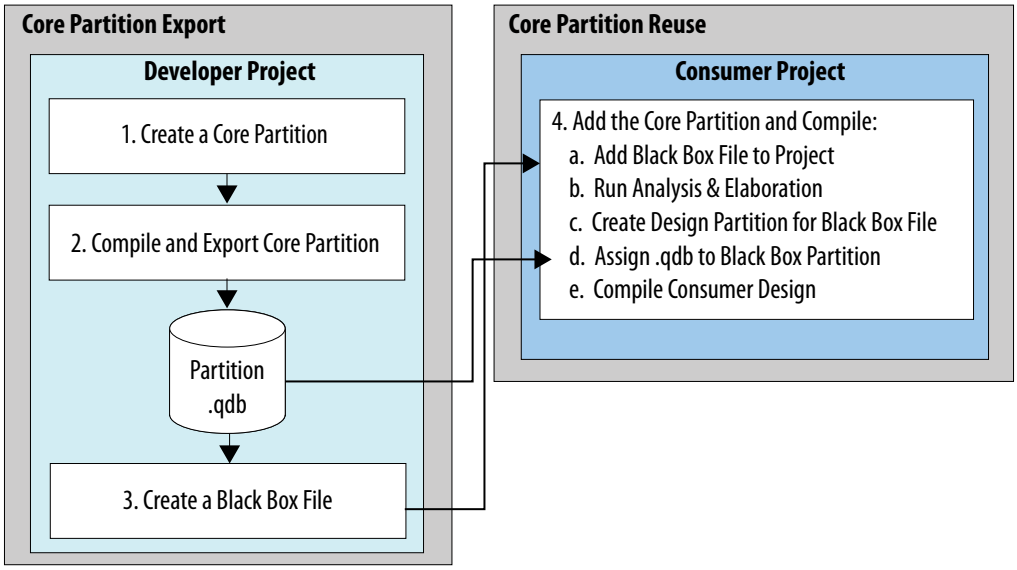
- Design Block Reuse Overview on page 4
- AN 839: Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board

1.5.1. Reusing Core Partitions

Reusing core partitions involves exporting the core partition from the Developer project as a `.qdb`, and then reusing the `.qdb` in a Consumer project.

The Consumer assigns the `.qdb` to an instance in the Consumer project. In the Consumer project, the Compiler runs stages not already exported with the partition.

Figure 10. Core Partition Reuse Flow



The following steps describe the core partition reuse flow in detail.

1.5.1.1. Step 1: Developer: Create a Design Partition

Define design partitions to create logical boundaries in the design hierarchy. Confine each core instance for export within a design partition. You can define partition instances from the Project Navigator or in the Design Partitions Window.

To define a core design partition:

1. Review the project to determine design elements suitable for reuse, and the appropriate snapshot for export.
2. Refer to Creating Design Partitions on page 13 to define a core partition. Select **Default** for the partition **Type**.

Related Information

Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration

1.5.1.2. Step 2: Developer: Compile and Export the Core Partition

This step describes generating and exporting a final snapshot of the core partition. You can manually export the core partition as a `.qdb` after compilation, or you can specify settings to automate export each time you compile. You can then reuse the core partition in the same project or in another project, starting the partition's compilation at the stage following the snapshot.

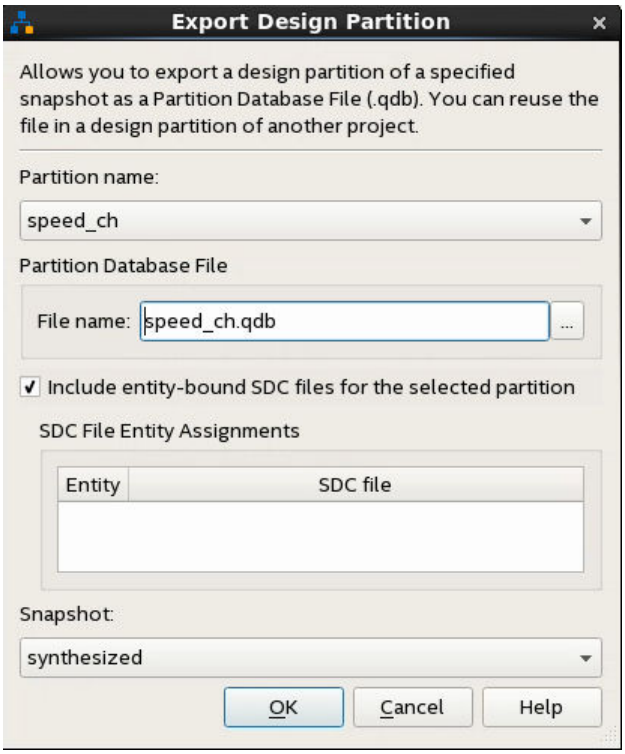
Manual Partition Export

To compile and manually export a core partition:

1. To run all compilation stages through Fitter (Finalize) and generate the final snapshot, click **Processing ➤ Start ➤ Start Fitter**.
2. To export the core partition, click **Project ➤ Export Design Partition**. Select the **Design Partition** name and the compilation **Snapshot** for export.
3. To include any entity-bound `.sdc` files in the exported `.qdb`, turn on **Include entity-bound SDC files for the selected partition**. By default, all Intel FPGA IP targeting Intel Stratix® 10 devices use entity-bound `.sdc` files.

   Note: Intel FPGA IP targeting Intel Arria 10 devices do not use entity-bound `.sdc` files by default. To use this option for Intel Arria 10 devices, you must first bind the `.sdc` file to the entity in the `.qsf`. Refer to "Using Entity-bound SDC Files," in *Intel Quartus Prime Pro Edition User Guide: Timing Analyzer*.
4. Confirm the **File name** for the **Partition Database File**, and then click **OK**.

Figure 11. Export Design Partition



The following command corresponds to partition export in the GUI:

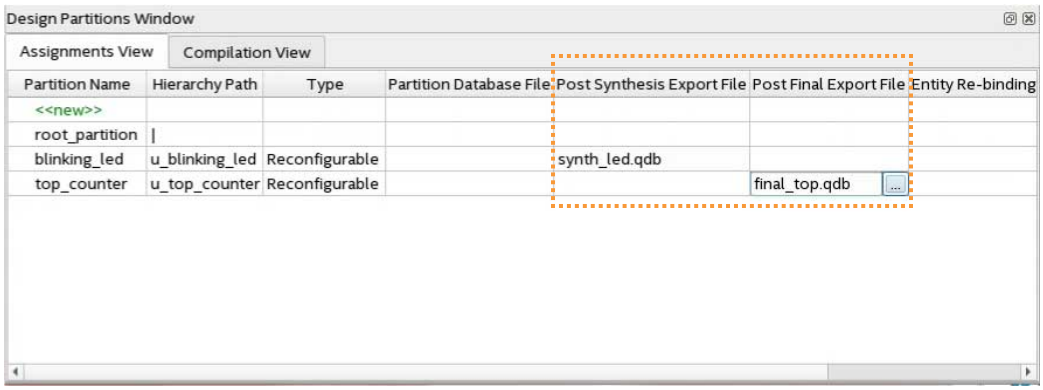
```
quartus_cdb <project name> -c <revision name> \
     --export_partition "<name>" --snapshot synthesized \
     --file <name>.qdb –-include_sdc_entity_in_partition
```

Automated Design Partition Export

Follow these steps in the Design Partitions Window to automatically export one or more design partitions following each compilation:

1. To automatically export a partition with synthesis results after each time you run synthesis, specify the a `.qdb` export path and file name for the **Post Synthesis Export File** option for that partition. If you specify only a file name without path, the file exports to the project directory after compilation.
2. To automatically export a partition with final snapshot results each time you run the Fitter, specify a `.qdb` file name for the **Post Final Export File** option for that partition.

Figure 12. Specifying Export File in Design Partitions Window



`.qsf` assignment syntax:

```
set_instance_assignment -name EXPORT_PARTITION_SNAPSHOT_SYNTHESIZED \
     <qdb file name> -to <hierarchy path> -entity <entity name>
```

Related Information

"Using Entity-bound SDC Files," Intel Quartus Prime Pro Edition User Guide: Timing Analyzer

1.5.1.3. Step 3: Developer: Create a Black Box File

Reusing a core partition `.qdb` file also requires that you add a supporting black box file to the Consumer project. A black box file is an RTL source file that only contains port and module or entity definitions, but does not contain any logic. The black box file defines the ports and port interface types for synthesis in the Consumer project. Follow these steps to create a block box port definitions file for the partition.

The Compiler analyzes and elaborates any RTL that you include in the black box file. Edits to the RTL do not affect a partition that uses a `.qdb` file.

1. Create an HDL file (`.v`, `.vhd`, `.sv`) that contains only the port definitions for the exported core partition. Include parameters or generics passed to the module or entity. For example:

```
module bus_shift #(
    parameter DEPTH=256,
    parameter WIDTH=8
  )(
```

```
    input clk,
    input enable,
    input reset,
    input [WIDTH-1:0] sr_in,
    output [WIDTH-1:0] sr_out
  );
endmodule
```

2. Provide the black box file and exported core partition `.qdb` file to the Consumer.

1.5.1.4. Step 4: Consumer: Add the Core Partition and Compile

To add the core partition, the Consumer adds the black box as a source file in the Consumer project. After design elaboration, the Consumer defines a core partition and assigns the exported `.qdb` file to an instance in the Design Partitions Window. Because the exported `.qdb` includes compiled netlist information, the Consumer project must target the same FPGA device family, and use the same Intel Quartus Prime software version, as the Developer project. The Consumer must supply a clock and any other constraints required for the interface to the core partition.

To add the core partition and compile the Consumer project:

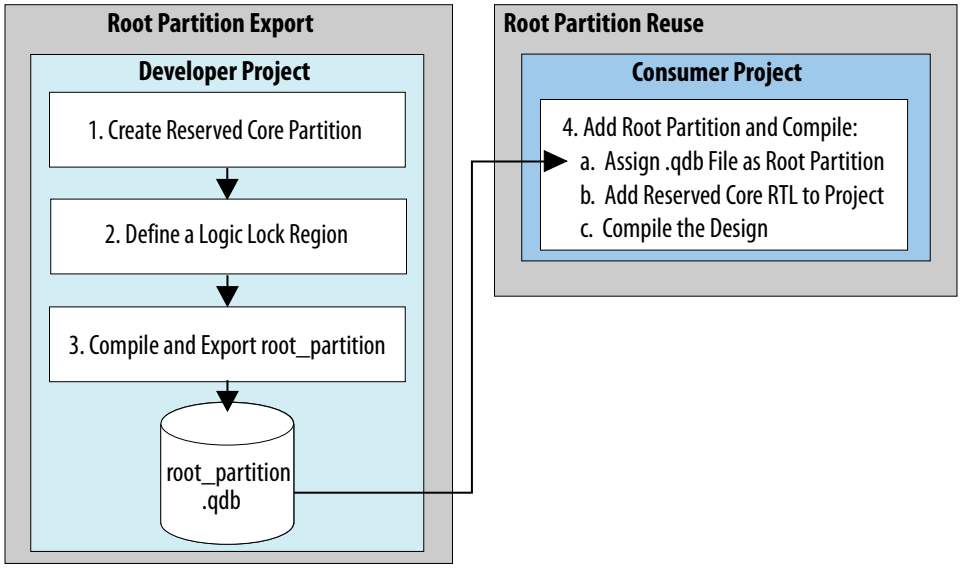
1. Create or open the Intel Quartus Prime project that you want to reuse the core partition.
2. To add one or more black box files to the consumer project, click **Project ➤ Add/Remove Files in Project** and select the black box file.
3. Follow the steps in Creating Design Partitions on page 13 to elaborate the design and define a core partition for the black box file. When defining the design partition, click the **Partition Database File** option and select the exported `.qdb` file for the core partition.
4. To run all compilation stages through Fitter (Finalize) and generate the final snapshot, click **Processing ➤ Start ➤ Start Fitter**.

1.5.2. Reusing Root Partitions

The root partition contains all the periphery resources, and may also include some core resources. To export and reuse periphery elements, you export the root partition. Reuse of root partitions allows you to design an FPGA-to-board interface and associated logic once, and then replicate that interface in other projects.

Note: When reusing the root partition across different devices within the same family, you can only reuse the Synthesized snapshot, and you must ensure that any Fitter constraints (such as Logic Lock regions) from the Developer project do not conflict with constraints in the Consumer project.

Figure 13. Root Partition Reuse Flow



1.5.2.1. Step 1: Developer: Create a Reserved Core Partition

To export and reuse the root partition, the Developer creates a reserved core partition for later core logic development in the Consumer project. The Compiler preserves post-fit results for the partition and reuses the post-fit netlist, if the netlist is available from previous compilation, and you make no partition changes requiring re-synthesis. Otherwise, the Compiler reuses the post-synthesis netlist if available, or resynthesizes the partition from source files.

To create a reserved core partition:

1. Adapt the steps in Step 1: Developer: Create a Design Partition on page 16 to create a reserved core partition.
2. When defining the design partition, select **Reserved Core** for the partition **Type**. Ensure that all other partition options are set to the default values.

1.5.2.2. Step 2: Developer: Define a Logic Lock Region

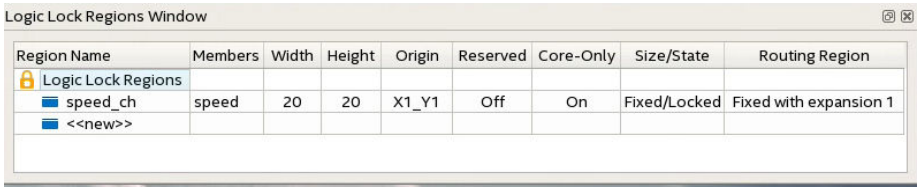
To reserve core resources in a Consumer project for the reserved core partition, the Developer defines a fixed size and location, core-only, reserved Logic Lock region with a defined routing region. The Consumer uses this same region in their project for core development. This region can contain only core logic. Ensure that the reserved placement region is large enough to contain all core logic that the Consumer plans to develop. For projects with multiple core partitions, constrain each partition in a non-overlapping Logic Lock routing region.

Note: When reusing the root partition across different devices within the same family, you can only reuse the Synthesized snapshot, and you must ensure that any Fitter constraints (such as Logic Lock regions) from the Developer project do not conflict with constraints in the Consumer project.

Follow these steps to define a Logic Lock region for core development in the Developer project:

1. Right-click the design instance in the **Project Navigator** and click **Logic Lock Region ➤ Create New Logic Lock Region**. The region appears in the Logic Lock Regions Window. You can also verify the region in the Chip Planner (**Locate Node ➤ Locate in Chip Planner**).
2. Specify the placement region co-ordinates in the **Origin** column.
3. Enable the **Reserved** and **Core-Only** options.
4. For **Size/State**, select **Fixed/Locked**.
5. Click the **Routing Region** cell. The **Logic Lock Routing Region Settings** dialog box appears.

Figure 14. Logic Lock Regions Window



6. Specify **Fixed with expansion** with **Expansion Length** of **1** for the **Routing Type**. For this flow you can select any value other than **Unconstrained**
7. Click **OK**.
8. Click **File ➤ Save Project**.

1.5.2.3. Step 3: Developer: Compile and Export the Root Partition

After compilation, the Developer exports the root partition at the synthesized or final stage. The Developer supplies any Synopsys* Design Constraints (`.sdc`) file for the partition. The Developer uses the `.sdc` files to drive placement and routing. The Consumer uses `.sdc` files for evaluation of partitions that reuse `.qdb` files, and to drive placement in the Fitter for non-reused or non-preserved partitions.

1. To run all compilation stages through Fitter (Finalize), click **Processing ➤ Start ➤ Start Fitter**.
2. To export the root partition to a `.qdb` file, click **Project ➤ Export Design Partition**. Select the **root_partition** and the **synthesized** or **final** snapshot.
3. To include any entity-bound `.sdc` files in the exported `.qdb`, turn on **Include entity-bound SDC files for the selected partition**. By default, all Intel FPGA IP targeting Intel Stratix 10 devices use entity-bound `.sdc` files.

   The following command corresponds to the root partition export in the GUI:

   ```
   quartus_cdb <project name> -c <revision name> \
        --export_partition "root_partition" --snapshot final \
        --file root_partition.qdb --include_sdc_entity_in_partition
   ```

4. The Developer provides the exported `.qdb` file and `.sdc` files for the reserved core to the Consumer.

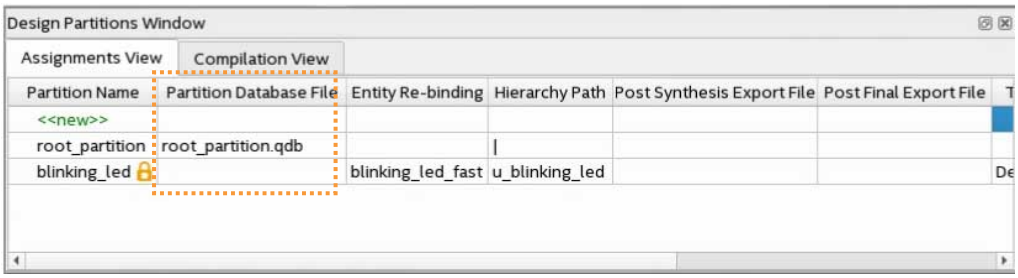
1.5.2.4. Step 4: Consumer: Add the Root Partition and Compile

To reuse the root partition in another project, the Consumer assigns the exported root partition `.qdb` in the Consumer project settings. The root partition `.qdb` includes all Logic Lock region and partition information for the reserved core from the Developer project. There is no need to recreate these constraints in the Consumer project. After assigning the `.qdb`, the Consumer project includes all additional information from the Developer project compilation snapshot. The Consumer then adds RTL for the reserved core partition.

Follow these steps to reuse the root partition in a Consumer project:

1. The Consumer obtains the exported root partition `.qdb` file from the Developer.
2. Open the project that you want to reuse the exported root partition.
3. In the Design Partitions Window, specify a `.qdb` in **Partition Database File** to replace the root_partition logic.

Figure 15. Partition Database File Option in Design Partitions Window



4. The Consumer adds RTL and any `.sdc` constraints for the reserved core partition.
5. To enable the **Fast Preserve** option that simplifies the logic of the preserved partition to only interface logic[2] during compilation, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Incremental Compile ➤ Fast Preserve**.
6. To run all compilation stages, click **Processing ➤ Start Compilation**. The Compiler implements the reused root partition and constraints.

   Note: To use the Entity Re-binding option, you add the .qdb to the project by specifying a .qdb for the **Partition Database File** option in the Design Partitions Window. Refer to Reserved Core Entity Re-Binding on page 22 for more information.

1.5.3. Reserved Core Entity Re-Binding

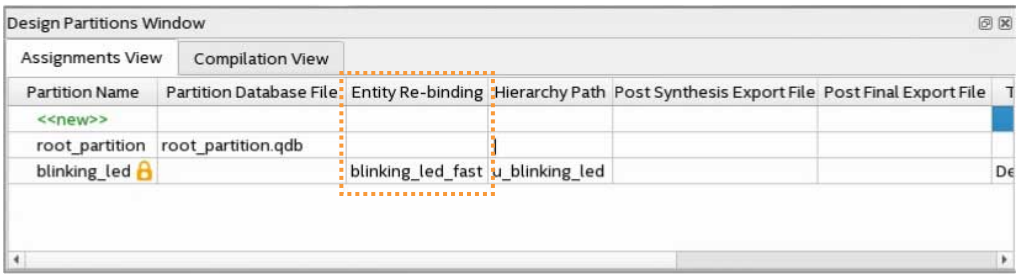
Entity re-binding allows the Consumer in a root partition reuse flow to use an entity name that is different from the Developer's reserved core partition name in the `root_partition.qdb`.

Entity Re-binding Example

The following example illustrates application of **Entity Re-binding**. You specify a value for the **Entity Re-binding** option in the Design Partitions Window to identify the entity bound to a reserved core partition.

[2] Interface logic is logic at the partition boundary that interfaces with the rest of the design.

Figure 16. Entity Rebinding Option in Design Partitions Window



In a root partition reuse example, the Developer's project includes the shell with entity name `blinking_led`, and the `u_blinking_led` instance. The Developer exports the `root_partition.qdb` that includes the root partition, and a reserved core region defined by a Logic Lock placement constraint. The reserved region is associated with the `u_blinking_led` instance.

The Consumer reuses the `root_partition.qdb` in their project. However, the entity that replaces the reserved core has a different name (`blinking_led_fast`) than the reserved core name (`u_blinking_led`) in the Developer project.

If the Consumer simply adds the `u_blinking_led` entity to the project without entity re-binding, an error occurs.

Rather, the Consumer can re-bind the new entity name to the reserved core partition name by setting the **Entity Re-binding** option to `blinking_led_fast`.

1.5.4. Viewing Quartus Database File Information

Although you cannot directly read a `.qdb` file, you can view helpful attributes about the file to quickly identify its contents and suitability for use.

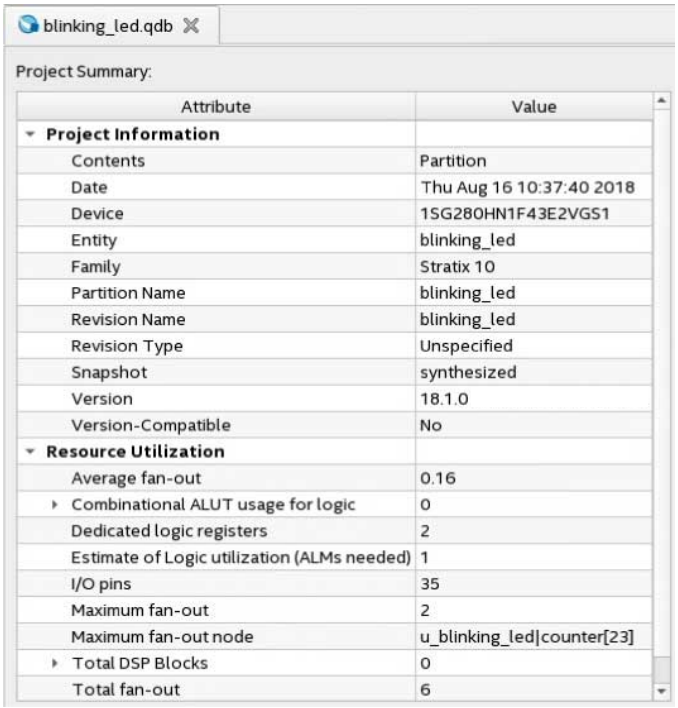
The Intel Quartus Prime software automatically stores metadata about the project of origin when you export a Quartus Database File (`.qdb`). You can then use the Quartus Database File Viewer to display the attributes of any of these `.qdb` files.
Follow these steps to view the attributes of a `.qdb` file:

1. In the Intel Quartus Prime software, click **File ➤ Open**, select **Design Files** for **Files of Type**, and select a `.qdb` file.
2. Click **Open**. The Quartus Database File Viewer displays project and resource utilization attributes of the `.qdb`.

   Alternatively, run the following command-line equivalent:

```
quartus_cdb --extract_metadata --file <archive_name.qdb> \
     --type quartus --dir <extraction_directory> \
     [--overwrite]
```

Figure 17. Quartus Database File Viewer



1.5.4.1. QDB File Attribute Types

The Quartus Database Viewer can display the following attributes of a `.qdb` file:

Table 3. QDB File Attributes

QDB Attribute Types	Attribute	Example
Project Information	Contents	`Partition`
	Date	`Thu Jan 23 10:56:23 2018`
	Device	`10AX016C3U19E2LG`
	Entity (if Partition)	`Counter`
	Family	`Arria 10`
	Partition Name	`root_partition`
	Revision Name	`Top`
	Revision Type	`PR_BASE`
	Snapshot	`synthesized`
	Version	`18.1.0 Pro Edition`
	Version-Compatible	`Yes`
Resource Utilization (exported for partition QDB only)	For synthesized snapshot partition lists data from the **Synthesis Resource Usage Summary** report.	`Average fan-out.16` `Dedicated logic registers:14` `Estimate of Logic utilization:1`

continued...

		I/O pins:35 Maximum fan-out:2 Maximum fan-out node:counter[23] Total DSP Blocks:0 Total fan-out:6 ...
	For the final snapshot partition, lists data from the **Fitter Partition Statistics** report.	Average fan-out:.16 Combinational ALUTs: 16 I/O Registers M20Ks ...

1.6. Incremental Block-Based Compilation Flow

The incremental block-based compilation flow allows you to perform design abstraction by emptying a partition.

1.6.1. Design Abstraction

Empty partitions are useful to account for undefined partitions developed independently or later in the design cycle. The Compiler uses an empty placeholder netlist for the partition, ties the partition output ports to ground, and removes the input ports. The Compiler removes any existing synthesis, placement, and routing information for an empty partition.

If you remove the **Empty** setting from a partition, the Compiler re-implements the partition from the source. Setting a partition to **Empty** can reduce design compilation time because the top-level design netlist does not include the logic for the empty partition. The Compiler does not run full synthesis and fitting algorithms on the empty partition logic.

Note: To avoid resource conflicts when using empty partitions, floorplan any empty partitions that you intend to subsequently replace with a `.qdb`.

Follow these steps to define an empty partition:

1. Create a design partition, as Step 1: Developer: Create a Design Partition on page 16 describes. Set the partition **Type** to **Default**. Any other setting is incompatible with empty partitions.
2. For the **Empty** option, select **Yes**. This setting corresponds to the following assignment in the `.qsf`.

```
set_instance_assignment -name EMPTY ON -to \
     <hierarchal path of partition> -entity <name>
```

1.6.1.1. Empty Partition Clock Source Preservation

Empty partitions preserve clock sources that the Intel Quartus Prime software recognizes.

The Intel Quartus Prime software recognizes and preserves the following as clock sources for a partition:

- Signals from a PLL.
- Feeds from internal clock inputs on flip-flops, memories, HSSIO, I/O registers, or PLLs outside the partition that you empty.

The Intel Quartus Prime software does not recognize the following as clock sources for a partition:

- Nets with sources external to the FPGA that do not feed a clock input inside the FPGA.
- Nets that connect only to combinatorial logic.
- Nets that connect only to an output pin.
- Nets that feed only logic within an empty partition.

1.7. Setting-Up Team-Based Designs

Use the following procedures to setup a project for a team-based design methodology.

1.7.1. Creating a Top-Level Project for a Team-Based Design

In team-based designs that reuse design blocks, all team members ideally work within the same top-level project framework. Using copies of the same project among team members ensures that everyone has the same settings and constraints that their partition requires.

This method helps the team to integrate the partitions into the top-level design. If some Developers do not have access to the top-level project framework, the team lead must provide information about the project and constraints to those Developers.

The following steps describe preparing a top-level project that enables other Developers to provide optimized lower-level design partitions. The top-level project specifies the top-level entity, and then instantiates other design entities that other Developers optimize in a separate Intel Quartus Prime project.

1. Set up the top-level project and add source files. You can represent incomplete sections of the design by adding black box files, as Step 3: Developer: Create a Black Box File on page 18 describes.
2. Define design partitions for any instance that you want to maintain as a separate Intel Quartus Prime project, as Creating Design Partitions on page 13 describes.
3. Define an empty partition for each design partition with unknown or incomplete definition.
4. Create a Logic Lock region constraint for each design block that you plan to integrate as a separate Intel Quartus Prime project. This physical partitioning of the device allows multiple team members to design independently without placement conflicts, as Step 2: Developer: Define a Logic Lock Region on page 20 describes.
5. To run full compilation, click **Processing ➤ Start Compilation**.
6. Use one of the following methods to provide the top-level project information to design Developers:

- If Developers have access to the top-level project framework, the team lead includes all settings and constraints. This framework may include clocks, PLLs, and other periphery interface logic that the Developer requires to develop their partition. If Developers are part of the same design environment, they can check out a copy of the project files they require from the same source control system. This is the best method for sharing a set of project files. Otherwise, the team lead provides a copy of the top-level project (the design and corresponding `.qsf` assignments), so that each Developer creates their partition within the same project framework.
- If Developers do not have access to the top-level project framework, the team lead provides a Tcl script or other specifications to create a separate Intel Quartus Prime project that matches the top-level. The team lead also adds logic around the design block for export, so that the partition is consistent with the key characteristics of the top-level design environment. For example, the team lead can include a top-level PLL in the project, outside of the partition for export, so that Developers can optimize the design with information about the clocks and PLL parameters. This technique provides more accurate timing requirements. Export the partition for the top-level design, without exporting any auxiliary components that you instantiate outside the partition you are exporting.

1.7.1.1. Prepare a Design Partition for Project Integration

Follow these steps to prepare a lower-level design partition for integration with the top-level project:

1. Obtain a copy of the top-level project, or create a new project with the same assignments and constraints as the top-level project. Ensure that your partition uses only the resources that the team lead allocates.
2. For each design partition that is incomplete in the top-level project, set the **Empty** option to **Yes** in the Design Partitions Window. This setting creates an empty partition for later development. For the Compiler to elaborate this partition, you must provide at least the port definitions and any parameters or generics passed to the RTL.
3. When the lower-level design partition is complete, follow the procedure in Step 2: Developer: Compile and Export the Core Partition on page 16. The project lead can now reuse the partition in the top-level project.

1.8. Bottom-Up Design Considerations

Consider the following when using bottom-up design methodology:

Table 4. Bottom-Up Design Recommendations and Limitations

Recommendation/ Limitation	Description
Recommendation 1	Define Logic Lock constraints that are **Reserved**, **Core-Only**, **Fixed/Locked**, with a specified **Routing Region**. While exporting partitions from a different project with a different top-level, generate the partitions with non-overlapping Logic Lock routing regions by setting the routing region to **Fixed with expansion** of 0.
Limitation 1	If you compile two partitions, in two different projects, with `top_level_1.sv` and `top_level_2.sv`, and reuse the partitions in a third project with `top_level_3.sv`, the Compiler cannot support two partitions with overlapping row clock regions. Apply Logic Lock region constraints in the Developer project to avoid two partitions occupying the same row clock region in the Consumer project. For example: 1. From the Consumer project, determine the approximate placement of the two partitions. Choose the Logic Lock constraints for the two partitions such that there is no overlap of the row clock region. 2. In the Developer project with `top_level_1.sv`, apply Logic Lock region constraints that the Consumer identifies for the first partition, followed by compilation and export of the partition at final snapshot. 3. In the Developer project with `top_level_2.sv`, apply Logic Lock region constraints that the Consumer identifies for the second partition, followed by compilation and export of the partition at final snapshot. 4. When reusing the exported partitions in the consumer project with `top_level_3.sv`, the partitions maintain the placement defined in the Developer projects using non-overlapping Logic Lock constraints.
Limitation 2	System-level design errors may not become apparent until late in the design cycle, which can require additional design iterations to resolve.

Related Information

Bottom-Up Design Methodology Overview on page 8

1.9. Debugging Block-Based Designs with the Signal Tap Logic Analyzer

The Intel Quartus Prime Pro Edition software supports debugging of block-based designs with the Signal Tap logic analyzer.

Signal Tap debugging of block-based designs requires specific preparation. For step-by-step details on debugging block-based designs with Signal Tap, refer to "Debugging Block-Based Designs with the Signal Tap Logic Analyzer" in *Intel Quartus Prime Pro Edition User Guide: Debug Tools*.

Related Information

"Debugging Block-Based Designs with the Signal Tap Logic Analyzer," Intel Quartus Prime Pro Edition User Guide: Debug Tools

1.10. Block-Based Design Flows Revision History

Document Version	Intel Quartus Prime Version	Changes
2023.11.07	23.3	• Added Top FAQs navigation to the document cover. • Removed references to design block preservation as new DNI infrastructure does not support the "preserve" assignment.
2019.12.16	19.4.0	• Updated for cross-device snapshot reuse support and limits throughout.
2019.11.11	19.2.0	• Described Fast Preserve option in "Block-Based Design Terminology" and "Step 4: Add the Root Partition and Compile."
2019.07.15	19.2.0	• Changed default file export location from output_files to project directory. • Updated description of partition type GUI. • Updated Support for "Combined Incremental Block-Based Compilation and Design Block Reuse" table for latest supported combinations. • Added note about merging partitions to "Creating Design Partitions."
2018.10.01	18.1.0	• Removed reference to Placed snapshot from "Step 3: Compile and Export the Root Partition." Only Synthesized and Final snapshots are supported for design block reuse.
2018.09.24	18.1.0	• Reorganized order of topics in chapter. • Added the following new topics: — Viewing Quartus Database File Information — Reserved Core Entity Re-Binding — Incremental Block-Based Compilation Examples — Design Methodologies — Top-Down Design Methodology Overview — Bottom-Up Design Methodology Overview — Bottom-Up Design Recommendations and Limitations — Team-Based Design Methodology Overview — Incremental Timing Closure — Incremental Timing Closure Recommendations and Limitations — Design Abstraction • Replaced references to "periphery reuse core" with "reserved core" to reflect latest GUI. • Added description of | as root partition hierarchy path in Design Partitions Window. • Replaced details in "Debugging Block-Based Designs with the Signal Tap Logic Analyzer" section with link to *AN 847: Signal Tap Tutorial with Design Block Reuse for Intel Arria 10 FPGA Development Board*. • Minor wording and graphic updates throughout. • Removed references to unsupported Planned snapshot.
2018.05.07	18.0.0	• First release of chapter as part of stand-alone *Block-Based Design User Guide*. • Added footnote and links to known issues. • Updated all design flow steps to match current GUI. • Updated description of Include entity-bound SDC files option. • Updated statement defining parent to child partition attribute inheritance. • Added *Design Partition Settings* topic. • Added *Block-Based Design Terminology* topic. • Added *Preservation and Reuse with Compiler Snapshots* topic. • Added *Empty Partition Clock Source Preservation* topic.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Added *Design Partitions Properties* table. • Added *Combining Incremental Block-Based Compilation and Design Block Reuse* topic. • Updated *Debugging Block-Based Designs* topic and linked to new Application Note.
2017.11.06	17.1.0	• Reorganization of introduction and Incremental Block-Based Compilation. • Added *Design Partitioning* section. • Added *Debugging Block-Based Designs* section. • Added *Use Empty Partitions to Reduce Compilation Time* topic. • Removed requirement to add `.psmf`, `.msf`, and `.sof` to Consumer project. • Added Intel Stratix 10 support, including information about bundling of `.sdc` with exported partitions for Intel Stratix 10 designs. • Documented changes to Design Partitions window, Export Design Partition dialog box, and Logic Lock Regions window. • Added reference to new Design Partition Planner. • Updated references to corresponding `.qsf` assignments. • Changed references from periphery reuse to root partition reuse. • Rebranded for latest Intel standards.
2017.05.08	17.0.0	• First public release.

1.11. Intel Quartus Prime Pro Edition User Guide: Block-Based Design Document Archive

For the latest and previous versions of this user guide, refer to Intel Quartus Prime Pro Edition User Guide: Block-Based Design. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Intel Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Intel Quartus Prime Pro Edition FPGA design flow.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Intel Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Intel Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Intel Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Intel Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Intel Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Partial Reconfiguration

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

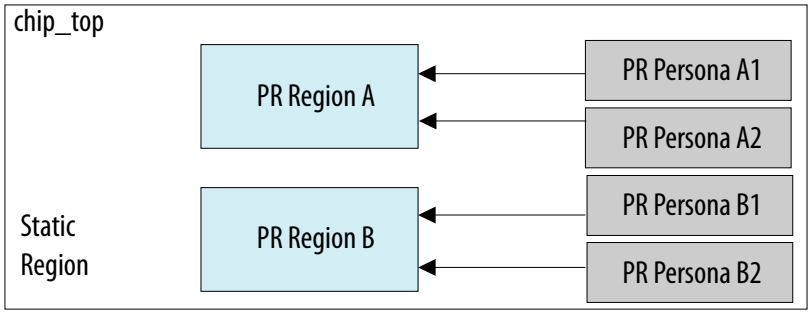
Contents

1. Creating a Partial Reconfiguration Design

Partial reconfiguration (PR) allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. You can define multiple personas for a particular region in your design, without impacting operation in areas outside this region. This methodology is effective in systems with multiple functions that time-share the same FPGA device resources. PR enables the implementation of more complex FPGA systems.

The Quartus® Prime Pro Edition software supports the PR feature for the Stratix® 10, Agilex™ 7, Agilex 5, Arria® 10, and Cyclone® 10 GX device families.

Figure 1. Partial Reconfiguration Design



PR provides the following advancements over a flat design:

- Allows run-time design reconfiguration
- Increases scalability of the design through time-multiplexing
- Lowers cost and power consumption through efficient use of board space
- Supports dynamic time-multiplexing functions in the design
- Improves initial programming time through smaller bitstreams
- Reduces system down-time through line upgrades
- Enables easy system update by allowing remote hardware change
- A simplified compilation flow for partial reconfiguration

Hierarchical Partial Reconfiguration

Quartus Prime Pro Edition software also supports hierarchical partial reconfiguration (HPR), with multiple parent and child design partitions, or multiple levels of partitions in a design. In HPR designs, a static region instantiates a parent PR region, and a parent PR region instantiates a child PR region. The same PR region reprogramming is possible for the child and parent partitions. Refer to Hierarchical Partial Reconfiguration on page 49.

Static Update Partial Reconfiguration

Static update partial reconfiguration (SUPR) allows you to define and modify a specialized static region, without requiring recompilation of all personas. This technique is useful for a portion of a design that you may possibly want to change for risk mitigation, but that never requires runtime reconfiguration. In PR without a SUPR partition, you must recompile all personas for any change to the static region. Refer to the Partial Reconfiguration Tutorials for detailed SUPR instructions.

Partial Reconfiguration Design Simulation

The Quartus Prime Pro Edition software supports simulation of PR persona transitions through the use of simulation multiplexers. You use the simulation multiplexers to change which persona drives logic inside the PR region during simulation. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition. Refer to Partial Reconfiguration Design Simulation on page 52 for details.

Related Information

Partial Reconfiguration Tutorials

1.1. What's New In This Version

- Referenced support for Agilex 5 devices and applied initial Altera rebranding.
- For change details, refer to the Creating a Partial Reconfiguration Design Revision History on page 70 and Partial Reconfiguration Solutions IP User Guide Revision History on page 146.

1.2. Partial Reconfiguration Terminology

This document refers to the following terms to explain partial reconfiguration:

Table 1. Partial Reconfiguration Terminology

Term	Description
Floorplan	The layout of physical resources on the device. Creating a design floorplan, or floorplanning, is the process of mapping the logical design hierarchy to physical regions in the device. PR requires floorplanning.
Hierarchical Partial Reconfiguration	Partial reconfiguration that includes multiple parent and child design partitions, or nesting of partitions in the same design.
PR control block	A dedicated block in Arria 10 and Cyclone 10 GX FPGAs. The PR control block processes the PR requests, handshake protocols, and verifies the cyclic redundancy check (CRC).
PR host	The system for coordinating PR. The PR host communicates with the PR control block (Arria 10 and Cyclone 10 GX designs) or Secure Device Manager (Stratix 10, Agilex 7, and Agilex 5 designs). Implement the PR host within the FPGA (internal PR host) or in a chip or microprocessor.
PR partition	Design partition that you designate as **Reconfigurable**. A PR project can contain one or more PR partitions.
PR Solutions Intel® FPGA IP	Suite of Intel FPGA IP that simplify implementation of PR handshaking and freeze logic, as Partial Reconfiguration Solutions IP User Guide on page 74 describes.

continued...

Term	Description
PR region	A physical portion of an FPGA device that you designate for partial reconfiguration. You define a PR region in the base configuration design. A device can contain more than one PR region. A PR region must be core-only, containing only core resources like LABs, RAM blocks, and DSP blocks. The PR region bitstream configures this region.
PR persona	A specific PR partition implementation in a PR region. A PR region can contain multiple personas. Static regions contain only one persona.
Revision	A collection of settings and constraints for one version of your project. An Quartus Prime Settings File (`.qsf`) preserves each revision of your project. Your Quartus Prime project can contain several revisions. Revisions allow you to organize several versions of your design within a single project.
Secure Device Manager (SDM)	A triple-redundant processor-based block in Agilex 7, Agilex 5, and Stratix 10 devices that performs authentication, decryption, and decompression on the configuration data the block receives, before sending the data over to the configurable nodes through the configuration network.
Snapshot	The output of a Compiler stage. You can export the synthesis or final compilation results snapshot.
Static region	All areas not occupied by PR regions in your project. You associate the static region with the top-level partition of the design. The static region contains both the core and periphery locations of the device. The static region bitstream configures this region.
Static update partial reconfiguration	A static region that you can change, without requiring the recompilation of all personas. This technique is useful for a portion of a design that you may *possibly* want to change for risk mitigation, but that never requires runtime reconfiguration.

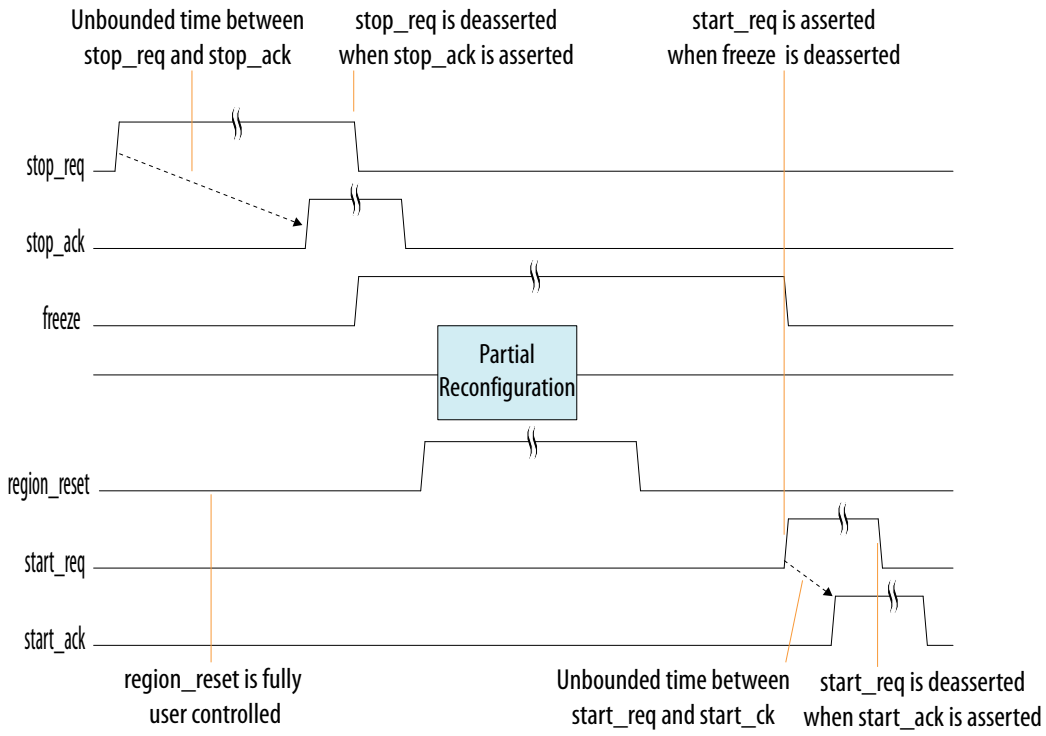
1.3. Partial Reconfiguration Process Sequence

Your partial reconfiguration design must initiate the PR operation and deliver the configuration file to the PR control block (Arria 10 and Cyclone 10 GX designs) or SDM (Agilex 7, Agilex 5, and Stratix 10 designs). Before partial reconfiguration, you must ensure that the FPGA device is in user mode, and in a functional state. The following steps describe the partial reconfiguration sequence:

1. Send the `stop_req` signal to the PR region from the sequential PR control logic to prepare for the PR operation. Upon receiving this signal, the PR regions complete any pending transactions and stop accepting new transactions.
2. Wait for the `stop_ack` signal to indicate that the PR region is ready for partial reconfiguration.
3. Use PR control logic to freeze all necessary outputs of the PR regions. Additionally, drive the clock enable for any initialized RAMs to a disabled state.
4. Send the PR bitstream to the PR control block (Arria 10 and Cyclone 10 GX designs) or SDM (Agilex 7, Agilex 5, and Stratix 10 designs) to initiate the PR process for the PR region. When using any of the Partial Reconfiguration Controller Intel FPGA IP, the Avalon® memory-mapped or Avalon streaming interface on the IP core provides this functionality. When directly instantiating the PR control block for Arria 10 designs, refer to PR Control Block Signal Timing Diagrams on page 103
5. On successful completion of the PR operation, reset the PR region.

6. Signal the start of PR operation by asserting the `start_req` signal, and deasserting the `freeze` signal.
7. Wait for the `start_ack` signal to indicate that the PR region is ready for operation.
8. Resume operation of the FPGA with the newly reconfigured PR region.

Figure 2. PR Process Sequence Timing Diagram



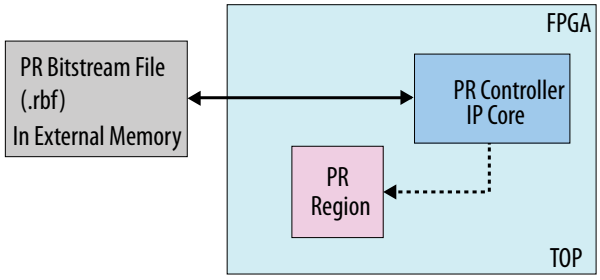
1.4. Internal Host Partial Reconfiguration

With internal host control, an internal controller, a Nios® II processor, or an interface such as PCI Express* (PCIe*) or Ethernet, communicates directly with the Arria 10 or Cyclone 10 GX PR control block, or with the SDM in Agilex 7, Agilex 5, and Stratix 10 devices.

To transfer the PR bitstream into the PR control block or SDM, you use the Avalon memory-mapped interface on the Partial Reconfiguration Controller IP core. When the device enters user mode, you initiate partial reconfiguration through the FPGA core fabric using the PR internal host.

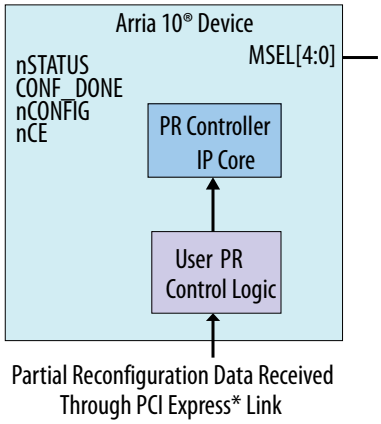
Note: If you create your own control logic for the PR host, the logic must meet the PR interface requirements.

Figure 3. Internal Host PR



When performing partial reconfiguration with an internal host, use the dedicated PR pins (PR_REQUEST, PR_READY, PR_DONE, and PR_ERROR) as regular I/Os. Implement your static region logic to retrieve the PR programming bitstreams from an external memory, for processing by the internal host.

Figure 4. Arria 10 FPGA System Using an Internal PR Host Example



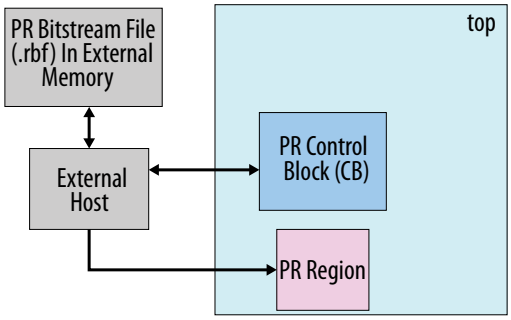
For example, send the programming bitstreams for partial reconfiguration through the PCI Express link. Then, you process the bitstreams with your PR control logic and send the bitstreams to the PR IP core for programming. nCONFIG moves the device out of the user mode into the device configuration mode.(1)

1.5. External Host Partial Reconfiguration

In external host control, an external FPGA or CPU controls the PR configuration using external dedicated PR pins on the target device. When using an external host, you must implement the control logic for transmission of the bitstream to the hard FPGA programming pins.

(1) nCONFIG can lock the device and force a power-cycle. PR programming may corrupt the static logic, due to improper use, causing disconnection of the core clock input to configuration block and unresponsive configuration. You must reset the PR IP before toggling nCONFIG.

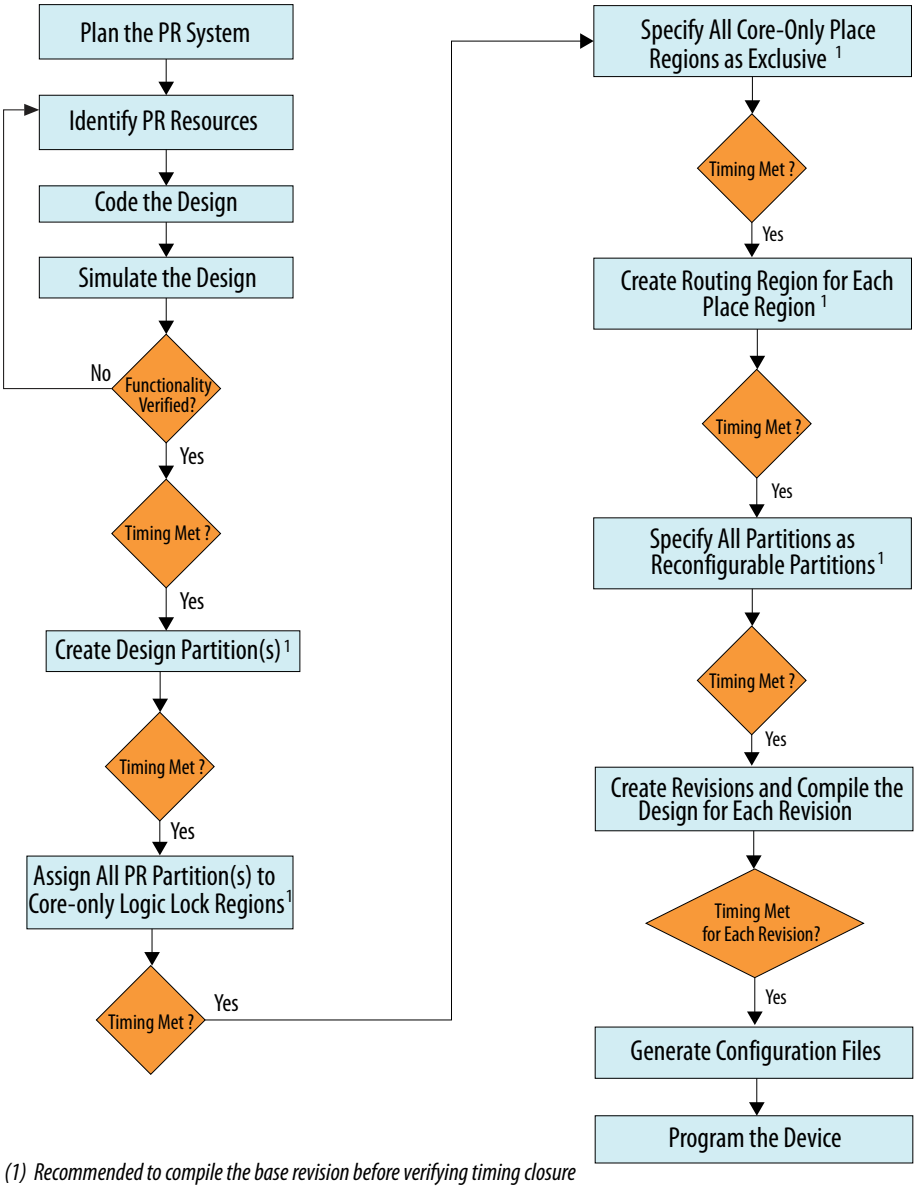
Figure 5. PR System Using an External Host (Arria 10 Example)



1.6. Partial Reconfiguration Design Flow

The PR design flow requires initial planning. This planning involves setting up one or more design partitions, and then determining the placement assignments in the floorplan. Well-planned PR partitions improve design area utilization and performance. The Quartus Prime software also allows you to create nested PR regions as part of an HPR flow.

Figure 6. Partial Reconfiguration Design Flow



(1) Recommended to compile the base revision before verifying timing closure

The PR design flow uses the project revisions feature in the Quartus Prime software. Your initial design is the base revision, where you define the static region boundaries and reconfigurable regions on the FPGA. From the base revision, you create multiple

revisions. These revisions contain the different implementations for the PR regions. However, all PR implementation revisions use the same top-level placement and routing results from the base revision.

The PR design flow includes the following steps:

1.6.1. Step 1: Identify Partial Reconfiguration Resources

When designing for partial reconfiguration, you must first determine the logical hierarchy boundaries that you can define as reconfigurable partitions. Reconfigurable partitions must contain only core resources, such as LABs, embedded memory blocks (M20Ks and MLABs), and DSP blocks in the FPGA.

All periphery resources, such as transceivers, external memory interfaces, GPIOs, I/O receivers, and the hard processor system (HPS), must be in the static region. Partial reconfiguration of global network buffers for clocks and resets is not possible.

Table 2. Supported Reconfiguration Methods

Hardware Resource Block	Reconfiguration Method
Logic Block	Partial reconfiguration
Digital Signal Processing	Partial reconfiguration
Memory Block	Partial reconfiguration
Core Routing	Partial reconfiguration
Transceivers/PLL	Dynamic reconfiguration
I/O Blocks	Not supported
Clock Control Blocks	Not supported

After identifying the resources for PR, set up the design hierarchy and source code to support this partitioning. Refer to Partial Reconfiguration Design Considerations on page 33.

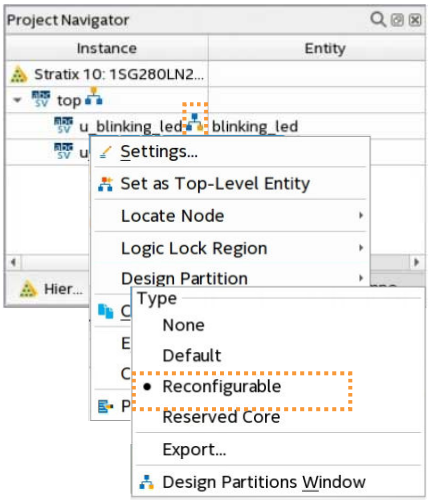
1.6.2. Step 2: Create Design Partitions

Create design partitions for each PR region that you want to partially reconfigure. Create any number of independent partitions or PR regions in your design. Create design partitions for partial reconfiguration from the Project Navigator, or the Design Partitions Window.

A design partition is only a logical partitioning of the design, and does not specify a physical area on the device. You associate a partition with a specific area of the FPGA using Logic Lock Region assignments. To avoid partitions obstructing design optimization, group the logic together within the same partition. If your design includes a hierarchical PR flow with parent and child partitions, you can define multiple parent or child partitions in your design, and multiple levels of PR partitions.

When you create a **Reconfigurable** partition, the Compiler preserves post-synthesis results for the partition and reuses the post-synthesis netlist, if you make no partition changes requiring re-synthesis. Otherwise, the Compiler resynthesizes the partition from source files. The Compiler adds wire LUTs for each interface of a **Reconfigurable** partition, and performs checks for PR compatibility.

Figure 7. Creating a Design Partition



Follow these steps to create design partitions:

1. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
2. In the Project Navigator, right-click an instance in the **Hierarchy** tab, click **Design Partition ➤ Set as Design Partition**. A design partition icon appears next to each partition you create.
3. To view and edit all design partitions in the project, click **Assignments ➤ Design Partitions Window**.
4. Specify **Reconfigurable** as the partition **Type** for each PR partition. The **Reconfigurable** type preserves synthesis results, while allowing refit of the partition in the PR flow.

Figure 8. Design Partitions Window

Design Partitions Window

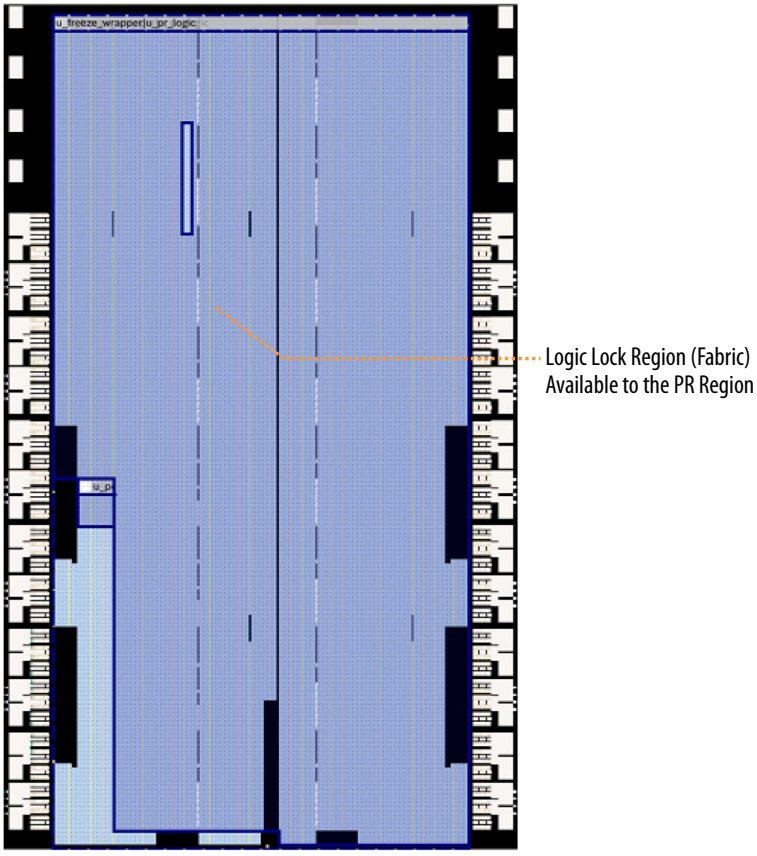
Assignments View | Compilation View

Partition Name	Hierarchy Path	Type	Preservation Level	Empty
<<new>>				
root_partition	|			
pr_partition	u_blinking_led	Reconfigurable	Not Set	No
supr_partition	u_top_counter	Reconfigurable	Not Set	No

1.6.3. Step 3: Floorplan the Design

Use Logic Lock floorplan constraints in your PR design to physically partition the device. Each PR partition in your design must have a corresponding, exclusive physical partition. You create Logic Lock regions to define the physical partition for your PR region. This partitioning ensures that the resources available to the PR region are the same for any persona that you implement.

Figure 9. PR Region Floorplan



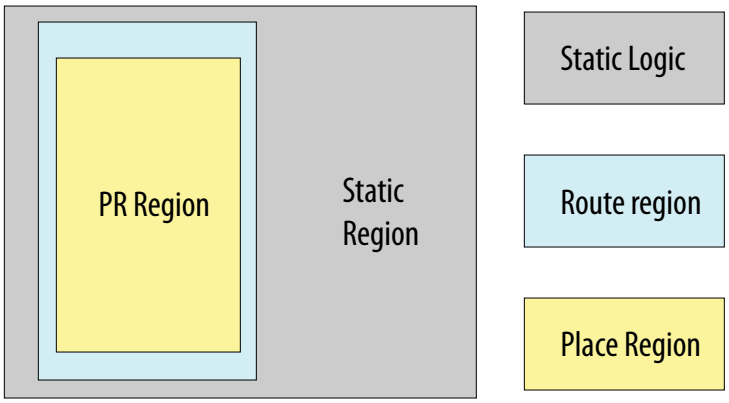
Your PR region must include only core logic, such as LABs, RAMs, ROMs, and DSPs in a PR region. Agilex 7, Agilex 5, and Stratix 10 designs can also include Hyper-Registers in the PR partition. Instantiate all periphery design elements, such as transceivers,

external memory interfaces, and clock networks in the static region of the design. The Logic Lock regions you create can cross periphery locations, such as the I/O columns and the HPS, because the constraint is core-only.

There are two region types:

- **Place regions**—use these regions to constrain logic to a specific area of the device. The Fitter places the logic in the region you specify. The Fitter can also place other logic in the region unless you designate the region as **Reserved**.
- **Route regions**—use these regions to constrain routing to a specific area. The routing region must fully enclose the placement region. Additionally, the routing regions for the PR regions cannot overlap.

Figure 10. Floorplanning your PR Design



Follow these guidelines when floorplanning your PR design:

- Complete the periphery and clock floorplan before core floorplanning. You can use Interface Planner (**Tools ➤ Interface Planner**) to create periphery floorplan assignments for your design.
- Define a routing region that is at least 1 unit larger than the placement region in all directions. In defining this region, avoid any overlapping routing regions between the static and PR regions.
- Do not overlap the routing regions of multiple PR regions.
- Select the PR region row-wise for least bitstream overhead. In Arria 10 and Cyclone 10 GX devices, short, wider regions generate smaller bitstreams than tall, narrower regions. Configuration occurs on sectors for Agilex 7, Agilex 5, and Stratix 10 devices. For the least bitstream overhead, ensure that you align the PR region to sector boundaries. Refer to "Analyzing and Optimizing the Design Floorplan," in *Quartus Prime Pro Edition User Guide: Design Optimization*.
- For Arria 10 and Cyclone 10 GX devices, the height of your PR region affects the reconfiguration time. A PR region larger in the Y direction takes longer to reconfigure. This condition does not apply to Agilex 7 or Stratix 10 devices because they configure according to sectors. The reconfiguration time of Agilex 7, Agilex 5, and Stratix 10 devices depends on the number of sectors the PR region covers. This reconfiguration time can also be affected by other factors, such as interleaving or the presence of other Logic Lock regions.

- To reduce programming files size for Agilex 7, Agilex 5, and Stratix 10 devices, target only the necessary number of sectors for PR. Also, ensure that the routing region of your PR region is 1 block (1 LAB row/column) inset from the edges of the clock sector boundaries.
- Define sub Logic Lock regions within PR regions to improve timing closure.
- If your design includes HPR parent and child partitions, the placement region of the parent region must fully enclose the routing and placement region of its child region. Also, the parent wire LUTs must be in an area outside the child PR region. This requirement is because the child PR region is exclusive to all other logic, which includes the parent and the static region.
- The base revision `.qdb` provides the only effective pin assignments for the implementation revision. Even if you subsequently change the pin assignments to the implementation revisions, those changes do not take effect.

Related Information

- Quartus Prime Pro Edition User Guide: Design Optimization
- Quartus Prime Pro Edition User Guide: Design Constraints

1.6.3.1. Applying Floorplan Constraints Incrementally

PR implementation requires additional constraints that identify the reconfigurable partitions of the design and device. These constraints significantly impact the Compiler's timing closure ability. You can avoid and more easily correct timing closure issues by incrementally implementing each constraint, running the Compiler, then verifying timing closure.

Note: PR designs require a more constrained floorplan, compared to a flat design. The overall density and performance of a PR design may be lower than an equivalent flat design.

The following steps describe incrementally developing the requirements for your PR design:

1. Implement the base revision using the most complex persona for each PR partition. This initial implementation must include the complete design with all periphery constraints, and top-level `.sdc` timing constraints. Do not include any Logic Lock region constraints for the PR regions with this implementation.
2. Create partitions by setting the region **Type** option to **Default** in the Design Partitions Window, for all the PR partitions.
3. Register the boundaries of each partition to ensure adequate timing margin.
4. Verify successful timing closure using the Timing Analyzer.
5. Ensure that all the desired signals are driven on global networks. Disable the **Auto Global Clock** option in the Fitter (**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**), to avoid promoting non-global signals.
6. Create Logic Lock core-only placement regions for each of the partitions.
7. Recompile the base revision with the Logic Lock constraints, and then verify timing closure.

8. Enable the **Reserved** option for each Logic Lock region to ensure the exclusive placement of the PR partitions within the placement regions. Enabling the **Reserved** option avoids placing the static region logic in the placement region of the PR partition.
9. Recompile the base revision with the **Reserved** constraint, and then verify timing closure.
10. In the Design Partitions Window, specify the **Type** for each of the PR partitions as **Reconfigurable**. This assignment ensures that the Compiler adds wire LUTs for each interface of the PR partition, and performs additional compilation checks for partial reconfiguration.
11. Recompile the base revision with the **Reconfigurable** constraint, and then verify timing closure. You can now export the top-level partition for reuse in the PR implementation compilation of the different personas.

1.6.4. Step 4: Add the Partial Reconfiguration Controller Intel FPGA IP

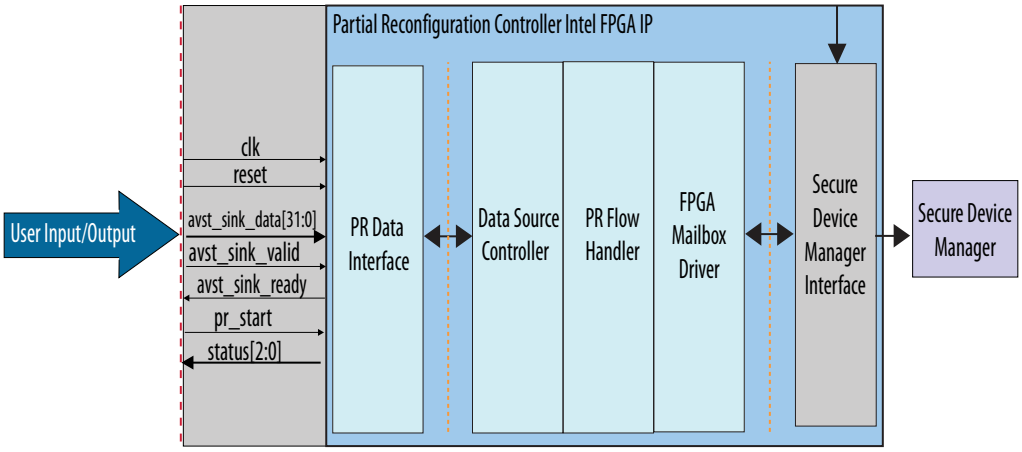
Depending on the target device family, you can add the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP or the Partial Reconfiguration Controller Intel FPGA IP to your design to send the partial reconfiguration bitstream to the PR control block or SDM in an internal host configuration.

1.6.4.1. Adding the Partial Reconfiguration Controller Intel FPGA IP

You can customize and instantiate the Partial Reconfiguration Controller Intel FPGA IP from the IP Catalog (**Tools ➤ IP Catalog**).

The Partial Reconfiguration Controller Intel FPGA IP interfaces with the Secure Device Manager (SDM) to manage the bitstream source. The SDM performs authentication and decompression on the configuration data. You can use this IP core in an Agilex 7, Agilex 5, or Stratix 10 design when performing partial reconfiguration with an internal PR host, Nios II processor, PCI Express, or Ethernet interface.

Figure 11. Partial Reconfiguration Controller (Avalon Streaming Interface)
(2)



(2) Avalon memory-mapped interface variant also available.

The Quartus Prime software supports PR over the core interface using the PR Controller IP core, or PR over the JTAG device pins. PR over JTAG pins does not require instantiation of the Partial Reconfiguration Controller Intel FPGA IP.

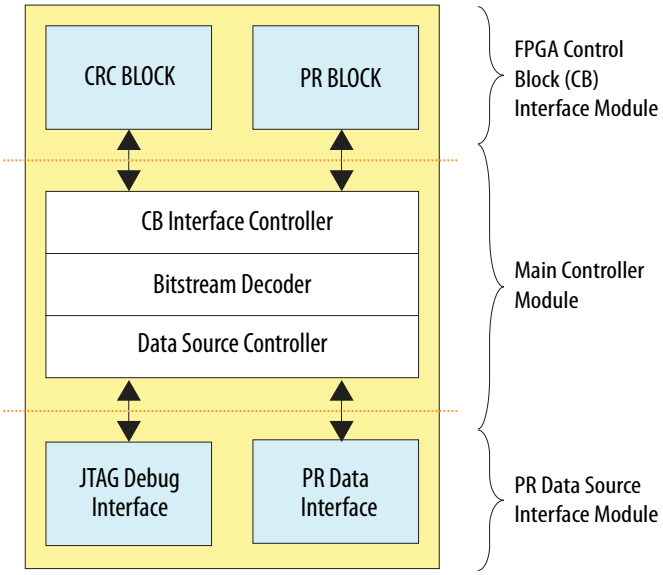
1.6.4.2. Adding the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP

The Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP interfaces with the Arria 10 or Cyclone 10 GX PR control block to manage the bitstream source.

Use this IP core in Arria 10 or Cyclone 10 GX designs when performing partial reconfiguration with an internal PR host, Nios II processor, PCI Express, or Ethernet interface.

During partial reconfiguration, you send a PR bitstream stored outside the FPGA to the PR control block inside the FPGA. This communication enables the control block to update the CRAM bits necessary for reconfiguring the PR region in the FPGA. The PR bitstream contains the instructions (opcodes) and the configuration bits necessary for reconfiguring a specific PR region.

Figure 12. Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP



Instantiate the IP core from the Quartus Prime IP Catalog (**Tools ➤ IP Catalog)** to automatically connect the IP to the Arria 10 or Cyclone 10 GX PR control block.

If you create your own custom logic to perform the function of the IP core, manually instantiate the control block to communicate with the FPGA system.

Related Information

- Partial Reconfiguration Controller Intel Arria® 10/Cyclone® 10 FPGA IP on page 90
- PR Control Block and CRC Block Verilog HDL Manual Instantiation on page 100
- PR Control Block and CRC Block VHDL Manual Instantiation on page 100

1.6.5. Step 5: Define Personas

Your partial reconfiguration design can have multiple PR partitions, each with multiple personas. You define the unique function of each persona in separate Verilog HDL, SystemVerilog HDL, or VHDL design files in the project directory. All the PR personas must use the same set of signals to interact with the static region.

Ensure that the signals interacting with the static region are a super-set of all the signals in all the personas. A PR design requires an identical I/O interface for each persona in the PR region. If all personas for your design do not have identical interfaces, you must also create wrapper logic to interface with the static region.

Note: If using the Quartus Prime Text Editor, disable **Add file to current project** when saving the files. These persona source files should not be part of the Quartus Prime project or compilations.

1.6.6. Step 6: Create Revisions for Personas

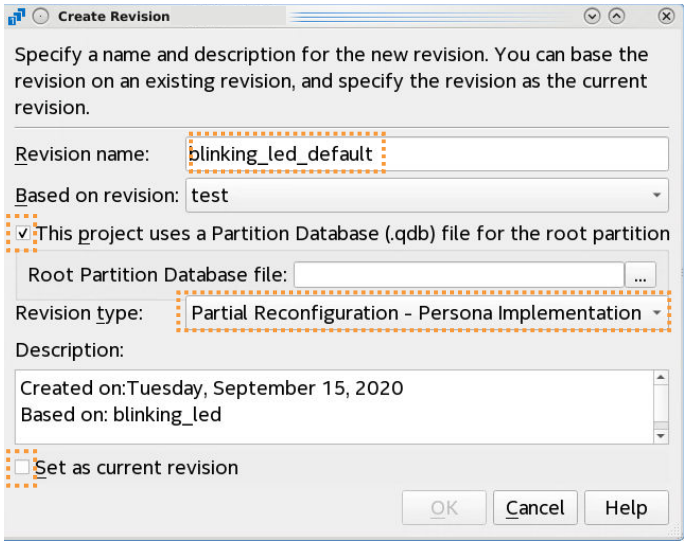
Create a base revision for the design, as well as PR implementation revisions for each of the personas. When you define revisions in the GUI or at the command line, the Quartus Prime software automatically adds these assignments required for PR implementation:

- Entity Rebinding assignment (`ENTITY_REBINDING`)—for each PR partition, the software adds an entity rebinding assignment with a place holder for the entity name. Your design may not require all of the entity rebinding assignments of each PR partition, based on the design and the implementation revision. For example, in HPR designs that use the default persona for the parent partition, you add the `.qdb` file for the PR parent, and then use entity rebinding only for the child.
- QDB File Partition assignment (`QDB_FILE_PARTITION`)—the software adds this assignment for the static region, if you specify a `.qdb` file name.
- Revision Type Assignment (`REVISION_TYPE`)

To create the PR implementation revisions:

1. Click **Project ➤ Revisions**.
2. To create a new revision, double-click **<<new revision>>**.
3. Specify a unique **Revision name**.
4. Select an existing revision for the **Based on revision** option.
5. For the **Revision type**, select **Partial Reconfiguration - Base** for the base revision or **Partial Reconfiguration - Persona Implementation** for an implementation revision.
6. Click **Apply** and **OK**.

Figure 13. Creating Revisions



The following assignments in the respective revision's `.qsf` file correspond to specifying the revision type from the **Settings** dialog box:

Base Revision Assignment:

```
set_global_assignment -name REVISION_TYPE PR_BASE
```

Implementation Revision Assignment:

```
set_global_assignment -name REVISION_TYPE PR_IMPL
```

For each PR partition, the Quartus Prime software also adds the entity rebinding assignment to the `.qsf`:

```
set_instance_assignment -name ENTITY_REBINDING <entity_name> -to
<hierarchical_path>
```

If you base a new implementation revision on an existing `.qdb` file, The Quartus Prime software also adds the `.qdb` file partition assignment, with a place holder for the file name:

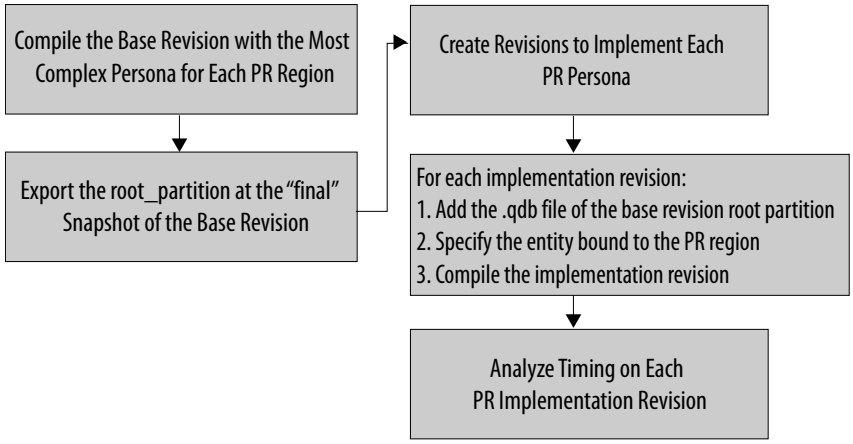
```
set_instance_assignment -name QDB_FILE_PARTITION <QDB file name>
```

As an example, to create a new implementation revision that uses a `.qdb` file from a base revision, use the following command:

```
create_revision impl_new -based_on <base_revision> \
     -new_rev_type impl -root_partition_qdb_file base_static.qdb
```

- `impl_new`—specifies the name of a new implementation revision.
- `-based_on <based_on_revision>` — specifies the PR base revision that the new `impl` revision is based on. Some global assignments from the `based_on` revision are copied over to the `impl` revision. Placeholder entity rebinding assignments are created in the `impl` revision for each PR partition in the base.
- `-new_rev_type <rev_type>`— only useful rev-type is `impl`.
- `root_partition_qdb_file <qdb_file>`—creates a `QDB_FILE_PARTITION` assignment in `impl` revision with the specified `.qdb` file.

Figure 14. Partial Reconfiguration Compilation Flow



1.6.7. Step 7: Compile the Base Revision and Export the Static Region

After defining and floorplanning PR partitions and revisions, you compile the base revision and export the static region. You can export individual design partitions manually, or you can export one or more partitions automatically each time you run the Compiler.

Follow these steps to compile and export the base and static region:

1. To specify the current revision, click **Project ➤ Revisions**, and then set the base revision as current, or select the base revision from the main toolbar drop-down list.
2. For Arria 10 and Cyclone 10 GX designs, you can optionally add the following assignments to the `.qsf` to automatically generate the required PR bitstreams following compilation. This step is not required for Agilex 7, Agilex 5, or Stratix 10 designs.

```
set_global_assignment -name GENERATE_PR_RBF_FILE ON
set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
```

3. To compile the base revision, click **Processing ➤ Start Compilation**.
4. To export the static region, click **Project ➤ Export Design Partition** and specify options for the partition export:

Figure 15. Export Design Partition

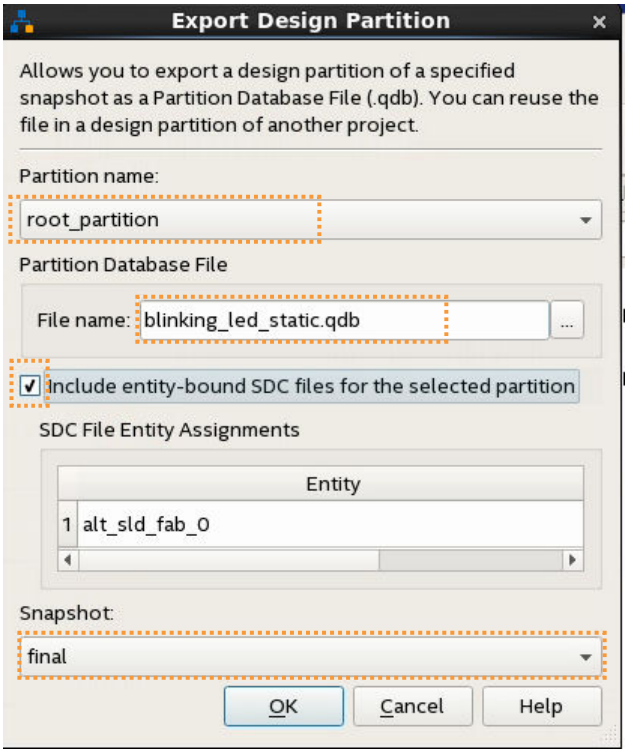
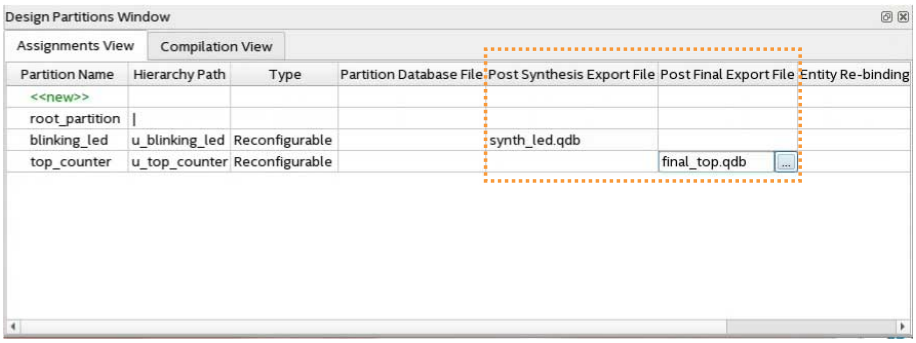


Table 3. Design Partition Options

Option	Setting
Partition name	Select **root_partition**.
Partition database file	Specify a descriptive file name.
Include entity-bound SDC files	Enable to include entity bound `.sdc` files with the partition export. *Note:* You must enable this option when exporting the base revision (root partition), so that the implementation compiles inherit the timing constraints defined in entity-bound `.sdc` files.
Snapshot	Select **final** snapshot.

5. Alternatively, follow these steps to automatically export one or more design partitions after each compilation. You can automatically export any design partition that does not have a preserved parent partition, including the root_partition.
   a. To open the Design Partitions Window, click **Assignments ➤ Design Partitions Window**.
   b. To automatically export a partition with final snapshot results any time you run the Fitter, specify a `.qdb` file name for the **Post Final Export File** option for that partition.

Figure 16. Specifying Export File in Design Partitions Window



QSF File Equivalent:

```
set_instance_assignment -name \
     EXPORT_PARTITION_SNAPSHOT_<FINAL|SYNTHESIZED> \
     <hierarchy_path> -to <file_name>.qdb
```

1.6.7.1. Understanding PR Logic Utilization Reports

After running the Fitter compilation stage, you can view reports about the logic utilization of your PR design in the Compilation Report. The Partial Reconfiguration and Periphery Reuse Statistics report provides data about the boundary ports, ALMs, and other resources that the PR region requires.

This report may show an increase in the implementation revision ALM logic utilization compared with the base revision utilization. Specifically, the value of '`[C] estimate of the ALMs unavailable`' can be significantly higher in the implementation revision compared to the base revision, as the following figures illustrate.

Figure 17. Partial Reconfiguration and Periphery Reuse Statistics Report (Base Revision)

Partial Reconfiguration and Periphery Reuse Statistics

<<Filter>>

	Statistic	root_partition	pr_partition
1	Hierarchical Path	|	u_blinking_led
2			
3	▾ Boundary Ports	9	42
1	Input Ports	4	39
2	Output Ports	5	3
4			
5	Include Partial Reconfigurable Sub-blocks		
6	▾ ALMs needed [=A-B+C]	100.0 / 933120 (< 1 %)	20.5 / 3200 (< 1 %)
1	[A] ALMs used in final placement	117.0 / 933120 (< 1 %)	20.5 / 3200 (< 1 %)
2	[B] Estimate of ALMs recoverable by dense packing	17.0 / 933120 (< 1 %)	0.0 / 3200 (0 %)
3	[C] Estimate of ALMs unavailable	0.0 / 933120 (0 %)	0.0 / 3200 (0 %)

Figure 18. Partial Reconfiguration and Periphery Reuse Statistics Report (Implementation Revision)

Partial Reconfiguration and Periphery Reuse Statistics

<<Filter>>

	Statistic	root_partition	pr_partition
1	Hierarchical Path	|	u_blinking_led
2			
3	▾ Boundary Ports	9	42
1	Input Ports	4	39
2	Output Ports	5	3
4			
5	Include Partial Reconfigurable Sub-blocks		
6	▾ ALMs needed [=A-B+C]	519.3 / 933120 (< 1 %)	344.3 / 3200 (10 %)
1	[A] ALMs used in final placement	498.5 / 933120 (< 1 %)	402.0 / 3200 (12 %)
2	[B] Estimate of ALMs recoverable by dense packing	71.9 / 933120 (< 1 %)	58.9 / 3200 (1 %)
3	[C] Estimate of ALMs unavailable	92.7 / 933120 (< 1 %)	1.2 / 3200 (< 1 %)

This increase in ALM usage appears because the base revision compilation report reflects logic utilization that is based only on the base revision RTL file. However, for the implementation revision, the static region is imported from the base revision `.qdb` file that also includes all of the logic (used and unused) of the static region. This additional base revision logic causes the increase in `[C]`. In this case, `[C]` includes every ALM of the static region `.qdb` file. In contrast, `[A]` is the actual number of used ALMs in the design after placement. For the Total Logic Utilization of PR designs, you must review the `[A]ALMs used in final placement`.

1.6.8. Step 8: Setup PR Implementation Revisions

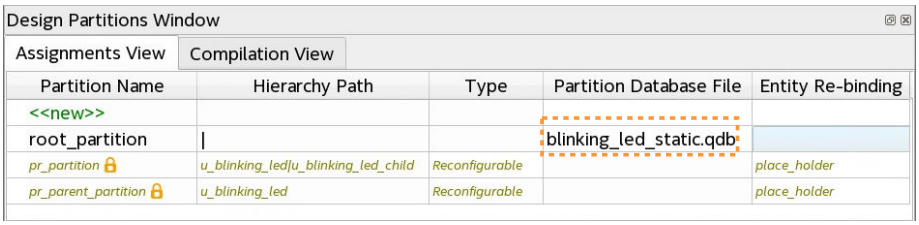
You must prepare the PR implementation revisions before you can generate the PR bitstream for device programming. This setup includes adding the static region `.qdb` file as the source file for each implementation revision. In addition, you must specify the corresponding entity of the PR region.

Note: The base revision `.qdb` provides the only effective pin assignments for the implementation revision. Even if you subsequently change the pin assignments in the implementation revision `.qsf`, those changes do not take effect.

Follow these steps to setup the PR implementation revisions:

1. Set an implementation revision as the **Current Revision**.
2. To specify the `.qdb` file as the source for `root_partition`, click **Assignments ➤ Design Partitions Window**. Double-click the **Partition Database File** cell and specify the appropriate `.qdb` file.
3. For each PR implementation revision, specify the name of the entity that you want to partially reconfigure in the **Entity Re-binding cell**. This entity name comes from the design file for the persona you want to implement in this implementation revision.

Figure 19. Design Partitions Window



4. To compile the design, click **Processing ➤ Start Compilation**.
5. Repeat steps 1 through 4 to setup and compile each implementation revision. Alternatively, use a simple Tcl script to compile all implementation revisions:

```
set_current_revision <implementation1 revision name>
execute_flow -compile
set_current_revision <implementation2 revision name>
execute_flow -compile
.
.
.
```

Note: When you generate a static `.qdb` for import into a PR implementation compile, make sure to preserve the entity-bound `.sdc` files for the static partition. Also, for the implementation revision to properly process the `.sdc` files, the order of assignments in the implementation file `.qsf` is very important. Verify the order of the `.sdc` files in the implementation revision. The implementation revision includes the entity-bound `.sdc` constraints pulled in by the static region `.qdb`. The implementation revision also includes the `.sdc` files for the implementation revision. If you require the `.sdc` files pulled in by the static region `.qdb` before the implementation revision `.sdc` files, ensure that the `QDB_FILE_PARTITION` assignment appears before any other `.sdc` file assignment.

1.6.9. Step 9: Program the FPGA Device

The Quartus Prime Assembler generates the PR bitstreams for your design personas. For Arria 10 and Cyclone 10 GX designs, you send the bitstreams to the PR control block. For Agilex 7, Agilex 5, and Stratix 10 designs, you send the PR bitstreams to the SDM. You must compile the PR project, including the base revision, and at least one implementation revision, before generating the PR bitstreams.

For Agilex 7, Agilex 5, and Stratix 10 designs, the Assembler generates a configuration `.rbf` automatically at the end of compilation. For Arria 10 and Cyclone 10 GX designs, you can add the GENERATE_PR_RBF_FILE assignment to the `.qsf` or use the **Convert Programming Files** dialog box to convert the Partial-Masked SRAM Object Files (`.pmsf`) to an `.rbf` file, as Generating PR Bitstream Files on page 28 describes.

Figure 20. Programming File Generation

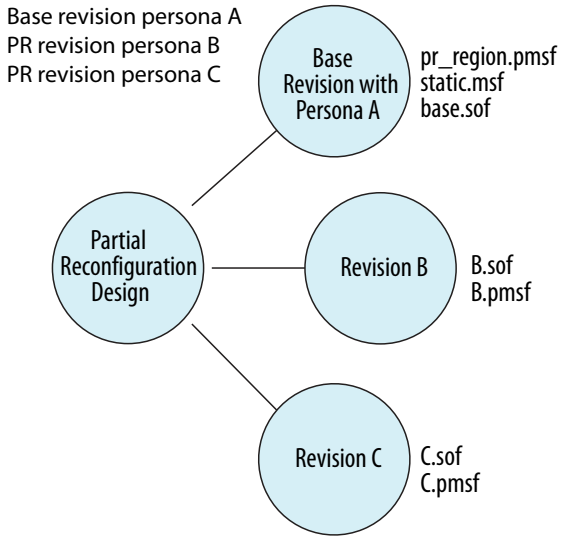


Table 4. PR Programming Files

Programming File	Description
`<rev>.<pr_region>.pmsf`	Contains the partial-mask bits for the PR region. The `.pmsf` file contains all the information for creating PR bitstreams. *Note:* The default file name corresponds to the partition name.
`<rev>.<static_region>.msf`	Contains the mask bits for the static region.
`<rev>.sof`	Contains configuration information for the entire device.

Related Information

- Partial Reconfiguration Security (Stratix 10 Designs) on page 59
- Agilex 7 Configuration User Guide
- Stratix 10 Configuration User Guide
- Arria 10 Configuration User Guide
- Cyclone 10 GX Core Fabric and General Purpose I/Os Handbook

1.6.9.1. Generating PR Bitstream Files

For Agilex 7, Agilex 5, and Stratix 10 designs, the Assembler generates a configuration `.rbf` automatically at the end of compilation. For Arria 10 and Cyclone 10 GX designs, use any of the following methods to process the PR bitstreams and generate the Raw Binary File (`.rbf`) file for reconfiguration.

Note: The Assembler generates a configuration `.rbf` automatically at the end of compilation for Agilex 7, Agilex 5, and Stratix 10 designs. You do not need to separately generate these files when targeting these devices.

Generating PR Bitstreams During Compilation (Arria 10 and Cyclone 10 GX Designs)

Follow these steps to generate the `.rbf` file during compilation for Arria 10 and Cyclone 10 GX designs:

1. Add the following assignments to the revision `.qsf` to automatically generate the required PR bitstreams following compilation:

```
set_global_assignment -name GENERATE_PR_RBF_FILE ON
set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
```

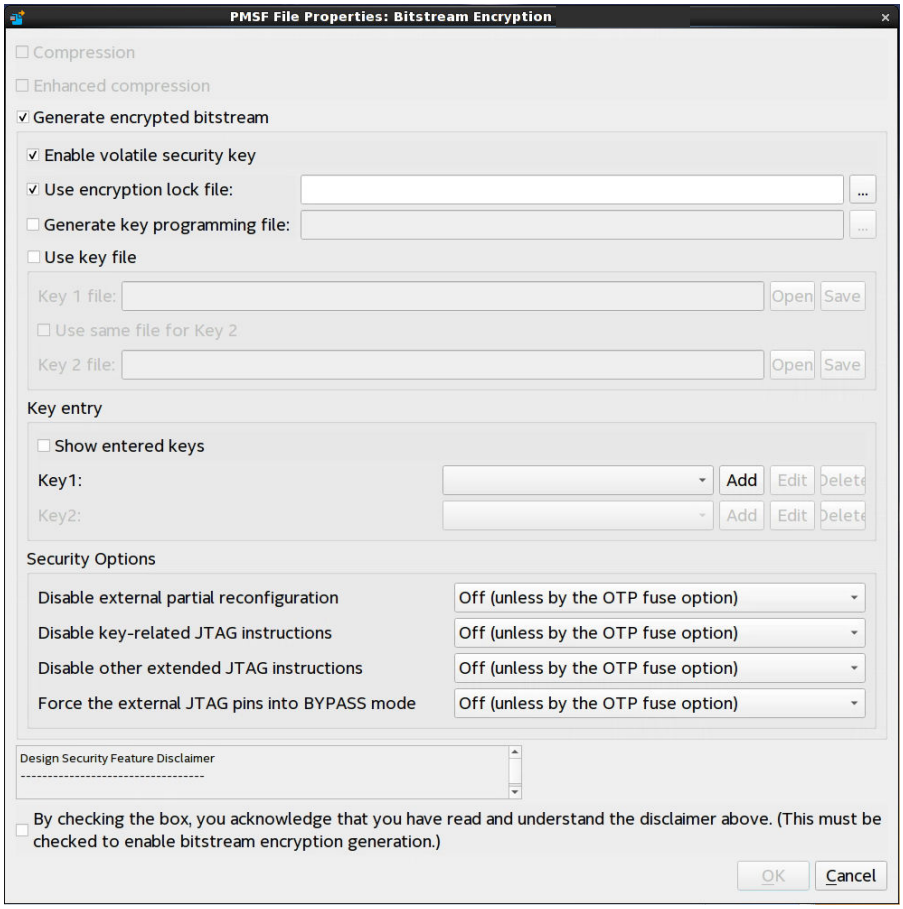
2. To compile the revision and generate the `.rbf`, click **Processing ➤ Start Compilation**.

Generating PR Bitstreams with Convert Programming Files Dialog Box (Arria 10 and Cyclone 10 GX Designs)

Follow these steps to generate the `.rbf` with the **Convert Programming Files** dialog box:

1. Click **File ➤ Convert Programming Files**. The **Convert Programming Files** dialog box appears.
2. Specify the output file name and **Programming file type** as **Raw Binary File for Partial Reconfiguration (.rbf)**.
3. To add the input `.pmsf` file to convert, click **Add File**.
4. Select the newly added `.pmsf` file, and click **Properties**.
5. Enable or disable any of the following options and click **OK**:
   - **Compression**—enables compression on PR bitstream.
   - **Enhanced compression**—enables enhanced compression on PR bitstream.
   - **Generate encrypted bitstream**—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (`.ekp`). If you enable **Generate encrypted bitstream**, enable or disable the **Enable volatile security key**, **Use encryption lock file**, and **Generate key programming file** options.
6. Click **Generate**. The PR bitstream files generate according to your specifications.

Figure 21. PMSF File Properties Bitstream Encryption



1.6.9.2. Generating PR Bitstream Files

For Agilex 7, Agilex 5, and Stratix 10 designs, the Assembler generates a configuration `.rbf` automatically at the end of compilation. For Arria 10 and Cyclone 10 GX designs, use any of the following methods to process the PR bitstreams and generate the Raw Binary File (`.rbf`) file for reconfiguration.

Generating PR Bitstreams During Compilation

Follow these steps to generate the `.rbf` file during compilation for Arria 10 and Cyclone 10 GX designs:

1. Add the following assignments to the revision `.qsf` to automatically generate the required PR bitstreams following compilation:

```
set_global_assignment -name GENERATE_PR_RBF_FILE ON
set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
```

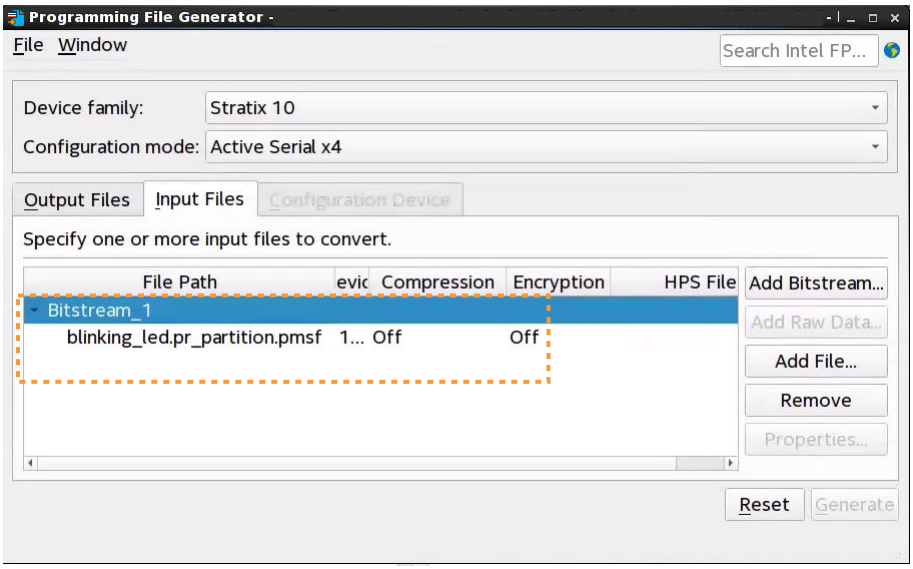
2. To compile the revision and generate the `.rbf`, click **Processing ➤ Start Compilation**.

Generating PR Bitstreams with Programming File Generator

Follow these steps to generate the `.rbf` for PR programming with the **Programming File Generator**:

1. Click **File ➤ Programming File Generator**. The **Programming File Generator** appears.
2. Specify the target **Device family** and the **Configuration mode** for partial reconfiguration.
3. On the **Output File** tab, specify the **Output directory**, file **name**, and enable the **Raw Binary File for Partial Reconfiguration (.rbf)** file type.
4. To add the input `.pmsf` file to convert, click the **Input Files** tab, click **Add Bitstream**, and specify the `.pmsf` that you generated in the Assembler.

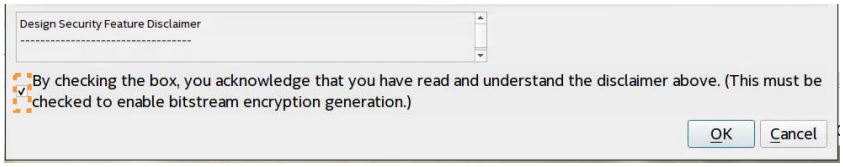
Figure 22. Adding Bitstream File



5. On the **Input Files** tab, select the bitstream `.pmsf` file and click **Properties**. Specify any of the following options for the `.rbf`:

- **Enable compression**—generates compressed PR bitstream files to reduce file size.
- **Enable encryption**—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (`.ekp`). You can also specify other **Security settings**.
- If you turn on **Enable encryption**, you must also acknowledge the **Design Security Feature Disclaimer** by checking the box.

Figure 23. Design Security Feature Disclaimer



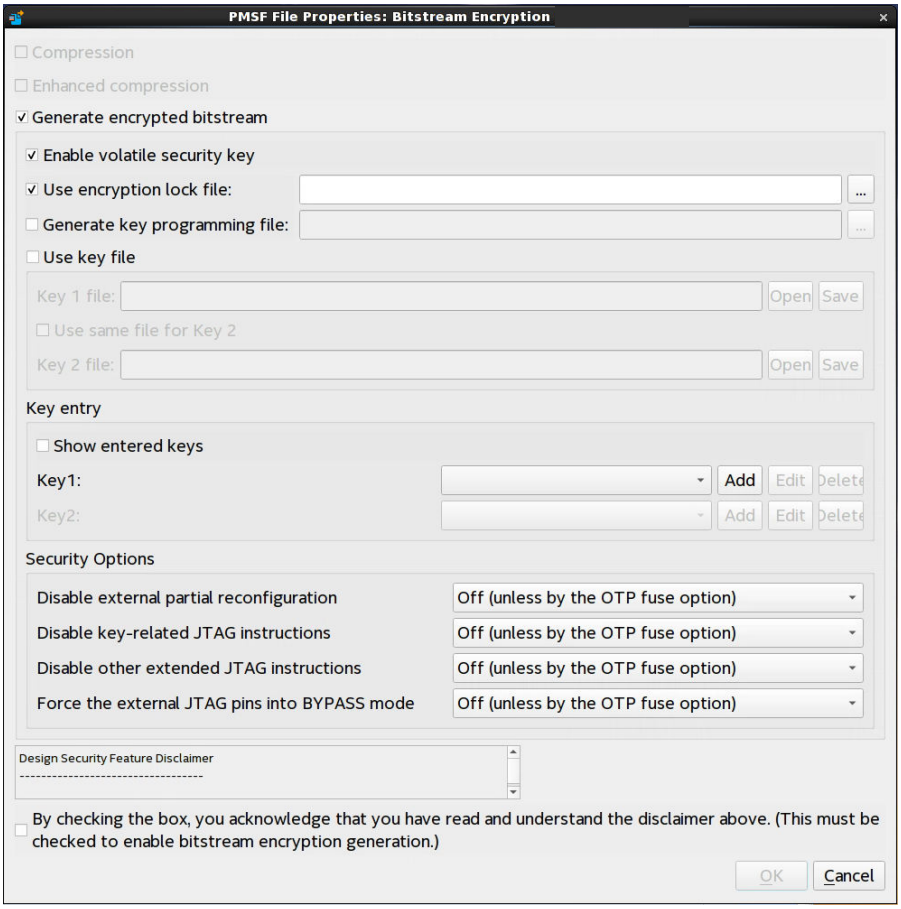
6. Click **OK**.
7. In **Programming File Generator**, click **Generate**. The PR bitstream files generate according to your specifications.

Generating PR Bitstreams with Convert Programming Files Dialog Box

Follow these steps to generate the `.rbf` with the **Convert Programming Files** dialog box:

1. Click **File ➤ Convert Programming Files**. The **Convert Programming Files** dialog box appears.
2. Specify the output file name and **Programming file type** as **Raw Binary File for Partial Reconfiguration (.rbf)**.
3. To add the input `.pmsf` file to convert, click **Add File**.
4. Select the newly added `.pmsf` file, and click **Properties**.
5. Enable or disable any of the following options and click **OK**:
   - **Compression**—enables compression on PR bitstream.
   - **Enhanced compression**—enables enhanced compression on PR bitstream.
   - **Generate encrypted bitstream**—generates encrypted independent bitstreams for base image and PR image. You can encrypt the PR image even if your base image has no encryption. The PR image can have a separate encryption key file (`.ekp`). If you enable **Generate encrypted bitstream**, enable or disable the **Enable volatile security key**, **Use encryption lock file**, and **Generate key programming file** options.
6. Click **Generate**. The PR bitstream files generate according to your specifications.

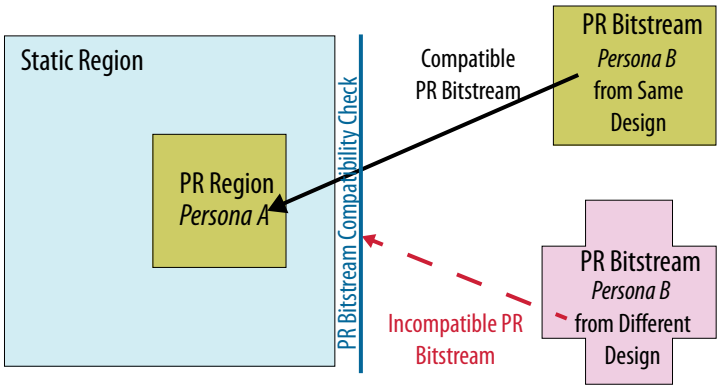
Figure 24. PMSF File Properties Bitstream Encryption



1.6.9.3. Partial Reconfiguration Bitstream Compatibility Checking

Partial reconfiguration bitstream compatibility checking verifies the compatibility of the reconfiguration bitstream to prevent configuration with an incompatible PR bitstream. The following sections describe PR bitstream compatibility check support.

Figure 25. PR Bitstream Compatibility Checking



PR Bitstream Compatibility Checking for Agilex 7, Agilex 5, and Stratix 10 Designs

For Agilex 7, Agilex 5, and Stratix 10 designs, PR bitstream compatibility checking is automatically enabled in the Compiler and in the Secure Device Manager (SDM) firmware by default. The following limitations apply to PR designs *if PR bitstream compatibility checking is enabled*:

- The firmware allows up to a total of 32 PR regions, irrespective of the number of hierarchical partial reconfiguration layers.
- Your PR design can have up to six hierarchical partial reconfiguration layers.
- Your PR design, when there is no hierarchy, can have up to 32 regions.
- Your PR design can have up to 15 child PR regions of any parent PR region (if it is hierarchical). Child PR regions count towards the total limit of 32 PR regions.

The Compiler generates an error if your PR design exceeds these limits when PR bitstream compatibility checking is enabled.

If you require more PR regions than this limitation allows, or otherwise want to disable PR bitstream compatibility checking , you can add the following assignment to the `.qsf` file:

```
set_global_assignment -name ENABLE_PR_POF_ID OFF
```

When you set this assignment to off, the limit of 32 total regions does not apply in the Compiler.

Note: If you require the PR bitstream authentication feature for your design, you must enable PR bitstream compatibility checking by setting the global assignment `ENABLE_PR_POF ID` to `ON`. The default setting is `ON`.

Arria 10 and Cyclone 10 GX PR Bitstream Compatibility Checking

For Arria 10 and Cyclone 10 GX designs, you enable or disable PR bitstream compatibility checking by turning on the **Enable bitstream compatibility check** option when instantiating the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP from the IP Catalog.

The PR IP verifies the partial reconfiguration PR Bitstream file (`.rbf`). When you enable the bitstream compatibility check, the PR `.pof` ID is encoded as the 71st word of the PR bitstream. If the PR IP detects an incompatible bitstream, then the PR IP stops the PR operation, and the status output reports an error.

When you turn on **Enable bitstream compatibility check**, the PR Controller IP core creates a **PR bitstream ID** and displays the bitstream ID in the configuration dialog box. For bitstream compatibility checking with hierarchical PR designs, refer to additional steps in *AN 806: Hierarchical Partial Reconfiguration Tutorial for Arria 10 GX FPGA Development Board*.

Related Information

AN 806: Hierarchical Partial Reconfiguration Tutorial for Intel Arria 10 GX FPGA Development Board

1.6.9.4. Raw Binary Programming File Byte Sequence Transmission Examples

The raw binary programming file (`.rbf`) file contains the device configuration data in little-endian raw binary format. The following example shows transmitting the `.rbf` byte sequence 02 1B EE 01 in x32 mode:

Table 5. Writing to the PR control block or SDM in x32 mode

In x32 mode, the first byte in the file is the least significant byte of the configuration double word, and the fourth byte is the most significant byte.

Double Word = 01EE1B02			
LSB: BYTE0 = 02	BYTE1 = 1B	BYTE2 = EE	MSB: BYTE3 = 01
D[7..0]	D[15..8]	D[23..16]	D[31..24]
0000 0010	0001 1011	1110 1110	0000 0001

1.6.9.5. Generating a Merged .pmsf File from Multiple .pmsf Files (Arria 10 and Cyclone 10 GX Designs)

Use a single merged `.rbf` file to reconfigure two PR regions simultaneously.

Note: This procedure supports only Arria 10 and Cyclone 10 GX devices. Agilex 7, Agilex 5, and Stratix 10 devices do not support merging `.pmsf` files.

To merge two or more `.pmsf` files:

1. Open the **Convert Programming Files** dialog box.
2. Specify the output file name and programming file type as **Merged Partial-Mask SRAM Object File (.pmsf)**.
3. In the **Input files to convert** dialog box, select **PMSF Data**.
4. To add input files, click **Add File**. You must specify two or more files for merging.
5. To generate the merged file, click **Generate**.

Alternatively, to merge two or more `.pmsf` files from the Quartus Prime shell, type the following command:

```
quartus_cpf --merge_pmsf=<number of merged files> <pmsf_input_file_1> \
     <pmsf_input_file_2> <pmsf_input_file_etc> <pmsf_output_file>
```

For example, to merge two `.pmsf` files, type the following command:

```
quartus_cpf --merge_pmsf=2 foo.pmsf bat.pmsf \
     combine.pmsf
```

After creating the merged `.pmsf`, generate a `.rbf`, as Generating PR Bitstream Files on page 28 describes.

1.7. Partial Reconfiguration Design Considerations

Partial reconfiguration is an advanced design flow in the Quartus Prime Pro Edition software. Creating a partial reconfiguration design requires an understanding of how the PR design guidelines apply to your design. When designing for partial reconfiguration, you must consider the entire system-level behavior initial conditions to maintain the integrity and correctness of the static region operation.

For example, during PR programming, you must ensure that other parts of the system do not read or write to the PR region. You must also freeze the write enable output from the PR region into the static region, to avoid interference with static region operation. If all personas for your design do not have identical top-level interfaces, you must create the wrapper logic to ensure that all the personas appear similar to the static region. Upon partial reconfiguration of a PR region, you must bring the registers in the PR region to a known state by applying a reset sequence. There are specific guidelines for global signals and on-chip memories. The following sections provide design considerations and guidelines to help you create design files for a PR design.

FPGA Device and Software Considerations

- All Agilex 7, Agilex 5, Stratix 10, Arria 10, and Cyclone 10 GX devices support partial reconfiguration.
- Use the nominal VCC of 0.9V or 0.95V as per the datasheet, including VID enabled devices.
- To minimize Arria 10 and Cyclone 10 GX programming files size, ensure that the PR regions are short and wide. For Agilex 7, Agilex 5, and Stratix 10 designs, use sector-aligned PR regions.
- The Quartus Prime Standard Edition software does not support partial reconfiguration for Arria 10 devices, nor provide any support for Agilex 7, Agilex 5, or Stratix 10 devices.
- The current version of the Quartus Prime Pro Edition software supports only one Signal Tap File (`.stp`) per revision.

Design Partition Considerations

- Reconfigurable partitions can only contain core resources, such as LABs, RAMs, and DSPs. All periphery resources, such as the transceivers, external memory interface, HPS, and clocks must be in the static portion of the design.
- To physically partition the device between static and individual PR regions, floorplan each PR region into exclusive, core-only, placement regions, with associated routing regions.
- A reconfiguration partition must contain the super-set of all ports that you use across all PR personas.

Clocking, Reset, and Freeze Signal Considerations

- The maximum number of clocks or other global signals for any Arria 10 or Cyclone 10 GX PR region is 33. The maximum number of clocks or other global signals for any Agilex 7, Agilex 5, and Stratix 10 PR region is 32. In the current version of the Quartus Prime Pro Edition software, no two PR regions can share a row-clock.
- PR regions do not require any input freeze logic. However, you must freeze all the outputs of each PR region to a known constant value to avoid unknown data during partial reconfiguration.
- Increase the reset length by 1 cycle to account for register duplication in the Fitter.
- Ensure that all low-skew global signals (clocks and resets) driving into PR regions in base revision compilations have destinations.
- In Agilex 7 and Agilex 5 devices, you must use global clock resources to clock M20K RAMs in PR regions. The Fitter issues an error if an M20K in a PR region is driven by a clock port from a locally routed clock.

1.7.1. Partial Reconfiguration Design Guidelines

The following table lists important design guidelines at various steps in the PR design flow:

Table 6. Partial Reconfiguration Design Guidelines

PR Design Step	Guideline	Reason
Designing for partial reconfiguration	Do not assume initial states in registers inside PR region. After PR is complete, reset all the control path registers to a known state. Unless required for your scenario, you can omit the data path registers from the reset.	Registers inside the PR region contain undefined values after reconfiguration. Omitting data path registers reduces congestion on reset signals. However, resetting the data registers is required in some cases.(3)
	You cannot define synchronous reset as a global signal for Arria 10 or Cyclone 10 GX partial reconfiguration.	PR regions do not support synchronous reset of registers as a global signal, because the Arria 10 and Cyclone 10 GX LAB does not support synchronous clear (`sclr`) signal on a global buffer. The LAB supports the asynchronous clear (`aclr`) signal driven from a local input, or from a global network row clock. As a result, only the `aclr` can be a global signal, feeding registers in a PR region.
	The PRESERVE_FANOUT_FREE_NODE assignment cannot preserve a fanout-free register that has no fanout inside the Verilog HDL or VHDL module in which you define it. To preserve these fanout-free registers, implement the `noprune` pragma in the source file: `(*noprune*)reg r;` If there are multiple instances of this module, with only some instances requiring preservation of the fanout-free register, set a dummy pragma on the register in the HDL and also set the PRESERVE_FANOUT_FREE_NODE assignment. This dummy pragma allows the register synthesis to implement the assignment. For example, set the following dummy pragma for a register `r` in Verilog HDL: `(*dummy*)reg r;` Then set this instance assignment: `set_instance_assignment -name \ PRESERVE_FANOUT_FREE_NODE ON \ -to r;`	The PRESERVE_FANOUT_FREE_NODE assignment does not apply when a register is not used in the Verilog HDL or VHDL module in which it is defined.
Partitioning the design	Register all the inputs and outputs for your PR region.	Improves timing closure and time budgeting.

continued...

(3) For example, there are occurrences where registers are being duplicated while simultaneously the register value is undefined after PR. The duplicated registers can result in a mismatch between the parity bit and the data written to the MLAB after PR. Hence, a different value is used to compute the parity bit compared with the actual data written to the MLAB, requiring reset of the data register, or re-write of the value to the register after PR.

PR Design Step	Guideline	Reason
	Reduce the number of signals interfacing the PR region with the static region in your design.	Reduces the wire LUT count.
	Create a wrapper for your PR region.	The wrapper creates a common footprint to the static region.
	Drive all the PR region output ports to inactive state when the PR region is held in reset and the freeze bit is asserted for the PR region.	Prevents the static region logic from receiving random data during the partial reconfiguration operation.
	PR boundary I/O interface must be a superset of all the PR persona I/O interfaces.	Ensures that each PR partition implements the same ports.
Preparing for partial reconfiguration	Complete all pending transactions.	Ensures that the static region is not in a wait state.
Maintaining a partially working system during partial reconfiguration	Hold all outputs to known constant values.	Ensures that the undefined values the PR region receives during and after the reconfiguration do not affect the PR control logic.
Initializing after partial reconfiguration	Initialize after reset.	Retrieves state from memory or other device resources.
Debugging the design using Signal Tap Logic Analyzer	Store all the tapped signals from a persona in one `.stp` file.	The current version of the Quartus Prime software supports only one `.stp` (Signal Tap file) per revision. This limitation requires you to select partitions, one at a time, to tap.
	Do not tap across regions in the same `.stp` file.	Ensures consistent interface (boundary) across all personas.
	Tap only the pre-synthesis signals. In the Node Finder, filter for **Signal Tap: pre-synthesis**.	Ensures that the signal tapping of PR personas start from synthesis.

1.7.2. PR Design Timing Closure Best Practices

The use of partition boundary ports for PR regions can make timing closure more challenging because the Compiler cannot optimize the logic across a partition boundary. The use of Logic Lock regions can also limit placement and routing flexibility. You must register all PR region boundary ports. Even when taking these steps, you may still find timing criticalities.

Each persona of a PR region can have different bits or input and output buses in use. Therefore, it is important to preserve the registers that do not have fan-out in a given persona, or that are driven by constants. You must ensure that the Compiler does not optimize away such registers during the compilation of a persona.

If the base compile does not use some bits of a bus, and the Compiler removes the corresponding registers for those bits, the logic may be untimed, resulting in unfavorable placement and routing. If you use those unregistered paths in other persona logic, you can have difficulty meeting timing on those paths. Preserving the unused port registers in the base compile ensures that the paths are timed in the base compile, and eases timing closure during persona compiles.

Follow these guidelines for effective register preservation in PR designs:

- Only the registers within PR regions require preservation.
- Only the PR base compilation requires register preservation.
- In a persona compile, the Compiler can safely remove fan-out free and constant-driven registers.
- For hierarchical PR compilations, only the base compile of the hierarchy requires register preservation.
- Preserve fan-out free nodes for input registers.
- Preserve constant-driven nodes for output registers.
- Only assign the attributes for the PR base compile. Remove the attributes for the persona compile (for example, via parameter or generic).
- You can set top-level parameters in the `.qsf`, which in turn pass down to lower hierarchies.

Use any of the following synthesis attributes to preserve registers:

- To preserve constant-driven or fan-out free registers, use the `noprune` attribute. `noprune` also disables all physical optimizations:

```
Verilog: (* noprune *) reg reg1;
VHDL: signal reg1: std_logic;
      attribute noprune: boolean;
      attribute noprune of reg1: signal is true;
```

- To preserve fan-out free registers while allowing retiming on bits that have fan-outs, assign PRESERVE_FANOUT_FREE_NODE ON as `altera_attribute`:

```
Verilog: (* altera_attribute = "-name PRESERVE_FANOUT_FREE_NODE ON" *) \
     reg reg1;
VHDL: signal reg1: stdlogic;
      attribute altera_attribute : string;
      attribute altera_attribute of reg1: signal is "-name \
     PRESERVE_FANOUT_FREE_NODE ON";
```

- Alternatively, use the `dummy` attribute with the PRESERVE_FANOUT_FREE_NODE ON assignment in the `.qsf`:

```
Verilog: (* dummy *) reg reg1;
VHDL: signal reg1: std_logic;
      attribute dummy: boolean;
      attribute dummy of reg1: signal is true;
```

  `.qsf` Assignment:

```
set_instance_assignment -name PRESERVE_FANOUT_FREE_NODE ON \
    -to <hierarchical path to reg1>
```

- To preserve constant-driven registers while allowing retiming on bits that have drivers, use the `preserve_syn_only` attribute:

```
Verilog: (* preserve_syn_only *) reg reg1;
VHDL: signal reg1: std_logic;
      attribute preserve_syn_only : boolean;
      attribute preserve_syn_only of reg1: signal is true;
```

The following example shows how to assign attributes in PR base compile using a parameter in System Verilog and in VHDL. The example contains a parameter called `base_compile`, which is set to `true` for the PR base compile only.

```
System Verilog:
localparam ON_OFF_STRING = base_compile ? "ON": "OFF";
(* altera_attribute = {"-name PRESERVE_FANOUT_FREE_NODE ", ON_OFF_STRING} *)
logic [WIDTH-1:0] pr_input_register;
(* altera_attribute = {"-name PRESERVE_REGISTER_SYN_ONLY ", ON_OFF_STRING} *)
logic [WIDTH-1:0] pr_output_regsiter;

VHDL:
attribute altera_attribute : string;
type attributeStr_type is array(boolean) of string(1 to 35);
constant attributeStr : attributeStr_type := (true => "-name
PRESERVE_FANOUT_FREE_NODE ON \
     ", false => "-name PRESERVE_FANOUT_FREE_NODE OFF");
attribute altera_attribute of  <PR input registers> : signal is
attributeStr(base_compile);
attribute preserve_syn_only : boolean;
attribute preserve_syn_only of  <PR output registers> : signal is base_compile;
```

Related Information

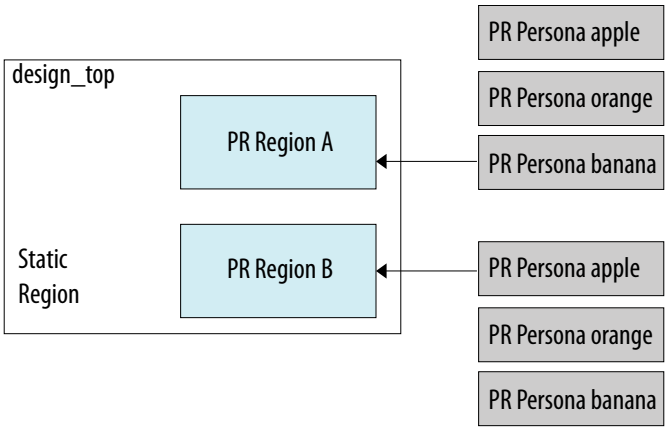
AN 899: Reducing Compile Time with Fast Preservation

1.7.3. PR File Management

You can simplify the management of PR personas and their corresponding source files by observing one of the following PR project file management methods.

To illustrate these methods, consider a design that includes two PR regions, each with the possible apple, orange, and banana personas.

Figure 26. Example Design with Two PR Regions and Three Personas



- Method 1 (Preferred): Specify Unique Entity and File Names for Each Persona on page 39
- Method 2: Set QSF Assignment for a Parameterized PR Persona on page 40

1.7.3.1. Method 1 (Preferred): Specify Unique Entity and File Names for Each Persona

In PR file management method 1, you specify unique entity and file name pairs for each persona in the project. For example:

- Define the apple persona in the `apple.sv` file
- Define the orange persona in the `orange.sv` file
- Define the banana persona in the `banana.sv` file

Note: For successful compilation and PR operation, all personas must have the exact same port names and widths defined in each `.sv` file.

In the base PR revision RTL, you specify "apple" as the PR persona for both PR regions:

Figure 27. Setting the Base PR Persona to "apple"

```
module design_top ();

    apple u_fruit_0 ( );

    apple u_fruit_1 ( );

endmodule
```

When you set the base persona to [`apple`, `apple`] by setting `u_fruit_0` and `u_fruit_1` as the PR partition and regions, you can easily change the persona occupying the PR region using the **Entity Rebinding** (`ENTITY_REBINDING`) option in the Design Partitions Window, or by editing the `.qsf` directly, as the following examples show:

To specify the orange persona for a PR implementation (impl) revision:

```
set_instance_assignment -name ENTITY_REBINDING orange -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING orange -to u_fruit_1
```

To specify the banana persona for another PR implementation (impl) revision:

```
set_instance_assignment -name ENTITY_REBINDING banana -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING banana -to u_fruit_1
```

To specify different personas for each PR region in an implementation revision:

```
set_instance_assignment -name ENTITY_REBINDING orange -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING banana -to u_fruit_1
```

For each implementation revision, you must ensure that you include the corresponding source file in the project (**Project ➤ Add/Remove Files in Project**).

1.7.3.2. Method 2: Set QSF Assignment for a Parameterized PR Persona

In PR file management method 2, you specify an assignment in the `.qsf` file that sets a parameter or generic to a targeted PR region. The following apply to this method:

- The parameter change applies to the top-level instance of a PR partition.
- Supports application to multiple PR partitions.
- Supports both VHDL and Verilog HDL.

For example, consider the design that Method 1 (Preferred): Specify Unique Entity and File Names for Each Persona on page 39 describes, with two PR regions, each with three possible personas for each PR region.

In the following example, `u_fruit_0` and `u_fruit_1` are set as the PR partitions and regions in the base compile. The FRUIT_TYPE parameter of `0` generates the `apple` entity for the PR personas.

Figure 28. Setting the PR Partitions and Regions in the Base Compile

```
//design_top.sv
module design_top ();

   fruit u_fruit_0 ( );

   fruit u_fruit_1 ( );

endmodule
```

```
module fruit
   #(parameter FRUIT_TYPE=0)
   (  );

   generate if (FRUIT_TYPE==0) begin: gen_apple
       apple appl_inst ( );
   end
   endgenerate

   generate if (FRUIT_TYPE==1) begin: gen_orange
       orange orange_inst ( );
   end
   endgenerate

   generate if (FRUIT_TYPE==2) begin: gen_banana
       banana banana_inst ( );
   end
   endgenerate
endmodule
```

You can then change the parameter values to change the personas.

For example, to set `orange` as the persona for both PR regions, specify the following in the PR implementation revision's `.qsf` file:

1. Add the following lines to set the FRUIT_TYPE parameter to `1`:

```
set_instance_assignment -name RTL_PARAMETER "FRUIT_TYPE=1" -to u_fruit_0
set_instance_assignment -name RTL_PARAMETER "FRUIT_TYPE=1" -to u_fruit_1
```

2. Specify the entity rebinding assignment to associate the fruit entity with instances of `u_fruit_0` and `u_fruit_1`:

```
set_instance_assignment -name ENTITY_REBINDING fruit -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING fruit -to u_fruit_1
```

The following additional example sets `orange` as the persona for the first PR region, and `banana` as the persona for the second PR region. Similarly, specify the following in the PR implementation revision's `.qsf` file:

1. Add the following lines to set the FRUIT_TYPE parameter to 1 for the first PR region, `u_fruit_0` and 2 for the second PR region, `u_fruit_1`:

```
set_instance_assignment -name RTL_PARAMETER "FRUIT_TYPE=1" -to u_fruit_0
set_instance_assignment -name RTL_PARAMETER "FRUIT_TYPE=2" -to u_fruit_1
```

2. Specify the entity rebinding assignment to associate the fruit entity with instances of `u_fruit_0` and `u_fruit_1`:

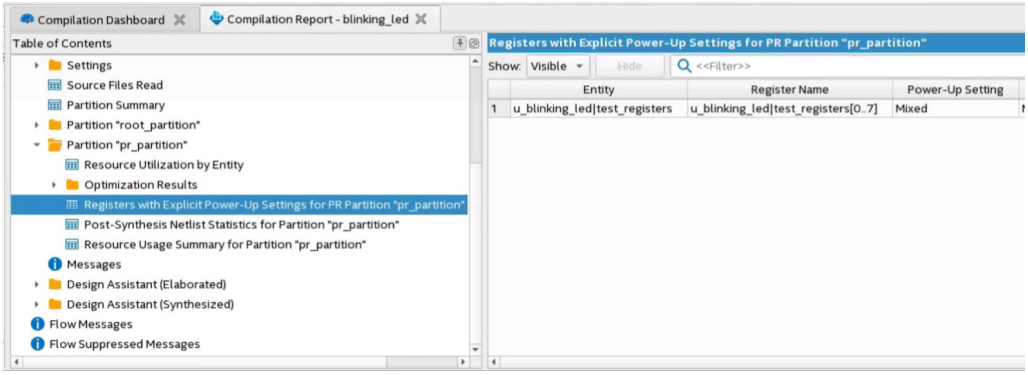
```
set_instance_assignment -name ENTITY_REBINDING fruit -to u_fruit_0
set_instance_assignment -name ENTITY_REBINDING fruit -to u_fruit_1
```

1.7.4. Evaluating PR Region Initial Conditions

Unintended initial conditions in a PR region can lead to errors during partial reconfiguration. Your design may include unintended initial conditions, especially if you port a design not originally intended for partial reconfiguration. The Quartus Prime Pro Edition software reports any initial conditions in the PR partitions for your evaluation following synthesis.

After compiling the base revision that defines the partition, you can view the Registers with Explicit Power-Up Settings report for the partition to identify, locate, and correct any unintended initial conditions. For a specific PR partition, you can view the power-up initial values after synthesizing the base revision in the Synthesis report. The Synthesis report includes power-up initial values in the Partition Statistics section.

Figure 29. Partition Statistics in Synthesis Report



The Messages window also generates a warning or error message about any initial conditions during synthesis processing. After evaluating the initial condition, you can determine whether the condition is correct for design functionality, or change the design to remove dependence on an initial condition that is incompatible with partial reconfiguration.

1.7.5. Creating Wrapper Logic for PR Regions

If all personas for your design do not have identical top-level interfaces, you must create the wrapper logic to ensure that all the personas appear similar to the static region. Define a wrapper for each persona, and instantiate the persona logic within the

wrapper. If all personas have identical top-level interfaces, the personas do not require wrapper logic. In this wrapper, you can create dummy ports to ensure that all the personas of a PR region have the same connection to the static region.

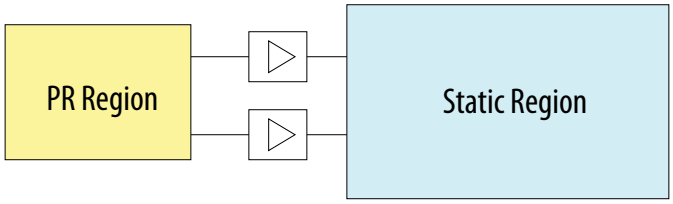
During the PR compilation, the Compiler converts each of the non-global ports on interfaces of the PR region into boundary port wire LUTS. The naming convention for boundary port wire LUTs are `<input_port>~IPORT` for input ports, and `<output_port>~OPORT` for output ports. For example, the instance name of the wire LUT for an input port with the name `my_input`, on a PR region with the name `my_region`, is `my_region|my_input~IPORT`.

1. Manually floorplan the boundary ports using Logic Lock region assignments, or place the boundary ports automatically using the Fitter. The Fitter places the boundary ports during the base revision compile. The boundary LUTs are invariant locations the Fitter derives from the persona you compile. These LUTs represent the boundaries between the static region and the PR routing and logic. The placement remains stationary regardless of the underlying persona, because the routing from the static logic does not vary with a different persona implementation.
2. To constrain all boundary ports within a given region, use a wildcard assignment. For example:

```
set_instance_assignment -name PLACE_REGION "65 59 65 85" -to \
     u_my_top|design_inst|pr_inst|pr_inputs.data_in*~IPORT
```

   This assignment constrains all the wire LUTS corresponding to the IPORTS that you specify within the place region, between the coordinates (65 59) and (65 85).

Figure 30. Wire-LUTs at the PR Region Boundary



Optionally, floorplan the boundary ports down to the LAB level, or individual LUT level. To floorplan to the LAB level, create a 1x1 Logic Lock `PLACE_REGION` constraint (single LAB tall and a single LAB wide). Optionally, specify a range constraint by creating a Logic Lock placement region that spans the range. For more information about floorplan assignments, refer to *Floorplan the Partial Reconfiguration Design*.

Related Information

Step 3: Floorplan the Design on page 14
For more information on floorplanning your design.

1.7.6. Creating Freeze Logic for PR Regions

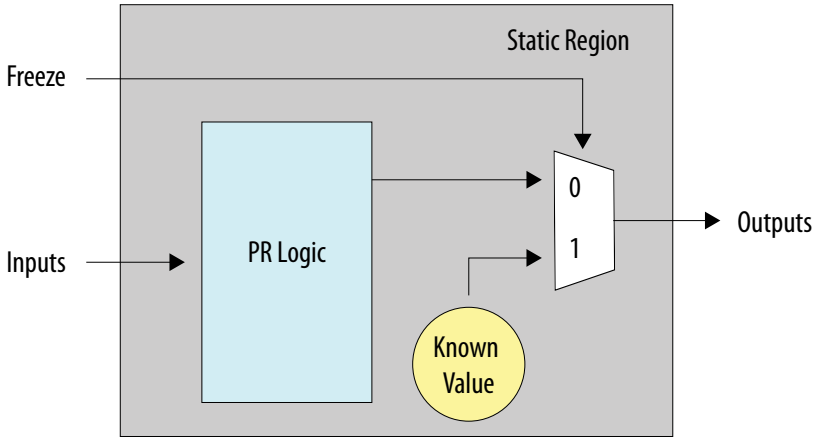
When partially reconfiguring a design, freeze all the outputs of each PR region to a known constant value. This freezing prevents the signal receivers in the static region from receiving undefined signals during the partial reconfiguration process.

The PR region cannot drive valid data until the partial reconfiguration process is complete, and the PR region is reset. Freezing is important for control signals that you drive from the PR region.

The freeze technique that you choose is optional, depending on the particular characteristics of your design. The freeze logic must reside in the static region of your design. A common freeze technique is to instantiate 2-to-1 multiplexers on each output of the PR region, to hold the output constant during partial reconfiguration.

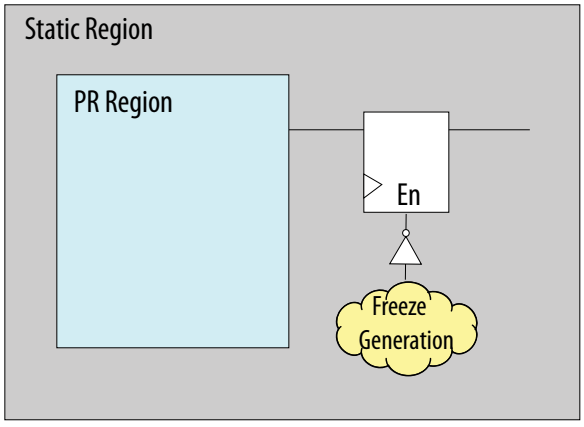
Note: There is no requirement to freeze the global and non-global inputs of a PR region.

Figure 31. Freeze Technique #1



An alternative freeze technique is to register all outputs of the PR region in the static region. Then, use an enable signal to hold the output of these registers constant during partial reconfiguration.

Figure 32. Freeze Technique #2



The Partial Reconfiguration Region Controller IP core includes a freeze port for the region that it controls. Include this IP component with your system-level control logic to freeze the PR region output. For designs with multiple PR regions, instantiate one PR Region Controller IP core for each PR region in the design. The Quartus Prime

software includes the Avalon Memory-Mapped Freeze Bridge and Avalon Streaming Freeze Bridge Intel FPGA IP cores. You can use these IP cores to implement freeze logic, or design your own freeze logic for these standard interface types.

The static region logic must be independent of all the outputs from the PR regions for a continuous operation. Control the outputs of the PR regions by adding the appropriate freeze logic for your design.

1.7.7. Resetting the PR Region Registers

Upon partial reconfiguration of a PR region, the status of the PR region registers become indeterminate. Bring the registers in the PR region to a known state by applying a reset sequence for the PR region. This reset ensures that the system behaves to your specifications. Simply reset the control path of the PR region, if the datapath eventually flushes out within a finite number of cycles. Use active-high local reset instead of active-low, wherever applicable. This technique allows you to automatically hold the PR region in reset, by virtue of the boundary port wire LUT.

Table 7. Supported PR Reset Implementation Guideline

PR Reset Type	Active-High Synchronous Reset	Active-High Asynchronous Reset	Active-Low Synchronous Reset	Active-Low Asynchronous Reset
On local signal	Yes	Yes	Yes	Yes
On global signal	• No (Arria 10) • No (Cyclone 10 GX) • Yes (Stratix 10) • Yes (Agilex 7) • Yes (Agilex 5)	Yes	• No (Arria 10) • No (Cyclone 10 GX) • Yes (Stratix 10) • Yes (Agilex 7) • Yes (Agilex 5)	Yes

1.7.8. Promoting Global Signals in a PR Region

In non-PR designs, the Quartus Prime software automatically promotes high fan-out signals onto dedicated global networks. The global promotion occurs in the Plan stage of design compilation.

In PR designs, the Compiler disables global promotion for signals originating within the logic of a PR region. Instantiate the clock control blocks only in the static region, because the clock floorplan and the clock buffers must be a part of the static region of the design. Manually instantiating a clock control block in a PR region, or assigning a signal in a PR region with the `GLOBAL_SIGNAL` assignment, results in compilation error. To drive a signal originating from the PR region onto a global network:

1. Expose the signal from the PR region.
2. Drive the signal onto the global network from the static region.
3. Drive the signal back into the PR region.

You can drive a maximum of 33 clocks (for Arria 10 and Cyclone 10 GX devices), or 32 clocks (for Agilex 7, Agilex 5, and Stratix 10 devices) into any PR region. You cannot share a row clock between two PR regions.

The Compiler allows only certain signals to be global inside a PR region. Use only global signals to route secondary signals into a PR region, as the following table describes:

Table 8. Supported Signal Types for Driving Clock Networks in a PR Region

Block Type	Supported Global Network Signals
LAB, MLAB	Clock, ACLR, SCLR[4]
RAM, ROM (M20K)	Clock, ACLR, Write Enable (WE), Read Enable (RE), SCLR
DSP	Clock, ACLR, SCLR

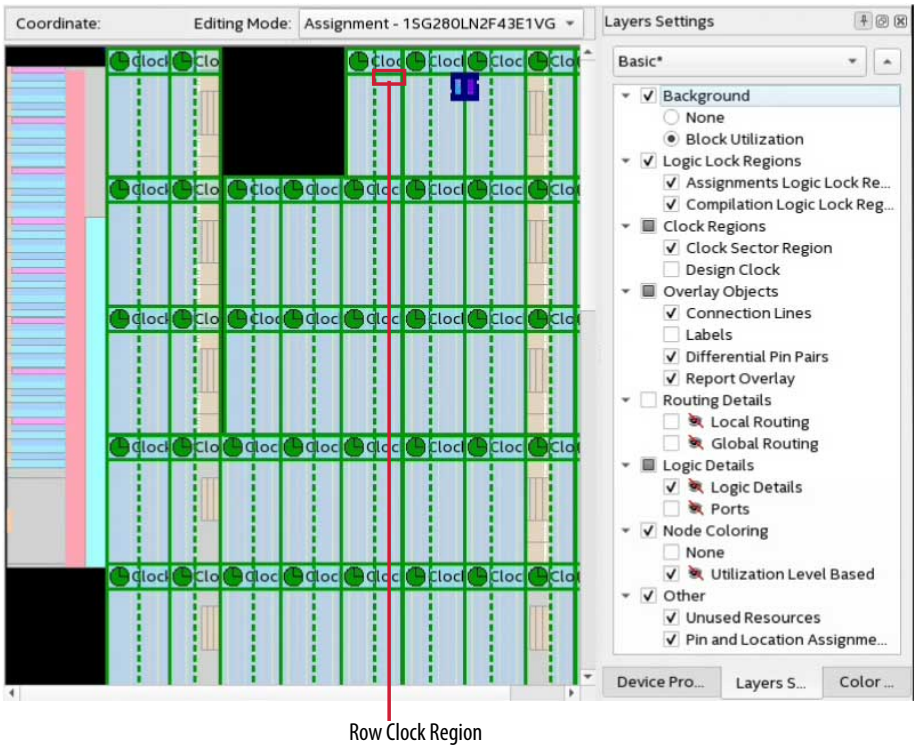
1.7.8.1. Viewing Row Clock Region Boundaries

You can use the Chip Planner to visualize the row clock region boundaries, and to ensure that no two PR regions share a row clock region.

1. Right-click a PR partition name in the Design Partitions Window and click **Locate Node ➤ Locate in Chip Planner**.

Figure 33. Row Clock Region Boundaries in Chip Planner

The green borders in the following floorplan figure indicate clock sectors. A row clock region is one half of a clock sector wide, and one LAB row tall, as indicated by the red box.



2. In Chip Planner, click the **Layers** tab and select the **Basic** layer. The Chip Planner overlays the row clock region boundaries. Adjust the **Basic** layer settings to display specific items.

(4) Only Agilex 7, Agilex 5, and Stratix 10 designs support global SCLR.

1.7.9. Planning Clocks and other Global Routing

There are special PR considerations for the planning for clocks and other global routing. For Agilex 7, Agilex 5, and Stratix 10 designs, you can use the low skew networks (globals) for clocks or resets.

During the base revision compile, you must route any global signal that any PR persona requires into a destination in the PR region. For clock signals, this destination is a register or other synchronous element and the signal entering the clock input. For a reset, the destination should be fed into the appropriate input.

This requirement occurs because PR only reconfigures the last part of the low skew network. If you do not route the root and middle sections of the network during the base compile, you cannot use that revision for the PR.

Consider an example with a super-set of signals for a PR region that consists of:

- Three clocks—`clk_1`, `clk_2`, and `clk_3`.
- Two resets—`rst_1` and `rst_2`.
- Base PR persona—uses `clk_1`, `clk_2`, and `rst_1` only.
- Other personas—use `clk_3` and `rst_2` only.

In this example, the base persona must have a proper destination for the "unused" `clk_3` and `rst_2`. You can accomplish this by driving a single register with a (`*no prune*`) directive inside the base PR persona, with `clk_3` and reset using `rst_2`.

Omitting these destinations results in an error during compilation of the PR implementation second persona.

1.7.10. Implementing Clock Enable for On-Chip Memories

Follow these guidelines to implement clock enable for on-chip memories:

1. To avoid spurious writes during PR programming for memories, implement the clock enable circuit in the same PR region as the M20K or MLAB RAM. This circuit depends on an active-high clear signal from the static region.
2. Before you begin the PR programming, assert this signal to disable the memory's clock enable. Your system PR controller must deassert the clear signal on PR programming completion. You can use the freeze signal for this purpose.
3. Use the Quartus Prime IP Catalog or Platform Designer to instantiate the On-Chip Memory and RAM Intel FPGA IP cores that include an option to automatically add this circuitry.

   Note: If you turn on the **Implement clock-enable circuitry for use in a partial reconfiguration region** option when parameterizing RAM Intel FPGA IP from the IP catalog, the Quartus Prime software adds a `freeze` port to the RAM IP for use in the PR region.

Figure 34. Clock-Enable Circuitry Option in RAM 1 Port Intel FPGA IP Parameter Editor

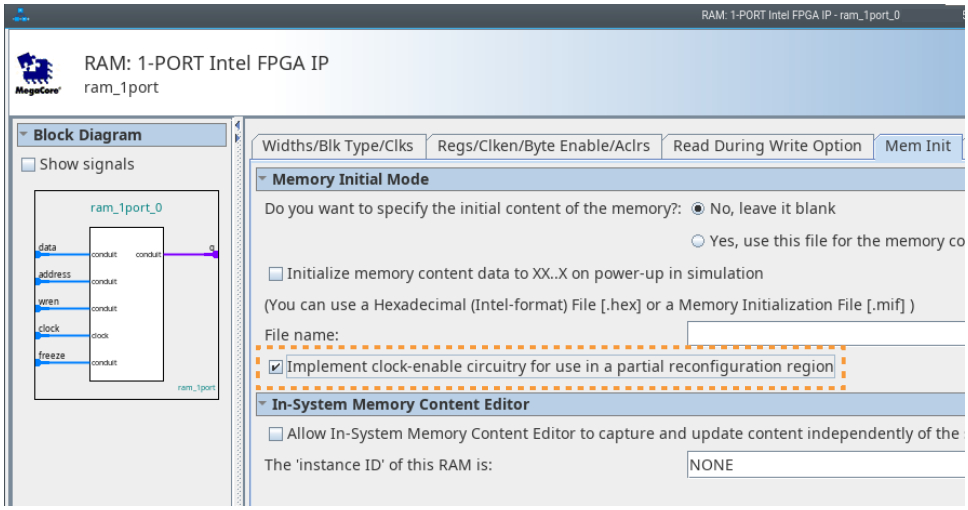
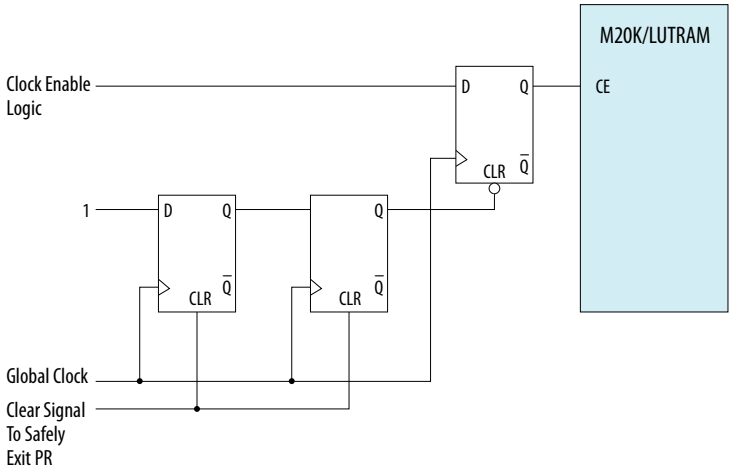


Figure 35. RAM Clock Enable Circuit for PR Region



Example 1. Verilog RTL for Clock Enable

```
module mem_enable_verilog (
        input clock,
        input freeze,
        input clken_in,
        output wire ram_wrclocken
);
   reg ce_reg;
    reg [1:0] ce_delay;

    always @(posedge clock, posedge freeze) begin
        if (freeze) begin
            ce_delay <= 2'b0;
        end
        else begin
            ce_delay <= {ce_delay[0], 1'b1};
        end
    end

    always @(posedge clock, negedge ce_delay[1]) begin
```

```
        if (~ce_delay[1]) begin
            ce_reg <= 1'b0;
        end
        else begin
            ce_reg <= clken_in;
        end
    end

    assign ram_wrclocken = ce_reg;
endmodule
```

Example 2. VHDL RTL for Clock Enable

```
ENTITY mem_enable_vhd IS PORT(
        clock      : in  std_logic;
        freeze  : in  std_logic;
        clken_in : in std_logic;
        ram_wrclocken : out std_logic);
END mem_enable_vhd;

ARCHITECTURE behave OF mem_enable_vhd is
    SIGNAL ce_reg: std_logic;
    SIGNAL ce_delay: std_logic_vector(1 downto 0);
BEGIN
PROCESS (clock, freeze)
BEGIN
    IF ((clock'EVENT AND clock = '1') or (freeze'EVENT AND freeze = '1')) THEN
        IF (freeze = '1') THEN
            ce_delay <= "00";
        ELSE
            ce_delay <= ce_delay(0) & '1';
        END IF;
    END IF;

END PROCESS;

PROCESS (clock, ce_delay(1))
BEGIN
    IF ((clock'EVENT AND clock = '1') or (ce_delay(1)'EVENT AND ce_delay(1) =
'0')) THEN
        IF (ce_delay(1) = '0') THEN
            ce_reg <= '0';
        ELSE
            ce_reg <= clken_in;
        END IF;
    END IF;

END PROCESS;

ram_wrclocken <= ce_reg;

END ARCHITECTURE behave;
```

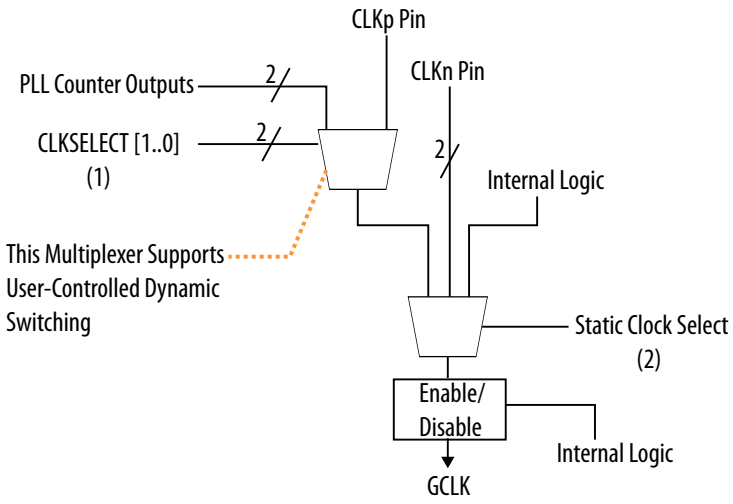
Related Information

Embedded Memory User Guide

1.7.10.1. Clock Gating

An alternate method to avoid spurious writes of initialized content memories is to implement clock gating circuitry in the PR static region, and feed the clock gating circuitry to the PR region in which the initialized memories are implemented.

Figure 36. Global Clock Control Block



Implement the gating circuitry in the static region, and feed it to the PR region in which the initialized memories are being implemented. Clock gating is logically equivalent to using clock enable on the memories. This method provides the following benefits:

- Uses the enable port of the global clock buffers to disable the clock before starting the partial reconfiguration operation. Also enables the clock on PR completion.
- Ensures that the clock does not switch during reconfiguration, and requires no additional logic to avoid spurious writes.

Related Information

Clock Control Block (ALTCLKCTRL) Intel FPGA IP User Guide

1.8. Hierarchical Partial Reconfiguration

Hierarchical partial reconfiguration (HPR) is an extension of partial reconfiguration (PR), where you contain one PR region within another PR region. You can create multiple personas for both the child and parent partitions. You nest the child partitions within their parent partitions. Reconfiguring a parent partition does not impact the operation in the static region, but replaces the child partitions of the parent region with default child partition personas.

The HPR design flow includes the following steps:

1. Create a base revision for the design and export the static region, as Step 7: Compile the Base Revision and Export the Static Region on page 21 describes.
2. Create the implementation revision for each persona, as Step 8: Setup PR Implementation Revisions on page 25 describes, and export the parent partitions.
3. Specify the `.qdb` file partition for the static and parent regions.
4. Specify the corresponding entity for the parent or child.

When compiling the implementation revision for an HPR design, you must fully floorplan the child partition, similar to planning the PR region of a base revision. Refer to Using Parent QDB Files from Different Compiles on page 50.

Note: Hierarchical PR (HPR) designs do not support PR bitstream security verification.

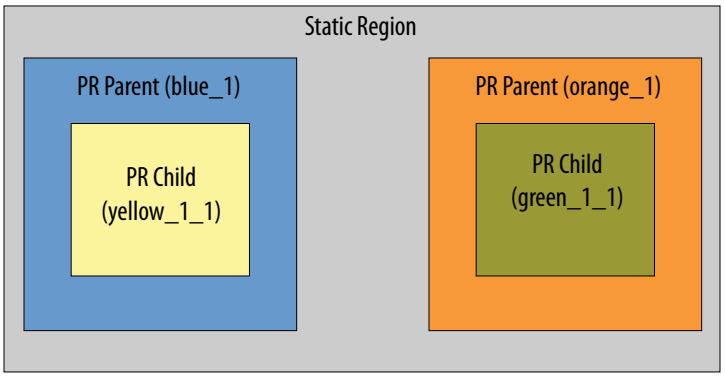
Related Information

AN826: Hierarchical Partial Reconfiguration Tutorial for Intel Stratix 10 GX FPGA Development Board
for step-by-step HPR instructions

1.8.1. Using Parent QDB Files from Different Compiles

For HPR designs, you can use the parent `.qdb` file from the same or different implementation compiles. The following examples illustrate two possible HPR compilation flows, with respect to the following design example block diagram:

Figure 37. Example HPR Design with Parent QDB Files from Different Compiles



Example HPR Design with Parent QDB Files from Different Compiles shows an HPR design hierarchy with the following characteristics:

- The blue and orange regions represent the HPR parent regions.
- The yellow and green boxes represent the child PR regions.
- The blue HPR parent has two personas, blue_1 and blue_2.
- For the yellow child region, the default persona that is compiled with parent blue_1 is yellow_1_1.
- The second child persona that can be compiled by blue_1 is yellow_2_1.
- The orange HPR parent has the same characteristics as the blue HPR parent.

Considering these HPR design characteristics, the following describes one possible HPR compilation flow:

HPR Compilation Flow A:

1. Blue_1, yellow_1_1, orange_1, green_1_1.
2. `Blue_1.qdb`, yellow_1_2, `orange_1.qdb`, green_1_2
3. Blue_2, yellow_2_1, orange_2, green_2_1
4. `Blue_2.qdb`, yellow_2_2, `orange_2.qdb`, green_2_2

In Flow A, in steps 2 and 4, the parent region `.qdb` files come from the same implementation compile. Step 2 uses `blue_1.qdb` and `orange_1.qdb` that step 1 generates in the same implementation compile.

HPR also supports import of the parent and child PR partitions from different implementation compiles:

HPR Compilation Flow B:

1. Blue_1, yellow_1_1, orange_1, green_1_1.
2. Blue_2, yellow_2_1, orange_2, green_2_1
3. `Blue_1.qdb`, yellow_1_2, `orange_2.qdb`, green_2_2
4. `Blue_2.qdb`, yellow_2_2, `orange_1.qdb`, green_1_2

In Flow B, `blue_1.qdb` and `orange_2.qdb` come from two different implementation compiles. Step 1 generates `blue_1.qdb`. Step 2 implementation compile generates `orange_2.qdb`.

1.9. Partial Reconfiguration Design Timing Analysis

The interface between partial and static partitions remains the same for each PR implementation revision. Perform timing analysis on each PR implementation revision to ensure that there are no timing violations. To ensure timing closure of a design with multiple PR regions, you can create aggregate revisions for all possible PR region combinations for timing analysis.

Note: Logic Lock regions impose placement constraints that affect the performance and resource utilization of your PR design. Ensure that the design has additional timing allowance and available device resources. Selecting the largest and most timing-critical persona as your base persona optimizes the timing closure. In addition, if you compile the base design with time borrowing enabled, compile the implementation designs with time borrowing enabled. Otherwise, time borrowing amounts in the base design are reset to zero, and the design may not pass timing. If this condition occurs, you can use the `update_timing_netlist –recompute_borrow` command to restore time borrowing amounts throughout the design for timing analysis.

Related Information

Quartus Prime Pro Edition User Guide: Timing Analyzer

1.9.1. Running Timing Analysis on Aggregate Revisions

To ensure timing closure of a design with multiple PR regions, you create aggregate revisions for all possible PR region combinations and run timing analysis.

1. To open the **Revisions** dialog box, click **Project ➤ Revisions**.
2. To create a new revision, double-click **<<new revision>>**.
3. Specify the **Revision name** and select the base revision for **Based on Revision**.
4. To export the post-fit database from the base compile (static partition), type the following command in the Quartus Prime shell:

```
quartus_cdb <project name> <base revision> --export_block \
     "root_partition" --snapshot final --file \
     "<base revision name>.qdb"
```

Note: Ensure that you include all the `.sdc` and `.ip` files for the static and PR regions. To detect the clocks, ensure that the `.sdc` file for the PR Controller IP follows the entry of any `.sdc` file that creates the clocks that the IP core uses. You facilitate this order by ensuring the `.ip` file for the PR Controller IP comes after any `.ip` or `.sdc` files that you use to create these clocks in the `.qsf` file for the project revision. Refer to Partial Reconfiguration Solutions IP User Guide on page 74 for more information.

5. To export the post-fit database from multiple personas (for the PR implementation revisions), type the following commands in the Quartus Prime shell:

```
quartus_cdb <project name> -c <PR1 revision> --export_block \
     <PR1 Partition name> --snapshot final --file "pr1.qdb"
quartus_cdb <project name> -c <PR2 revision> --export_block \
     <PR2 Partition name> --snapshot final --file "pr2.qdb"
```

6. To import the post-fit databases of the static region as an aggregate revision, type the following commands in the Quartus Prime shell:

```
quartus_cdb <project name> -c <aggr_rev> --import_block \
     "root_partition" --file "<base revision name>.qdb"
quartus_cdb <project name> -c <aggr_rev> --import_block \
     <PR1 partition name> --file "pr1.qdb"
quartus_cdb <project name> -c <aggr_rev> --import_block \
     <PR2 Partition name> --file "pr2.qdb"
```

7. To integrate post-fit database of all the partitions, type the following command in the Quartus Prime shell:

```
quartus_fit <project name> -c <aggr_rev>
```

Note: The Fitter verifies the legality of the post-fit database, and combines the netlist for timing analysis. The Fitter does not reroute the design.

8. To perform timing analysis on the aggregate revision, type the following command in the Quartus Prime shell:

```
quartus_sta <proj name> -c <aggr_rev>
```

9. Run timing analysis on aggregate revision for all possible PR persona combinations. If a specific persona fails timing closure, recompile the persona and perform timing analysis again.

1.10. Partial Reconfiguration Design Simulation

Simulation verifies the behavior of your design before device programming. The Quartus Prime Pro Edition software supports simulating the delivery of a partial reconfiguration bitstream to the PR region. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

The Quartus Prime Pro Edition software supports simulation of PR persona transitions through the use of simulation multiplexers. You use the simulation multiplexers to change which persona drives logic inside the PR region during simulation. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

Similar to non-PR design simulations, preparing for a PR simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation.

The Quartus Prime software provides simulation components to help simulate a PR design, and can generate the gate-level PR simulation models for each persona. Use either the behavioral RTL or the gate-level PR simulation model for simulation of the PR personas. The gate-level PR simulation model allows for accurate simulation of registers in your design and reset sequence verification. These technology-mapped registers do not assume initial conditions.

You can use the PR mode of the EDA netlist writer to generate the gate level netlist of a PR region. Refer to the "EDA Netlist Writer and Gate Level-Netlists" section of the *Quartus Prime Pro Edition User Guide: Third Party Simulation*.

Related Information

- Generating and Simulating Intel FPGA IP on page 133
- Quartus Prime Pro Edition User Guide: Third Party Simulation

1.10.1. Partial Reconfiguration Simulation Flow

At a high-level, a PR operation consists of the following steps:

1. System-level preparation for a PR event.
2. Retrieval of the partial bitstream from memory.
3. Transmission of the partial bitstream to the PR control block or SDM.
4. Resulting change in the design as a new persona becomes active.
5. Post-PR system coordination.
6. Use of the new persona in the system.

You can simulate each of these process steps in isolation, or as a larger sequence depending on your verification type requirement.

Related Information

- Arria 10 and Cyclone 10 GX PR Control Block Simulation Model on page 138
- Generating the PR Persona Simulation Model on page 140
- Secure Device Manager Partial Reconfiguration Simulation Model on page 143

1.10.2. Simulating PR Persona Replacement

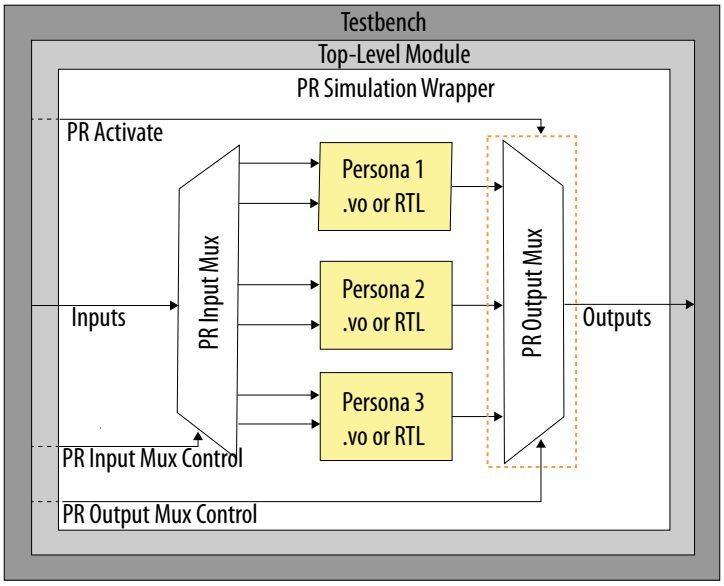
The logical operation of the PR partition changes when a new persona loads during the partial reconfiguration process. Simulate the replacement of personas using multiplexers on the input and output of the persona under simulation.

Create RTL wrapper logic to represent the top-level of the persona. The wrapper instantiates the default persona during compilation. During simulation, the wrapper allows the replacement of the active persona with another persona. Instantiate each persona as the behavioral RTL in the PR simulation model the Quartus Prime EDA Netlist Writer generates.

The Quartus Prime software includes simulation modules to interface with your simulation testbench:

- `altera_pr_wrapper_mux_in`
- `altera_pr_wrapper_mux_out`
- `altera_pr_persona_if` (SystemVerilog interface allows you to connect the wrapper multiplexers to a testbench driver)

Figure 38. Simulation of PR Persona Switching



Example 3. RTL Wrapper for PR Persona Switching Simulation

The `pr_activate` input of the `altera_pr_wrapper_mux_out` module enables the MUX to output X. This functionality allows the simulation of unknown outputs from the PR persona, and also verifies the normal operation of the design's freeze logic. The following code corresponds to the simulation of PR persona switching, shown in the above figure:

```
module pr_core_wrapper
(
   input wire a,
   input wire b,
   output wire o
);

localparam ENABLE_PERSONA_1 = 1;
localparam ENABLE_PERSONA_2 = 1;
localparam ENABLE_PERSONA_3 = 1;
localparam NUM_PERSONA = 3;

logic pr_activate;
int persona_select;

altera_pr_persona_if persona_bfm();
assign pr_activate = persona_bfm.pr_activate;
assign persona_select = persona_bfm.persona_select;


wire a_mux [NUM_PERSONA-1:0];
```

```
wire b_mux [NUM_PERSONA-1:0];
wire o_mux [NUM_PERSONA-1:0];

generate
   if (ENABLE_PERSONA_1) begin
      localparam persona_id = 0;

      `ifdef ALTERA_ENABLE_PR_MODEL
         assign u_persona_0.altera_sim_pr_activate = pr_activate;
      `endif

      pr_and u_persona_0
      (
         .a(a_mux[persona_id]),
         .b(b_mux[persona_id]),
         .o(o_mux[persona_id])
      );
   end
endgenerate

generate
   if (ENABLE_PERSONA_2) begin
      localparam persona_id = 1;

      `ifdef ALTERA_ENABLE_PR_MODEL
         assign u_persona_1.altera_sim_pr_activate = pr_activate;
      `endif

      pr_or u_persona_1
      (
         .a(a_mux[persona_id]),
         .b(b_mux[persona_id]),
         .o(o_mux[persona_id])
      );

   end
endgenerate

generate
   if (ENABLE_PERSONA_3) begin
      localparam persona_id = 2;

      `ifdef ALTERA ENABLE PR MODEL
         assign u_persona_2.altera_sim_pr_activate = pr_activate;
      `endif

      pr_empty u_persona_2
      (
         .a(a_mux[persona_id]),
         .b(b_mux[persona_id]),
         .o(o_mux[persona_id])
      );

   end
endgenerate


altera_pr_wrapper_mux_in #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1)) \
     u_a_mux(.sel(persona_select), .mux_in(a), .mux_out(a_mux));

altera_pr_wrapper_mux_in #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1)) \
     u_b_mux(.sel(persona_select), .mux_in(b), .mux_out(b_mux));

altera_pr_wrapper_mux_out #(.NUM_PERSONA(NUM_PERSONA), .WIDTH(1)) \
     u_o_mux(.sel(persona_select), .mux_in(o_mux), .mux_out(o), .pr_activate \
     (pr_activate));

endmodule
```

1.10.2.1. altera_pr_persona_if Module

Instantiate the altera_pr_persona_if SystemVerilog interface in a PR region simulation wrapper to connect to all the wrapper multiplexers. Optionally, connect pr_activate to the PR simulation model.

Connect the interface's persona_select to the sel port of all input and output multiplexers. Connect the pr_activate to the pr_activate of all the output multiplexers. Optionally, connect the report events to the report event ports of the PR simulation model. Then, the PR region driver testbench component can drive the interface.

```
interface altera_pr_persona_if;
   logic pr_activate;
   int   persona_select;

   event report_storage_if_x_event;
   event report_storage_if_1_event;
   event report_storage_if_0_event;
   event report_storage_event;

   initial begin
      pr_activate <= 1'b0;
   end
endinterface : altera_pr_persona_if
```

The <*QUARTUS_INSTALL_DIR*>/eda/sim_lib/altera_lnsim.sv file defines the altera_pr_persona_if component.

1.10.2.2. altera_pr_wrapper_mux_out Module

The altera_pr_wrapper_mux_out module allows you to multiplex the outputs of all PR personas to the outputs of the PR region wrapper.

Instantiate one multiplexer per output port. Specify the active persona using the sel port of the multiplexer. The pr_activate port allows you to drive the multiplexer output to "x", to emulate the unknown value of PR region outputs during a PR operation. Parameterize the component to specify the number of persona inputs, the multiplexer width, and the MUX output value when pr_activate asserts.

```
module altera_pr_wrapper_mux_out #(
   parameter NUM_PERSONA = 1,
   parameter WIDTH = 1,
   parameter [0:0] DISABLED_OUTPUT_VAL = 1'bx
) (
   input int sel,
   input wire [WIDTH-1 : 0] mux_in [NUM_PERSONA-1:0],
   output reg [WIDTH-1:0]   mux_out,
   input wire               pr_activate
);

   always_comb begin
      if ((sel < NUM_PERSONA) && (!pr_activate))
         mux_out = mux_in[sel];
      else
         mux_out = {WIDTH{DISABLED_OUTPUT_VAL}};
   end

endmodule : altera_pr_wrapper_mux_out
```

The <*QUARTUS_INSTALL_DIR*>/eda/sim_lib/altera_lnsim.sv file defines the altera_pr_wrapper_mux_out component.

1.10.2.3. `altera_pr_wrapper_mux_in` Module

The `altera_pr_wrapper_mux_in` module allows you to de-multiplex inputs to a PR partition wrapper for all PR personas.

Instantiate one multiplexer per input port. Specify the active persona using the `sel` port of the multiplexer. Parameterize the component to specify the number of persona outputs, the multiplexer width, and the MUX output for any disabled output. When using the `altera_pr_wrapper_mux_in` to mux a clock input, use the `DISABLED_OUTPUT_VAL` of 0, to ensure there are no simulation clock events of the disabled personas.

```
module altera_pr_wrapper_mux_in#(
   parameter NUM_PERSONA = 1,
   parameter WIDTH = 1,
   parameter [0:0] DISABLED_OUTPUT_VAL = 1'bx
) (

   input int sel,
   input wire [WIDTH-1:0] mux_in,
   output reg [WIDTH-1 : 0] mux_out [NUM_PERSONA-1:0]
);
   always_comb begin
      for (int i = 0; i < NUM_PERSONA; i++)
         if (i == sel)
            mux_out[i] = mux_in;
         else
            mux_out[i] = {WIDTH{DISABLED_OUTPUT_VAL}};
   end


endmodule : altera_pr_wrapper_mux_in
```

The *<QUARTUS_INSTALL_DIR>*`/eda/sim_lib/altera_lnsim.sv` file defines the `altera_pr_wrapper_mux_in` component.

1.11. Partial Reconfiguration Design Debugging

The following Intel FPGA IP cores support system-level debugging in the static region of a PR design:

- In-System Memory Content Editor
- In-System Sources and Probes Editor
- Virtual JTAG
- Nios II JTAG Debug Module
- Signal Tap Logic Analyzer

In addition, the Signal Tap logic analyzer allows you to debug the static or partial reconfiguration (PR) regions of the design. If you only want to debug the static region, you can use the In-System Sources and Probes Editor, In-System Memory Content Editor, or System Console with a JTAG Avalon bridge.

Related Information

- System Debugging Tools Overview
- AN 841: Signal Tap Tutorial for Intel Stratix 10 Partial Reconfiguration Design

1.11.1. Debugging PR Designs with the Signal Tap Logic Analyzer

To use the Signal Tap logic analyzer to debug PR designs, you must create a debug bridge to extend Signal Tap debugging into the PR partition. You can then use Signal Tap to debug by connecting to the debug bridge. To use the debug bridge, you instantiate the SLD JTAG Bridge Agent Intel FPGA IP, SLD JTAG Bridge Host Intel FPGA IP, and Intel Configuration Reset Release Endpoint to Debug Logic IP for each PR region in your design.

You must instantiate the following IP in your design to ensure you can use Signal Tap to debug your PR region:

1. Instantiate the SLD JTAG Bridge Agent IP in the static region.
2. Instantiate the SLD JTAG Bridge Host IP and the Intel Configuration Reset Release Endpoint to Debug Logic IP in the PR region of the default persona.
3. Instantiate the SLD JTAG Bridge Host IP and the Intel Configuration Reset Release Endpoint to Debug Logic IP, for each of the personas, whenever creating revisions for the personas.

The Signal Tap logic analyzer uses the hierarchical debug capabilities provided by the Quartus Prime software to tap signals in the static and PR regions simultaneously.

You can debug multiple personas present in your PR region, as well as multiple PR regions. For complete information on the debug infrastructure using hierarchical hubs, refer to *Quartus Prime Pro Edition User Guide: Debug Tools*.

1.11.2. Instantiating the Intel Configuration Reset Release Endpoint to Debug Logic IP

You must instantiate the Intel Configuration Reset Release Endpoint to Debug Logic IP in each PR region if multiple PR regions are present in the design. This IP ensures proper function by providing a reset signal to debug logic, such as Signal Tap logic, after partial reconfiguration. This reset signal must be high during configuration, and then this reset signal must go low once partial reconfiguration is complete. You must not release this reset signal after releasing the PR logic reset. The time of this reset release affects the Signal Tap power-up trigger feature. The reset signal must stay low until the next reconfiguration.

Note: Do not assert this reset input while the device is in the user operational mode. Asserting this reset input while the device is in the user operational mode results in incorrect operation in Signal Tap and other debugging tools.

If you omit the Intel Configuration Reset Release Endpoint to Debug Logic IP from your PR design, The Compiler issues the following error message:

```
Error(11176): Alt_sld_fab_1.alt_sld_fab_1.alt_sld_fab_1: The Intel Configuration
Reset Release
Endpoint to Debug Logic IP must be instantiated to provide the reset signal to
the debug logic,
such as Signal Tap, etc. after the partial configuration is performed.
```

Refer to the Intel FPGA Knowledge Database and search for Error 11176 for more information.

Related Information

Intel FPGA Knowledge Database

1.12. Partial Reconfiguration Security (Stratix 10 Designs)

Stratix 10 devices support the following optional PR security features to help confirm that a PR region persona is protected, contains no threats to platform integrity or confidentiality, and cannot access unauthorized areas of the FPGA device before loading a persona into the FPGA.

- PR Bitstream Security Validation—confirms that the persona does not use FPGA resources that are unauthorized by validating a `.pmsf` against a Secure Mask Settings File (`.smsf`), as PR Bitstream Security Validation (Stratix 10 Designs) on page 59 describes.
- PR Bitstream Authentication—ensures that the firmware and PR bitstream are from a trusted source by provisioning the FPGA device with the owner public root key, as PR Bitstream Authentication (Stratix 10 Designs) on page 61 describes.
- PR Bitstream Encryption—protects the bitstream contents by encrypting the static region and all associated bitstreams using the same AES root key, as PR Bitstream Encryption (Stratix 10 Designs) on page 61 describes.

1.12.1. PR Bitstream Security Validation (Stratix 10 Designs)

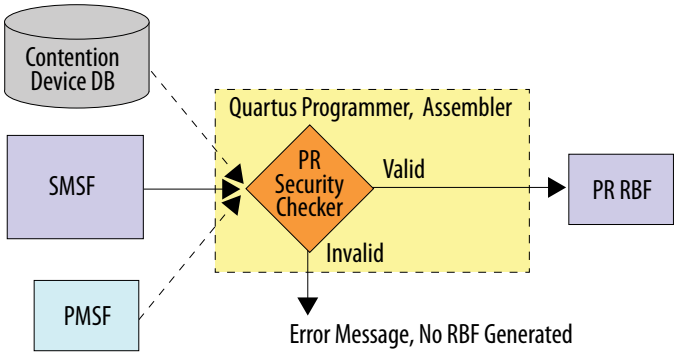
PR bitstream security validation confirms that the persona does not access FPGA resources that are unauthorized by the platform owner.

Note: PR bitstream security validation only supports Stratix 10 devices. Hierarchical PR (HPR) designs do not support PR bitstream security verification.

PR bitstream security validation enables multi-tenant FPGA usage. For example, a platform owner partitions a single device to host multiple third-party clients. The platform owner may not trust the clients, and the clients may not trust each other, but the clients trust the platform owner. PR bitstream security validation provides the platform owner and clients protection from any party corrupting the proprietary server, the client configurations, or from initiating a peek or poke attack by a subsequent partial reconfiguration.

PR bitstream validation allows the platform owner to determine whether the client has modified their `.pmsf` file in an attempt to damage the FPGA, or has attempted connection to signals without access. To be effective, the platform owner must accept only `.pmsf` files (not `.rbf`) from the client, and the platform owner must validate all client `.pmsf` files. Thereafter, the Programmer requires both the `.pmsf` and `.smsf` to generate the PR bitstream (`.rbf`) for this PR region, ensuring that the PR persona can only change bits that the persona owns. The Platform Owner can optionally release `.smsf` files to third-party Clients as part of the PR region collateral.

Figure 39. PR Bitstream Security Validation in Programmer



For PR bitstream validation, the platform owner generates the `.smsf` file themselves, to ensure that the platform owner can trust the `.smsf`. The bitstream validation check compares the client supplied `.pmsf` against the trusted `.smsf`. The comparison fails if the `.pmsf` is invalid for deliberate or accidental reasons.

The Platform Owner should follow these steps to license, enable, and use PR bitstream security validation:

1. Obtain the license file to enable generation of `.smsf` files for PR regions during base compilation, and to perform PR bitstream security validation during PR bitstream generation in the Programmer. To obtain the license, login or register for a My Intel account, and then submit an Intel Premier Support case quoting reference number 22013030316 to request a license key.
2. To add the license file to the Quartus Prime Pro Edition software, click **Tools ➤ License Setup** and specify the feature **License File**.
3. To enable PR security validation features, add the following line to the project `.qsf`:

```
set_global_assignment -name PR_SECURITY_VALIDATION on
```

4. Compile the base revision.
5. Following base compilation, view the Assembler reports to view the generated `.smsf` files required for bitstream generation for each PR region.
6. The Client provides the `.pmsf` to the Platform Owner.
7. The Platform Owner validates the `.pmsf`, converts the `.pmsf` to `.rbf`, and configures the FPGA device with the `.rbf`. The Platform Owner converts the `.pmsf` to a PR bitstream. Provide the `.smsf` file to `quartus_pfg` to instruct the tool to validate the `.pmsf` against that `.smsf`. Then generate a bitstream only if the files are compatible.

```
quartus_pfg -c -o smsf_file=<smsf_file> <pmsf_file> <output_rbf_file>
```

Related Information

Stratix 10 Device Security User Guide

1.12.2. PR Bitstream Authentication (Stratix 10 Designs)

PR bitstream authentication helps to ensure that the firmware and the PR bitstream are from a trusted source, by provisioning the FPGA device with the owner public root key. Authentication is a basic component of device security and bitstream protection.

In PR bitstream authentication, the signed base bitstream must first be configured to the device. Then, the signed PR bitstream is used to configure one or more partial reconfiguration regions of the FPGA device. The signed PR bitstream must match the configured static region.

The following use cases summarize successful and unsuccessful PR bitstream authentication:

PR Authentication Success Use Case:

- **Partial Reconfiguration with Authenticated PR Bitstream**—in a successful PR authentication use case, the designer performs full chip configuration using an authenticated `.sof` file. The designer can only configure the partially reconfigurable regions of the FPGA that are signed with the design signature private key, and that match the currently configured static region. The PR bitstreams are authenticated to ensure that only authorized users can provide the PR bitstream.

PR Authentication Failure Scenarios

The following are some PR authentication failure scenarios:

- **PR Bitstream Is Unsigned**—when the target FPGA device determines that the PR bitstream is unsigned, then the PR operation halts and PR bitstream security displays a PR error message.
- **PR Bitstream Is Signed with Expired or Invalid Signature**—when the target FPGA device determines that the PR bitstream is signed with an expired or invalid signature, then the PR operation halts and PR bitstream security displays a PR error message.
- **PR Success after PR Failure from Expired or Invalid Signature**—when PR of the target FPGA device fails with an error caused by an expired or invalid signature, you can provide a bitstream signed with a valid key to perform the PR operation successfully.

Related Information

Stratix 10 Device Security User Guide

1.12.3. PR Bitstream Encryption (Stratix 10 Designs)

PR bitstream encryption helps protect the bitstream. You can configure each PR region with multiple PR bitstream files. Any of these files may contain sensitive or valuable data that encryption can protect. PR bitstream encryption allows you to encrypt the static region and all associated bitstreams using the same AES root key.

Note: PR bitstream authentication is a prerequisite of PR bitstream encryption use. You must enable PR bitstream authentication before using PR bitstream encryption.

In PR bitstream encryption, you must first configure the device with the encrypted base bitstream. Next, you configure one or more partial reconfiguration regions with the encrypted PR bitstream. The encrypted PR bitstream must match the configured static region.

You also can configure the signed PR bitstream after the first encrypted base bitstream configuration. For all subsequent partial reconfigurations, both the signed and encrypted PR bitstreams are supported.

PR bitstream encryption requires the following prerequisite conditions:

- The Base and PR designs must share the same authentication key.
- The Base and PR designs must share the same encryption key.
- All PR regions must be encrypted or none. A combination of encrypted and non-encrypted designs is unsupported.
- When you enable authentication, both the base and the PR design must be authenticated. This requirement ensures that only authorized users can provide the full or PR bitstream to the owned FPGA device.
- When you enable authentication or encryption, the Quartus Prime Assembler skips the auto-generation of `.rbf` files for PR designs, and only generates the `.pmsf` file.

Note: For bitstream encryption details, refer to the *Stratix 10 Device Security User Guide*.

Related Information

Stratix 10 Device Security User Guide

1.13. PR Bitstream Compression and Encryption (Arria 10 and Cyclone 10 GX Designs)

You can compress and encrypt the base bitstream and the PR bitstream for your Arria 10 and Cyclone 10 GX PR project using options available in the Quartus Prime software.

Compress the base and PR programming bitstreams independently, based on your design requirements. When encrypting only the base image, specify whether or not to encrypt the PR images. The following guidelines apply to PR bitstream compression and encryption:

- You can encrypt the base and PR image independently. You can use a non-volatile encryption key for the base image, and a volatile encryption key for the PR image.
- Refer to Clock-To-Data Ratio for Bitstream Encryption and Compression to ensure the correct Clock-to-Data (CD) ratio setting for encryption or compression.

Enable enhanced decompression by turning on the **Enable enhanced decompression** option when specifying the parameters in the IP Catalog or Platform Designer parameter editors.

Note: You cannot use enhanced decompression together with encryption simultaneously. Enhanced decompression is only available with the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP.

1.13.1. Generating an Encrypted PR Bitstream (Arria 10 or Cyclone 10 GX Designs)

To partially reconfigure your Arria 10 or Cyclone 10 GX device with an encrypted bitstream:

1. Create a 256-bit key file (`.key`).
2. To generate the key programming file (`.ekp`) from the Quartus Prime shell, type the following command:

```
quartus_cpf --key <keyfile>:<keyid> \
     <base_sof_file> <output_ekp_file>
```

   For example:

```
quartus_cpf --key my_key.key:key1 base.sof key.ekp
```

3. To generate the encrypted PR bitstream (`.rbf`), run the following command:

```
quartus_cpf -c <pr_pmsf_file> <pr_rbf_file>
qcrypt -e --keyfile=<keyfile> --keyname=<keyid> –lockto=\
     <qlk file> --keystore=<battery|OTP> \
     <pr_rbf_file> <pr_encrypted_rbf_file>
```

   - `lockto`—specifies the encryption lock.
   - `keystore`—specifies the volatile key (battery) or the non-volatile key (OTP).

   For example:

```
quartus_cpf -c top_v1.pr_region.pmsf top_v1.pr_region.rbf \
     qcrypt -e --keyfile=my_key.key --keyname=key1 --keystore=battery \
     top_v1.pr_region.rbf top_v1_encrypted.rbf
```

4. To program the key file as volatile key (default) into the device, type the following command:

```
quartus_pgm -m jtag -o P;<output_ekp_file>
```

   For example:

```
quartus_pgm -m jtag -o P;key.ekp
```

5. To program the base image into the device, type the following command:

```
quartus_pgm -m jtag -o P;<base_sof_file>
```

   For example:

```
quartus_pgm -m jtag -o P;base.sof
```

6. To partially reconfigure the device with the encrypted bitstream, type the following command:

```
quartus_pgm -m jtag --pr <output_encrypted_rbf_file>
```

   For example:

```
quartus_pgm -m jtag --pr top_v1_encrypted.rbf
```

Note: `qcrypt` generates an error if the **Enable bitstream compatibility check** parameter is enabled for an instance of the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP. Use one of the following methods to avoid this error:

- Use the **Convert Programming Files** dialog box, rather than `qcrypt`, to generate the encrypted PR bitstream, as *Generating PR Bitstream Files* describes.
- If you want use `qcrypt` with Arria 10 or Cyclone 10 GX designs, regenerate the Partial Reconfiguration Controller IP without the **Enable bitstream compatibility check** option enabled, and with the **Enable hierarchical PR support** option enabled, as *Adding the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP* describes. Recompile the design before regenerating the PR bitstream.

Related Information

- AN 556: Using the Design Security Features in Intel FPGAs
- Generating PR Bitstream Files
- Adding the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP

1.13.2. Clock-to-Data Ratio for Bitstream Encryption and Compression (Arria 10 or Cyclone 10 GX Designs)

The following table lists the valid combinations of bitstream encryption and compression. The Clock-to-Data (CD) ratio is defined as the number of clock cycles that each cycle of data must remain valid before the next clock cycle. For example, a CD ratio of 4 means that the data must remain valid for 4 clock cycles before the next cycle. Enhanced decompression uses the same CD ratio as plain bitstreams (that is, with both encryption and compression off). When enhanced compression is enabled, always refer to x16 data width. If you use compression and enhanced compression together, the CD ratio follows the compression bitstream - 4. If you use plain and enhanced compression together, the CD ratio follows the plain bitstream - 1.

Table 9. Valid Combinations and CD Ratio for Bitstream Encryption and Compression

Configuration Data Width	AES Encryption	Basic Compression	CD Ratio
x8	Off	Off	1
	Off	On	2
	On	Off	1
x16	Off	Off	1
	Off	On	4
	On	Off	2
x32	Off	Off	1
	Off	On	8
	On	Off	4

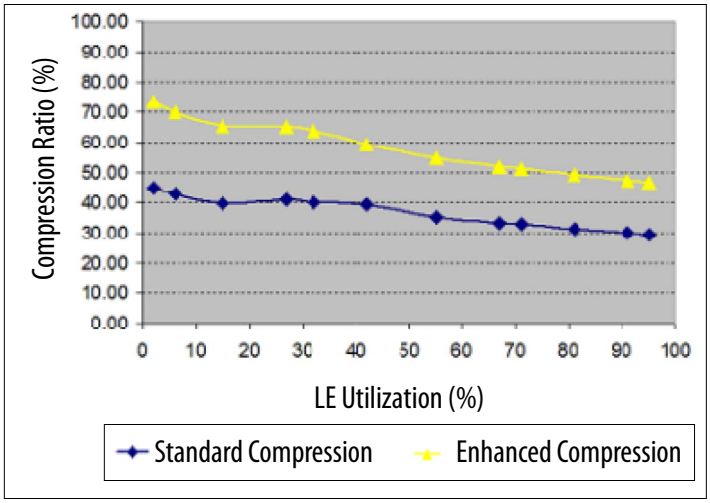
Use the exact CD ratio that the *Valid combinations and CD Ratio for Bitstream Encryption and Compression* table specifies for different bitstream types. The CD ratio for plain `.rbf` must be 1. The CD ratio for compressed `.rbf` must be 2, 4 or 8, depending on the width. Do not specify the CD ratio as the necessary minimum to support different bitstream types.

1.13.3. Data Compression Comparison

Standard compression results in a 30-45% decrease in `.rbf` size. Use of the enhanced data compression algorithm results in 55-75% decrease in `.rbf` size. The algorithm increases the compression at the expense of additional core area required to implement the compression algorithm.

The following figure shows the compression ratio comparison across PR designs with varying degrees of Logic Element (LE):

Figure 40. Compression Ratio Comparison between Standard Compression and Enhanced Compression



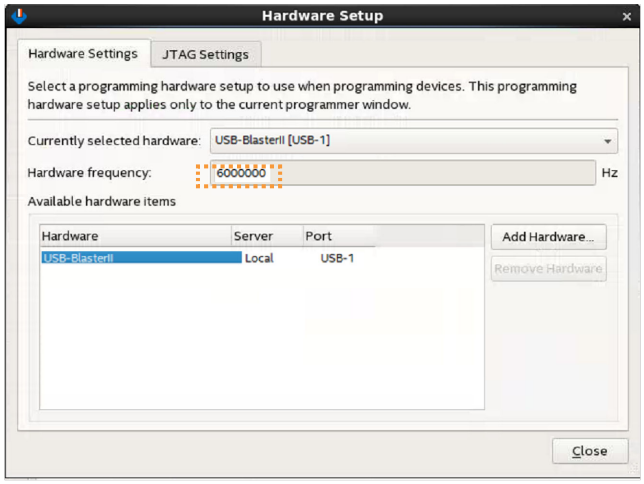
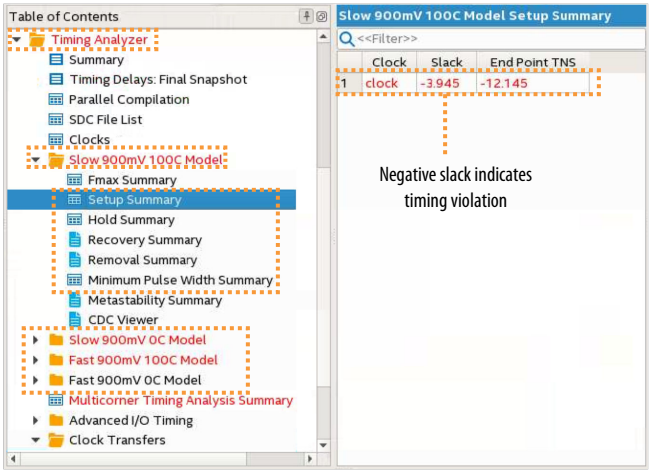
1.14. Avoiding PR Programming Errors

You can use the following guidelines to avoid or resolve common PR programming errors.

Table 10. PR Programming Guidelines

PR Programming Guideline	Description
Device in project must match device on board	Confirm the target FPGA device that you specify for the project matches the device on the development kit you target. These two devices must be the same. Click **Assignments ➤ Device** to view the target device.
Programmer versions must match	When using the Quartus Prime Programmer for PR programming, confirm that the Programmer version matches the Quartus Prime version that you use for compilation. A mismatch between the Programmer and Quartus Prime software version can occur if you compile on one machine, and then program on a different machine with a different Quartus Prime version. The software version match is especially critical for Agilex 7, Agilex 5, and Stratix 10 designs because the PR configuration hardware has dependencies inside the Programmer.
Specify a lower JTAG clock frequency	Lower the JTAG clock frequency to 6MHz: 1. In the Programmer window, click **Hardware Setup**, and then select **Intel FPGA Download Cable II** as the programming hardware. 2. For the **Hardware frequency**, specify a value from 24000000 (24MHz) to 6000000 (6MHz).

continued...

PR Programming Guideline	Description
	
Close timing for all revisions	Confirm that each project revision closes timing after design compilation: 1. In the Compilation Report, expand the **Timing Analyzer ➤ Slow 900mV 100C Model** folders, and then view the Setup Summary, Hold Summary, Recovery Summary, Removal Summary, and Minimum Pulse Width Summary reports. In each report, verify that there are no timing violations indicated by a negative **Slack** value in the report. 2. Repeat step 1 to verify timing closure in the **Slow 900mV 0C Model**, the **Fast 900mV 100C Model**, and the **Fast 900mV 0C Model**. The design closes timing when there are no negative **Slack** values for any clock in the report.  3. Repeat steps 1 and 2 for each project revision in the PR design.

Note: If an error occurs during PR operation for an Agilex 7, Agilex 5, or Stratix 10 design using SEU detection, the PR region is frozen, becomes non-functional, and SEU detection is disabled for all sectors within the PR region and certain sectors adjacent to PR region. To resolve this error and restore SEU detection on affected areas, perform a full chip configuration.

1.15. Exporting a Version-Compatible Compilation Database for PR Designs

Using version-compatible databases, you can import the base revision of a PR design to a later version of the Quartus Prime software, and then compile the PR revisions in the later version of software, without recompiling the static region.

This technique is helpful when you want to compile and generate bitstreams for the PR implementation revisions with a later version of the Quartus Prime software. Configuration bitstreams are not version-compatible, and you must generate all bitstreams from the same version of the Quartus Prime software.

After migrating the base revision to a later version of the Quartus Prime software, the bitstream you generate is only compatible with bitstreams from PR implementation compilations using that same Quartus Prime software version. Such a bitstream is incompatible with the PR bitstreams from an earlier version of the Quartus Prime software.

The Quartus Prime Pro Edition software supports version-compatible databases for PR designs for the following software versions and devices:

Table 11. Version-Compatible Compilation Database Support

The first table column indicates the first version to support version-compatible compilation database export for the specified devices.

Note:

- Database import supports two major versions back. For example, a database that you export from version 19.3, you can then import using version 19.3, 20.1, and 20.3. However, you cannot import version 19.3 to 21.1.
- You can export from any version that follows a supported version, if the version still supports the devices.

First Version with 'Export Design' Support	Stratix 10 and Devices	Arria 10 and Cyclone 10 GX Devices
18.0	No Support.	Supports all devices.
18.1	• 1SG250L • 1SG280H_S2 • 1SG280L • 1SG280L_S3 • 1SX250L • 1SX280L • 1SX280L_S3	Supports all devices.
19.1	• 1SM16BH • 1SM21BH • 1SM16CH • 1SM21CH • 1SM21KH • 1SM16KH • 1SM21LH • 1SM16LH	Supports all devices.
19.3	• 1SG10MH_U1 • 1SG10MH_U2 • 1ST250E • 1ST280E	Supports all devices.

continued...

First Version with 'Export Design' Support	Stratix 10 and Devices	Arria 10 and Cyclone 10 GX Devices
	• 1SM16E • 1SM21E • 1ST165E • 1ST210E • 1SG166H • 1SG211H	
20.1	• 1SD280P • 1ST040E • 1ST085E • 1ST110E	Supports all devices.
20.3	• 1SD21BP • 1SG040H • 1SX040H	Supports all devices.
20.4	• 1SN21BH • 1SN21CE	Supports all devices.

The following topics describe the version-compatible database generation flow and steps.

Version-Compatible Database Flow for PR Designs on page 68

Generating a Version-Compatible Compilation Database for PR Designs on page 69

Related Information

Quartus Prime Pro Edition User Guide: Getting Started
For more information on version-compatible compilation database file generation.

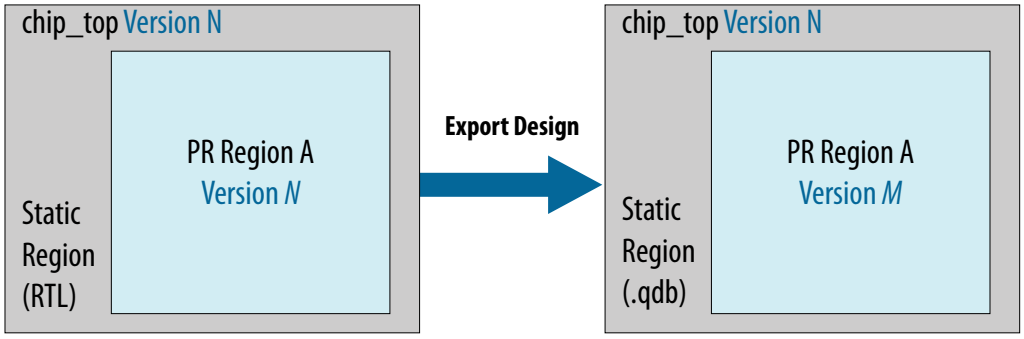
1.15.1. Version-Compatible Database Flow for PR Designs

Migrating a design with a single PR region involves the following high-level steps:

1. Perform initial compilation of the base revision in the Quartus Prime software version *N*.
2. Export a version-compatible database for the entire design in the Quartus Prime software version *N*.
3. Import the version-compatible database into the Quartus Prime software version *M (M>N)*.
4. Generate the base revision `.sof` file and bitstreams with the Quartus Prime Assembler version *M*.
5. Export the static region `.qdb` in the Quartus Prime software version *M*.
6. Perform a PR implementation compile in the Quartus Prime software version *M*.

Note: You must generate all of the PR bitstreams that you use with the Quartus Prime software version (*M*), including the full-chip configuration bitstream and the PR bitstream `.rbf`.

Figure 41. Static Region Migration (Single PR Region Compiled in Later Version)



1.15.2. Generating a Version-Compatible Compilation Database for PR Designs

Follow these steps to generate a version-compatible compilation database for PR designs:

1. Export the entire compiled design from the Quartus Prime software version *N* by clicking **Project ➤ Export Design**, or by command line:

```
quartus_cdb <project> -c <base_revision> --export_design --snapshot final \
     --file <base_revision>.qdb
```

2. Import the compiled design to the Quartus Prime software version *M* by clicking **Project ➤ Import Design**, or by command line:

```
quartus_cdb <project> -c <base_revision_import> --import_design --file \
     <base_revision>.qdb
```

   Note: Whenever possible, import the design into a different working directory than the directory that you use to compile the base design. If you must use the same directory for import and for compiling the base design, make a backup copy of your compiled design by archiving that design with `qdb/*` included, or make a copy of the entire directory and subdirectories elsewhere. You must also remove the old database directory `qdb/*` and all the bitstream related files (`*.sof`, `*.msf`, `*.pmsf`).

3. Rerun the finalize stage of the Fitter in the Quartus Prime Pro Edition software version *M* by clicking **Processing ➤ Start ➤ Start Fitter (Finalize)**, or by command line:

```
quartus_fit <project> -c <base_revision_import> --finalize
```

4. Run the Assembler in the Quartus Prime Pro Edition software version *M* to regenerate the static region bitstream by clicking **Processing ➤ Start ➤ Start Assembler**, or by command line:

```
quartus_asm <project> -c <base_revision_import>
```

5. Export the static region `.qdb` in the Quartus Prime Pro Edition software version *M* by clicking **Project ➤ Export Design Partition**, or by command line:

```
quartus_cdb <project> -c <base_revision_import> --export_block \
root_partition --snapshot final --file --include_sdc_entity_in_partition
static.qdb
```

Note: When exporting the base revision and the static partition, you must include any `.sdc` files that apply to the partition, by using the `include_sdc_entity_in_partition` option.

6. Compile each implementation revision in the Quartus Prime Pro Edition software version *M*, using the static revision `.qdb` that you exported in the previous step.

```
quartus_sh –flow compile <project> –c <impl_rev>
```

1.16. Creating a Partial Reconfiguration Design Revision History

Document Version	Quartus Prime Version	Changes
2023.04.01	24.1	• Applied initial Altera rebranding throughout. • Updated throughout to reflect support for Agilex 5 devices.
2023.07.31	23.2	• Added *Understanding PR Logic Utilization Reports* topic to describe interpretation of logic utilization values.
2023.06.30	23.2	• Updated *What's New In This Version* topic for current changes that impact this document. • Noted requirement that you must use global clock resources to clock M20K RAMs in PR regions for Agilex 7 devices in *Partial Reconfiguration Design Considerations* topic.
2023.06.26	23.2	• Updated *What's New In This Version* topic for current changes that impact this document. • Removed obsolete *Partial Reconfiguration M20K Protection Methodology for Intel Agilex 7 Devices* appendix.
2023.04.03	23.1	• Updated *What's New In This Version* topic for current changes that impact this document. • Updated product family name to "Intel Agilex 7."
2022.08.05	22.2	• Updated *What's New In This Version* topic to note speed grade, compression, and encryption limitation removal. All speed grades support PR. Compression is always enabled for Intel Agilex devices and Intel Stratix 10 devices. Encryption is supported as part of the bitstream security feature for these devices. • Added note about avoiding overlapping routing regions to *Step 3: Floorplan the Design* topic. • Removed speed grade limitations from *Partial Reconfiguration Design Considerations* topic. • Added footnote to *Partial Reconfiguration Design Guidelines* topic clarifying need to sometimes reset data registers. • Corrected figure highlight in *Viewing Row Clock Region Boundaries* topic. • Revised statement about need to install specific IP for debugging the PR region in *Debugging PR Regions with the Signal Tap Logic Analyzer* topic. • Revised *Implementing Clock Enable for On-Chip Memories with Initialized Contents* for requirement to gate appropriately the Clock Enable signal in some cases. • Removed note stating no support for PR bitstream encryption and compression for Intel Agilex devices nor Intel Stratix 10 devices. Compression is always enabled by default for Intel Agilex devices and Intel Stratix 10 devices. Encryption is supported as part of the security feature.
2022.01.11	21.4	• Add Top FAQs navigation to the cover page. • Minor wording changes to *Partial Reconfiguration Terminology* table. • Added *What's New in this Version* topic.

continued...

Document Version	Quartus Prime Version	Changes
		• Added clarifying notes to image in *Viewing Row Clock Region Boundaries* topic. • Added note about pin assignments deriving from only from the base revision to *Step 3: Floorplan the Design* and *Step 8: Setup Implementation Revisions Viewing Row Clock Region Boundaries* topic. • Added notes about the relationship between sector coverage and reconfiguration time to *Step 3: Floorplan the Design* topic. • Removed Post Synthesis Export File substep from *Step 7: Compile the Base Revision and Export the Static Region* topic. • Corrected typo and added note to *Step 8: Setup PR Implementation Revisions* topic. • Revised *Generating PR Bitstream Files* topic to refer to Convert Programming Files dialog box. • Corrected steps in *Generating a Version-Compatible Compilation Database for PR Designs* topic.
2021.10.04	21.3	• Updated *PR Bitstream Security Validation* topic to refer to Programming File Generator and revise figure. • Updated *PR File Management* topic for new QSF assignment method. • Clarified device support and corrected code example in *Generating a Merged .pmsf File from Multiple .pmsf Files* topic. • Revised *Version-Compatible Database Flow for PR Designs* topic equation and figure. • Updated non-inclusive terms with "host" and "agent" for Avalon Memory Mapped interface references throughout.
2021.08.02	21.2	• Updated *Register States and Programming Model* diagram. • Updated *Avoiding PR Programming Errors* topic SEU note.
2021.06.21	21.2	• Updated *Version-Compatible Compilation Database Support* table.
2021.05.06	21.1	• Indicated support for Agilex 7 design PR bitstream generation, and removed PR bitstream generation limitation notes. • Revised *Partial Reconfiguration Design Debugging* topic. • Revised *Debugging PR Designs with the Signal Tap Logic Analyzer* topic. • Added new *Instantiating the Intel Configuration Reset Release Endpoint to Debug Logic* topic. • Updated *Partial Reconfiguration Security* topic for latest information. • Updated *PR Bitstream Security Validation* topic license statement and to remove outdated references. • Revised *PR Authentication* topic use cases. • Revised *PR Authentication* topic use cases. • Revised *PR Bitstream Encryption* topic use case and prerequisites.
2020.12.11	20.3	• Corrected typo in PMSF file definition in "Using PR Bitstream Security Verification"
2020.09.28	20.3	• Added note about preserving SDC files to "Step 8: Setup PR Implementation Revisions" topic. • Added new "Using Parent QDB Files from Different Compiles" topic. • Added "PR Design Timing Closure Best Practices" topic. • Replaced references to Avalon-MM and Avalon-ST with Avalon memory-mapped and Avalon streaming for legal compliance.

continued...

Document Version	Quartus Prime Version	Changes
2020.08.07	20.2	• Added screenshot and details about Synthesis report to "Evaluating PR Region Initial Conditions" topic. • Added details about Include entity-bound SDC files option requirements to "Step 7: Compile the Base Revision and Export the Static Region" topic. • Removed stated support for "PR Bitstream Security Verification" for Agilex 7 devices. This feature is not yet supported for Agilex 7 devices. • Added details about Include entity-bound SDC files option requirements to "Generating a Version-Compatible Compilation Database" for PR Designs topic. • Added reference to "EDA Netlist Writer and Gate Level-Netlists" section of the *Quartus Prime Pro Edition User Guide: Third Party Simulation* to the "Partial Reconfiguration Design Simulation" section.
2020.06.22	20.2	• Added PR simulation support for Agilex 7 designs.
2020.05.11	20.1	• Revised description of PR bitstream compatibility checking steps for Cyclone 10 GX and Arria 10 devices.
2020.04.13	20.1	• Updated requirements in "Partial Reconfiguration Bitstream Compatibility Checking" topic. • Added note about time borrowing to "Partial Reconfiguration Design Timing Analysis" topic. • Added note indicating HPR designs do not support PR security bitstream verification.
2019.11.18	19.3.0	• Changed title of "Migrating PR Regions to a Later Software Version" to "Exporting a Version-Compatible Compilation Database for a PR Design," generalized examples, and removed INI requirement.
2019.09.30	19.3.0	• Added compilation support for Cyclone 10 GX and Agilex 7 PR designs. • Updated IP name from "Partial Reconfiguration Controller Intel Stratix 10 FPGA IP" to "Partial Reconfiguration Controller Intel FPGA IP" to encompass Agilex 7 designs. • Updated wording of "Clock Gating" topic for clarity. • Added note to Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP "Parameters" topic about support for enhanced decompression.
2019.06.10	19.1.0	• Added details about synthesis of PRESERVE_FANOUT_FREE_NODE to "Partial Reconfiguration Design Guidelines."
2019.04.22	19.1.0	• Indicated support for POF generation support for Intel Cyclone GX devices. • Corrected code example in "PR Migration Flow" topic.
2019.04.01	19.1.0	• Described migration of the static region of a PR design to a later version of the Quartus Prime software. • Described new "PR Bitstream Security Validation" feature. • Described new location of auto export from output_files to project directory.
2018.12.30	18.1.1	• Described "Partial Reconfiguration Bitstream Compatibility Check" and PR region limitations.
2018.10.24	18.1.0	• Added "PR File Management" topic. • Updated first guideline in "Partial Reconfiguration Design Guidelines."
2018.09.24	18.1.0	• Described automated `.qdb` partition export in "Exporting a Design Partition." • Added details about required assignments to "Step 6: Create Revisions for Personas." • Removed references to placed snapshot. Only synthesized and final snapshots are supported.

continued...

Document Version	Quartus Prime Version	Changes
		• Corrected description of Entity Re-binding option in *Design Partition Settings* table. • Added command line instructions for creating a revision. • Stated PR compilation flow support for Cyclone 10 GX devices. • Updated Partial Reconfiguration Controller Arria 10 FPGA IP name to Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP. • Added "Viewing Row Clock Region Boundaries." • Added "Planning Clocks and other Global Routing."
2018.07.18	18.0.0	• Corrected signals in *Simulation of PR Persona Switching* diagram.
2018.06.18	18.0.0	• Corrected syntax errors and added note in *Running Timing Analysis on Aggregate Revisions*.
2018.05.29	18.0.0	• Added description of "|" that identifies the root partition hierarchy path in Design Partitions Window. • Clarified .qsf assignment in *Running Timing Analysis on Aggregate Revisions*.
2018.05.07	18.0.0	• Added description of new Partial Reconfiguration External Configuration Controller Stratix 10 FPGA IP. • Removed descriptions of obsolete synthesis-only revisions and corresponding personas. Replaced with latest simplified flow instructions. • Updated names of Partial Reconfiguration Controller Arria 10 FPGA IP and Partial Reconfiguration Controller Stratix 10 FPGA IP. • Added *Design Partition Settings* topic. • Added *Evaluating PR Partition Initial Conditions* topic. • Added *Avoiding PR Programming Errors* topic. • Described qcrypt incompatibility with Enable bitstream compatibility check and workaround. • Added as chapter in new *Partial Reconfiguration User Guide*. • Updated command-line syntax in *Running Timing Analysis on Aggregate Revisions* topic. • Removed obsolete HPR flow script information and linked to *AN826: Hierarchical Partial Reconfiguration Tutorial for Intel Stratix 10 GX FPGA Development Board* • Added note about recovery after PR error when using SEU detection in Stratix 10 designs.
2017.11.06	17.1.0	• Added partial reconfiguration support for Stratix 10 devices. • Added descriptions of Stratix 10 Partial Reconfiguration Controller IP, SUPR, HPR, and SDM to terms list. • Updated for latest Intel branding and software user interface.
2017.05.08	17.0.0	• Added information about Hierarchical Partial Reconfiguration. • Added new topic Partial Reconfiguration Simulation and Verification. • Added new topic 'Run Timing Analysis on a Design with Multiple PR Partitions'. • Updated Freeze Logic for PR Regions. • Added new topic Debugging Using Signal Tap Logic Analyzer. • Other minor updates.
10.31.2016	16.1.0	• Initial release.

2. Partial Reconfiguration Solutions IP User Guide

The Quartus Prime Pro Edition software includes the following Intel FPGA IP cores that simplify partial reconfiguration implementation.

Instantiate one or more of these IP cores to implement handshake and freeze logic for PR functionality in your design. Alternatively, create your own PR handshake and freeze logic that interfaces with the PR region.

Table 12. Partial Reconfiguration IP Cores

Intel FPGA IP	Description	Usage
Partial Reconfiguration Controller Intel FPGA IP	Dedicated IP component that sends the partial reconfiguration bitstream for the Agilex 7, Agilex 5, or Stratix 10 FPGA. The PR bitstream performs reconfiguration by adjusting CRAM bits in the FPGA.	One instance per Stratix 10, Agilex 5, or Agilex 7 FPGA
Partial Reconfiguration External Configuration Controller Intel FPGA IP	IP component that supports Stratix 10 and Agilex 7 FPGA partial reconfiguration via an external source over dedicated PR pins.	One instance per Stratix 10, Agilex 5, or Agilex 7 FPGA for external configuration
Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP	Dedicated IP component that sends the partial reconfiguration bitstream to the Arria 10 or Cyclone 10 GX FPGA. The PR bitstream performs reconfiguration by adjusting CRAM bits in the FPGA.	One instance per Arria 10 or Cyclone 10 GX FPGA, internal or external configuration.
Partial Reconfiguration Region Controller Intel FPGA IP	Provides a standard Avalon memory-mapped interface to the block that controls handshaking with the PR region. Ensures that PR region stops, resets, and restarts, according to the PR handshake.	One instance per PR region.
Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge Intel FPGA IP	Provides freeze capabilities to the PR region for Avalon memory-mapped interfaces.	One instance for each interface in each PR region.
Avalon Streaming Partial Reconfiguration Freeze Bridge Intel FPGA IP	Provides freeze capabilities to the PR region for Avalon streaming interfaces.	One instance for each interface in each PR region.

2.1. Internal and External PR Host Configurations

You perform PR with either an internal host residing in the core resources, or with an external host via dedicated device pins. Use of an internal host stores all PR host logic on the FPGA device, rather than on an external device. The PR host interfaces with the control block through simple handshaking and data transfer.

Figure 42. Arria 10 and Cyclone 10 GX Partial Reconfiguration IP Components (Internal Host)

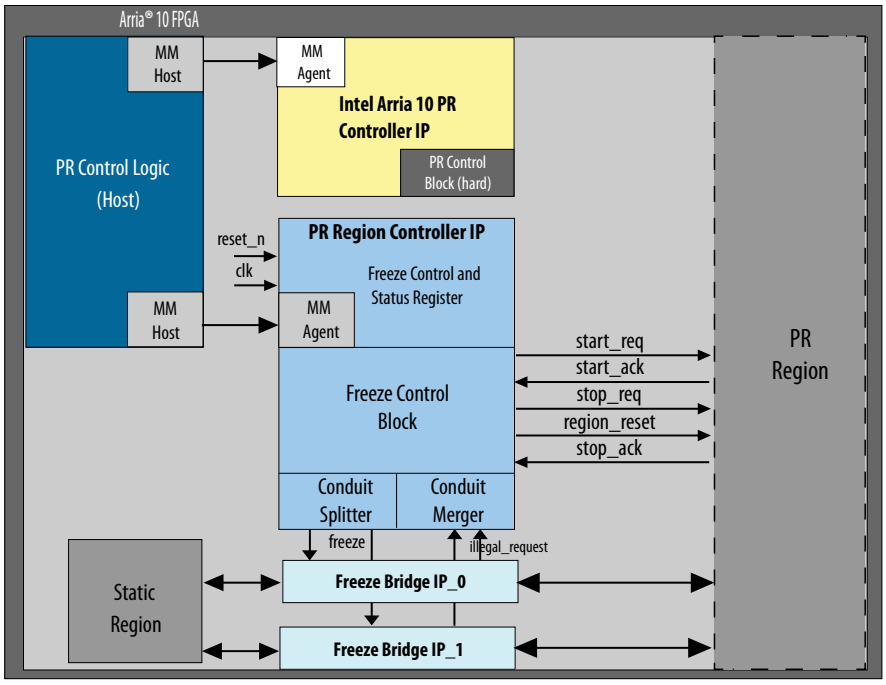


Figure 43. Agilex 7, Agilex 5, and Stratix 10 Partial Reconfiguration IP Components for (Internal Host)

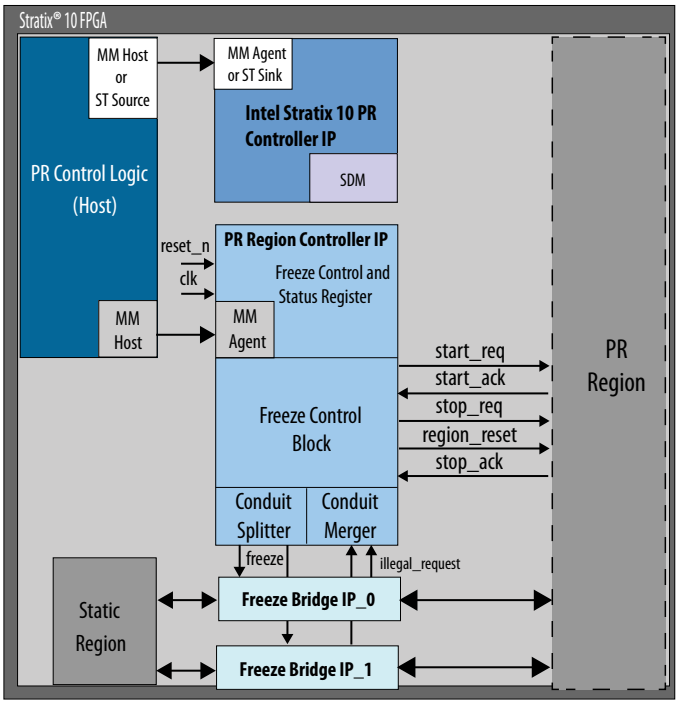


Figure 44. Partial Reconfiguration with Microcontroller External Host (Arria 10 or Cyclone 10 GX device)

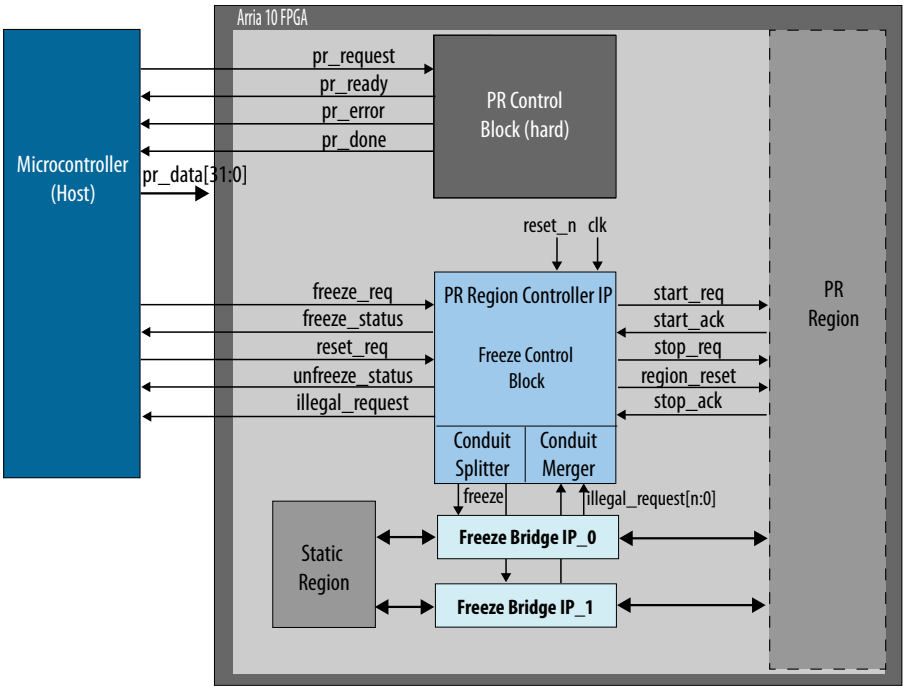
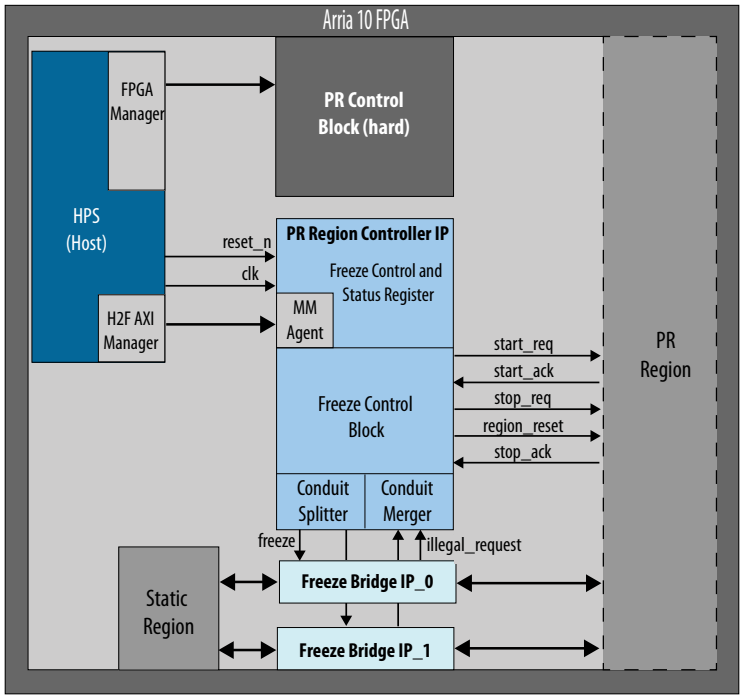


Figure 45. Partial Reconfiguration with HPS Internal Host (Arria 10 or Cyclone 10 GX device)

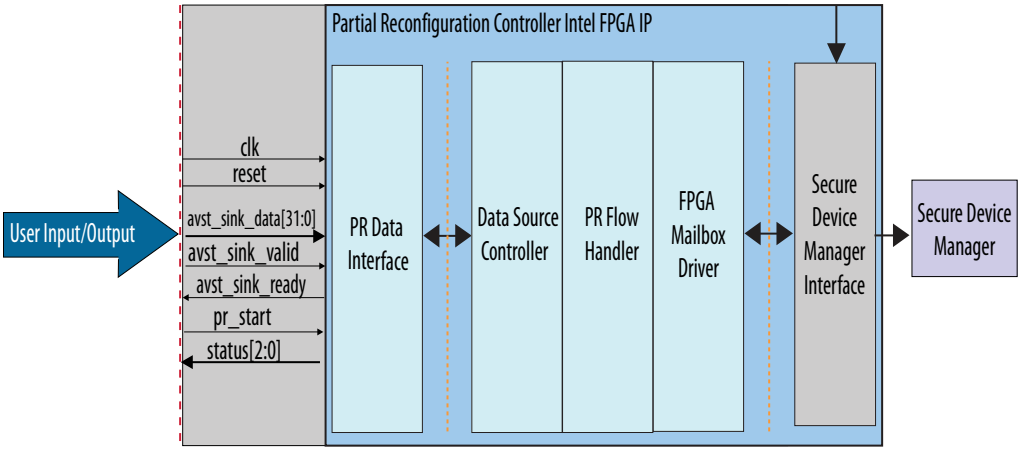


2.2. Partial Reconfiguration Controller Intel FPGA IP

The Partial Reconfiguration Controller Intel FPGA IP provides partial reconfiguration functionality for Stratix 10 and Agilex 7 designs. The IP core provides a standard interface to the FPGA secure device manager (SDM), and has a maximum clock frequency of 200 MHz.

Figure 46. Partial Reconfiguration Controller Avalon Streaming Interface (Agilex 7, Agilex 5, and Stratix 10 Designs)

(5)



Note: If an error occurs during PR operation for an Agilex 7, Agilex 5, or Stratix 10 design using SEU detection, the PR region is frozen, becomes non-functional, and SEU detection is disabled for all sectors within the PR region and certain sectors adjacent to PR region. To resolve this error and restore SEU detection on affected areas, perform a full chip configuration.

2.2.1. Memory Map

The Partial Reconfiguration Controller Intel FPGA IP has the following memory map.

Table 13. Avalon Memory-Mapped Slave Memory Map

Name	Address Offset	Width	Access	Description
PR_DATA	0x00	32	Write	Every data `write` to this address indicates this bitstream is sending to the IP core. Width is set by the **Input data width** parameter.
PR_CSR	0x01	32	Read or Write	Control and status registers with the following offset bits:

continued...

(5) Avalon memory mapped interface variant also available.

Name	Address Offset	Width	Access	Description
				• 31 - 7: Reserved. • 6: Protocol violation. This bit is asserted when the Avalon memory-mapped or Avalon streaming protocol is violated. • 5: Read/Write for `irq` signal mask bit. Write `1` to this bit enable `irq` signal and `0` to disable the `irq` signal. • 4: Read/Clear for `irq` signal. The `irq` signal asserts if an error occurs. The Master must read the status signal and clear the interrupt by writing `1` to this bit. • 3 - 1: Read-only for `status` signal. • 0: Read/Write for `pr_start` signal. To streamline the flow, the IP core automatically de-asserts to value `0`, one clock cycle after the signal asserts.
PR_SW_VER	0x02	32	Read	Read-only SW version register. Register is currently 0xBA500000.
PR_FW_HANDSHAKE	0x03	32	Read	Current location of mailbox handshake between the PR IP and the SDM in the PR operation with the following offset bits: • 31 - 8: Reserved. • 7 - 0: Current location of the mailbox handshake between the PR IP and the SDM.
PR_FW_RESPONSE	0x04	32	Read	SDM mailbox response. You must use this in conjunction with PR_FW_HANDSHAKE. If PR_FW_HANDSHAKE is `0x2` or `0x6`, the following offset bits apply: • 31 - 11: Reserved. • 10 - 0: Response header of the response payload. If PR_FW_HANDSHAKE is 0x4, the following offset bit applies: • 31 - 0: First response word of response payload.

Note: For IP core instantiation guidelines, refer to the appropriate device configuration user guide.

Related Information

- Stratix 10 Configuration User Guide
- Agilex 7 Configuration User Guide

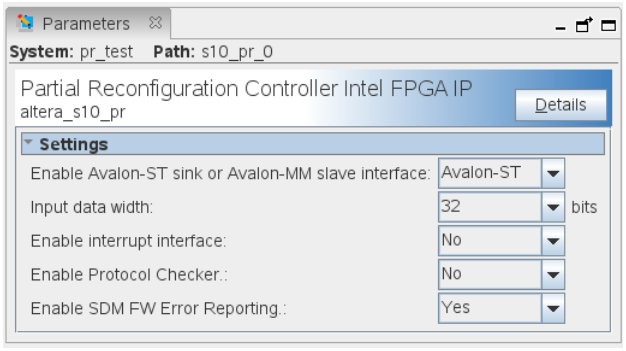
2.2.2. Parameters

The Partial Reconfiguration Controller Intel FPGA IP supports customization of the following parameters.

Table 14. Partial Reconfiguration Controller Intel FPGA IP Parameter Settings

Parameter	Value	Description
Enable Avalon-ST sink or Avalon-MM slave interface	**Avalon-ST/Avalon-MM**	Enables the controller's Avalon streaming sink or Avalon memory-mapped agent interface.
Input data width	*<bits>*	Specifies the size of the controller's data conduit interface in bits. The IP supports device widths of 32 and 64. The Avalon memory-mapped slave interface supports 32-bits only.
Enable interrupt interface	**Yes/No**	Enables interrupt assertion for detection of incompatible bitstream, CRC_ERROR, PR_ERROR, or successful partial reconfiguration. Upon interrupt, query PR_CSR[3:1] for status. Write a 1 to PR_CSR[4] to clear the interrupt. Use only together with the Avalon memory-mapped agent interface.
Enable Protocol Checker	**Yes/No**	Reads out the error bit from the CSR register (PR_CSR[6]).
Enable SDM FW Error Reporting	**Yes/No**	Enables the SDM firmware error reporting ports and CSR. Enables the additional pr_fw_handshake and pr_fw_response ports in Avalon streaming mode, and can be read out from the CSR register (base address offsets 3 and 4) in Avalon memory-mapped mode. *Note:* This parameter is only supported for Agilex 7 and Agilex 5 devices.

Figure 47. Parameter Editor



2.2.3. Ports

The Partial Reconfiguration Controller Intel FPGA IP includes the following interface ports.

Figure 48. Avalon Streaming Sink Interface Ports

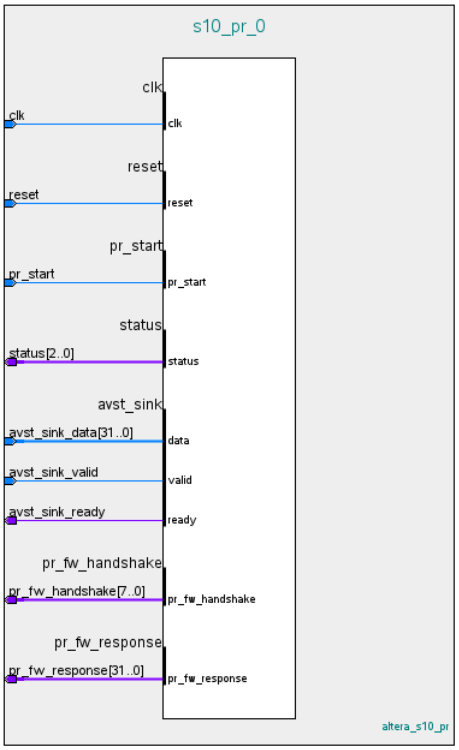
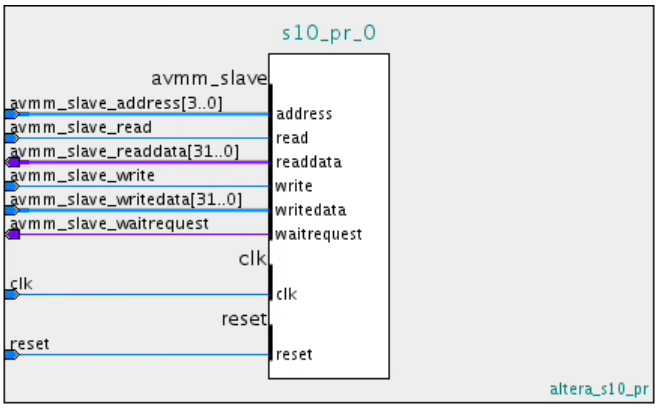


Figure 49. Avalon Memory-Mapped Agent Interface Ports

(6)



(6) The terms host and agent now replace non-inclusive terms in the Avalon Memory Mapped specification.

Table 15. Clock/Reset Ports

Port Name	Width	Direction	Function
reset	1	Input	Asynchronous reset for the PR Controller IP core. Resetting the PR Controller IP core during a partial reconfiguration operation can cause the device to lock up.
clk	1	Input	Input clock to the PR Controller IP core. The input clock must be free-running. The IP core has a maximum clock frequency of 200 MHz.

Table 16. Avalon Streaming Sink interface Ports

These ports are available when you enable the **Avalon Streaming** sink interface.

Port Name	Width	Direction	Function
pr_start	1	Input	A signal arriving at this port asserted high initiates a PR event. You must assert this signal high for a minimum of one clock cycle, and de-assert it low, prior to the end of the PR operation.
avst_sink_data[]	32|64	Input	Avalon streaming data signal that is synchronous with the rising edge of the clk signal. The **Input data width** parameter specifies this port width.
avst_sink_valid	1	Input	Avalon streaming data valid signal that indicates the avst_sink_data port contains valid data.
avst_sink_ready	1	Output	Avalon streaming ready signal that indicates the device is ready to read the streaming data on the avst_sink_data port whenever the avst_sink_valid signal asserts high. Stop sending valid data when this port is low.
status[2:0]	3	Output	A 3-bit error output that indicates the status of a PR event. Once the outputs latch high as follow, you can only reset the outputs at the beginning of the next PR event: 3'b000 – power-up nreset asserted 3'b001 – configuration system is busy 3'b010 – PR operation is in progress 3'b011 – PR operation successful 3'b100 – PR_ERROR is triggered 3'b101 – Reserved 3'b110 – Incompatible bitstream error 3'b111 – Reserved
protocol_error	1	Output	Reads out the error bit from the CSR register.
pr_fw_handshake	8	Output	Indicates the current state of the mailbox handshake between the PR IP and the SDM firmware in the PR operation.
pr_fw_response	32	Output	SDM firmware mailbox response.

Table 17. Avalon Memory-Mapped Agent Interface Ports

These ports are available when you enable the **Avalon memory-mapped** agent interface.

Port Name	Width	Direction	Function
avmm_slave_address	4	Input	Avalon memory-mapped address bus in the unit of Word addressing.
avmm_slave_read	1	Input	Avalon memory-mapped read control.

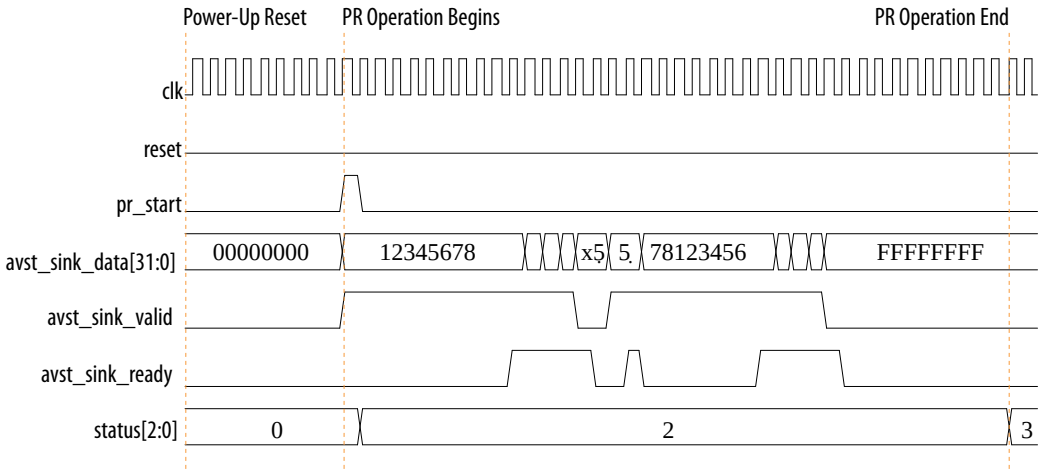
continued...

Port Name	Width	Direction	Function
avmm_slave_readdata	32	Output	Avalon memory-mapped read data bus.
avmm_slave_write	1	Input	Avalon memory-mapped write control.
avmm_slave_writedata	32	Input	Avalon memory-mapped write data bus.
avmm_slave_waitrequest	1	Output	Upon assertion, indicates that the IP is busy and the IP is unable to respond to a read or write request.
irq	1	Output	Interrupt signal when you enable the **Enable interrupt interface** parameter.

2.2.4. Timing Specifications

The following timing diagram illustrates a successful PR operation with the Partial Reconfiguration Controller Intel FPGA IP. The `status[2:0]` output signal indicates whether the operation passes or fails. The PR operation initiates upon assertion of the `pr_start` signal. Monitor the `status[]` signal to detect the end of the PR operation.

Figure 50. Timing Specifications



2.2.5. PR Error Recovery

The Partial Reconfiguration Controller Intel FPGA IP supports error recovery during partial reconfiguration.

Note: PR error recovery is only supported for Agilex 7 and Agilex 5 devices.

When PR_ERROR triggers, the PR Controller IP initiates the error recovery mechanism by de-asserting the `avst_sink_ready` signal to flush out any remaining corrupted PR bitstream that remains in the Avalon streaming pipeline.

Note: You must ensure that the remaining corrupted PR bitstream is fully flushed from the Avalon streaming pipeline before initiating another PR operation, or before performing a PR controller IP reset.

For PR Controller IP designs that have an Avalon memory-mapped interface, when PR_ERROR is triggered, continue to write (by asserting both `avmm_slave_write` and `avmm_slave_writedata`) until the PR bitstream in the Avalon memory-mapped host depletes.

To prevent inadvertent flushing of the new bitstream, do not provide a new, uncorrupted PR bitstream to the PR Controller IP until flushing is complete. Once flushing is complete, you can send a new, uncorrupted PR bitstream to the PR controller IP when `pr_start` is asserted and after the reset is de-asserted.

Note: Once the PR process initiates, you cannot alter nor replace the PR bitstream provided in the Avalon streaming pipeline. For example, if the PR Controller IP reflects a status of configuration is busy (`3'b001`), you must re-initiate PR with the PR bitstream that is already present, without replacing the bitstream. The existing PR bitstream must undergo the whole PR process until the PR operation completes with status of either success (`3'b011`) or fail (PR_ERROR is triggered: `3'b100`, or incompatible bitstream error: `3'b110`) with the error recovery mechanism to clear the Avalon streaming pipeline.

After one of the statuses occurs, you can send the PR bitstream to the PR Controller IP when initiating another PR operation, or after performing a PR Controller IP reset.

This PR error recovery feature is available only for the Partial Reconfiguration Controller Intel FPGA IP, which controls Avalon streaming and Avalon memory-mapped paths where the data comes in.

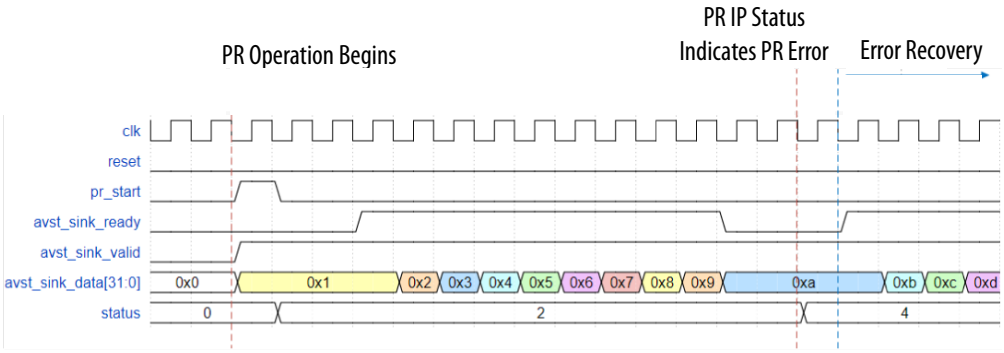
The Partial Reconfiguration Controller Intel FPGA IP only acts as a block that reports the handshaking of the external user host and the SDM of the FPGA. There is no interaction between the streaming path and the IP. You connect directly to the Avalon streaming pins in the SDM I/O. These SDM I/O pins are exactly the same as the Avalon Streaming pins that you use for full device configuration. The PR IP cannot control the SDM I/O pins.

2.2.5.1. PR Error Recovery Timing Specifications

The following describes the timing specifications for PR error recovery.

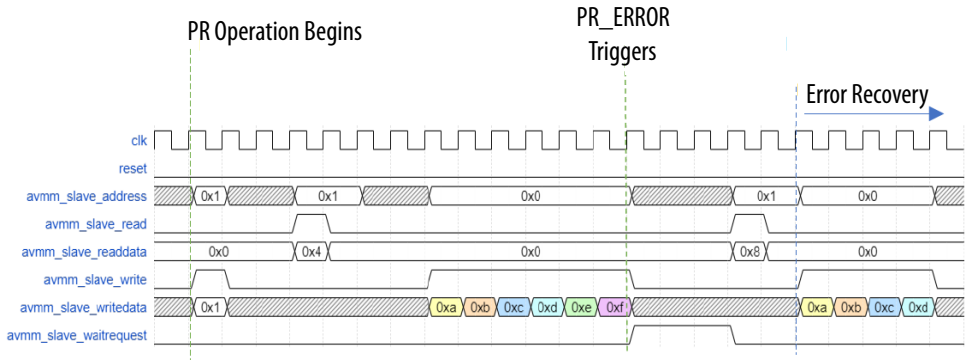
Timing Diagram: PR Operation with PR_ERROR Triggered - PR Controller Intel FPGA IP (Avalon Streaming), and Timing Diagram: PR Operation with PR_ERROR Triggered - PR Controller Intel FPGA IP (Avalon Memory-Mapped) describe a PR operation that encounters PR_ERROR. When PR_ERROR is triggered, the FPGA SDM operation de-asserts the `avst_sink_ready` signal to backpressure any remaining corrupted PR bitstream. Next, the error recovery mechanism initiates by re-asserting the `avst_sink_ready` signal to flush out any remaining corrupted PR bitstream in the Avalon streaming pipeline.

Figure 51. Timing Diagram: PR Operation with PR_ERROR Triggered - PR Controller Intel FPGA IP (Avalon Streaming)



For PR Controller IP designs that have the Avalon memory-mapped interface, when PR_ERROR triggers, continue to write (by asserting both `avmm_slave_write` and `avmm_slave_writedata`) until the PR bitstream in the Avalon memory-mapped host depletes, as Timing Diagram: PR Operation with PR_ERROR Triggered - PR Controller Intel FPGA IP (Avalon Memory-Mapped) shows.

Figure 52. Timing Diagram: PR Operation with PR_ERROR Triggered - PR Controller Intel FPGA IP (Avalon Memory-Mapped)



The error recovery mechanism continues until you initiate a PR Controller IP reset or another PR operation, as the following illustrations show. Then, you can send a new, uncorrupted PR bitstream to the PR Controller IP.

Note: Do not provide the new, uncorrupted PR bitstream to the PR Controller IP before reset or `pr_start` to prevent flushing the new bitstream during error recovery.

Figure 53. Timing Diagram: Error Recovery Until Reset Assert - PR Controller Intel FPGA IP (Avalon Streaming)

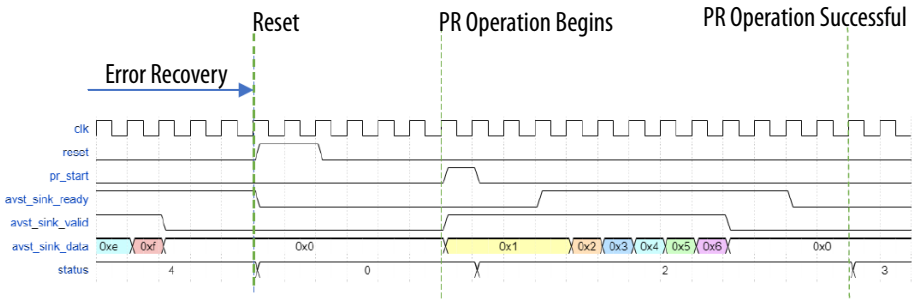


Figure 54. Timing Diagram: Error Recovery Until Reset Assert - PR Controller Intel FPGA IP (Avalon Memory-Mapped)

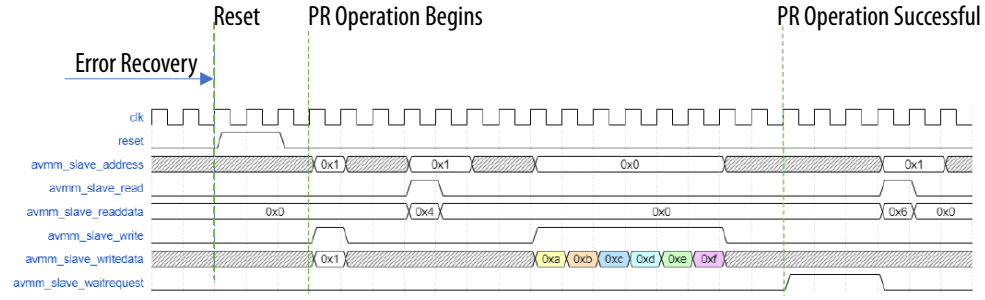
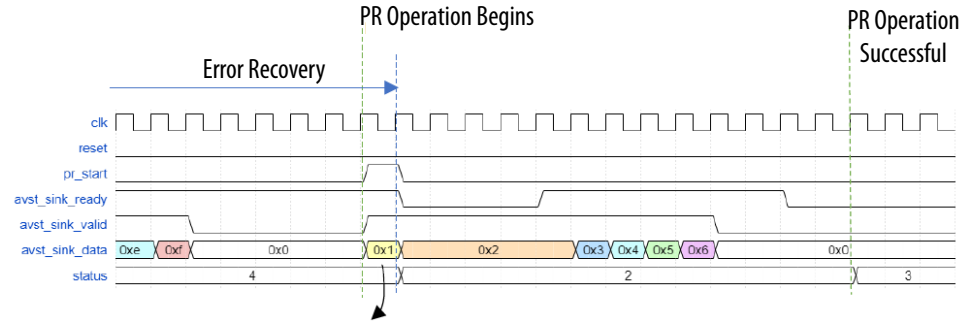
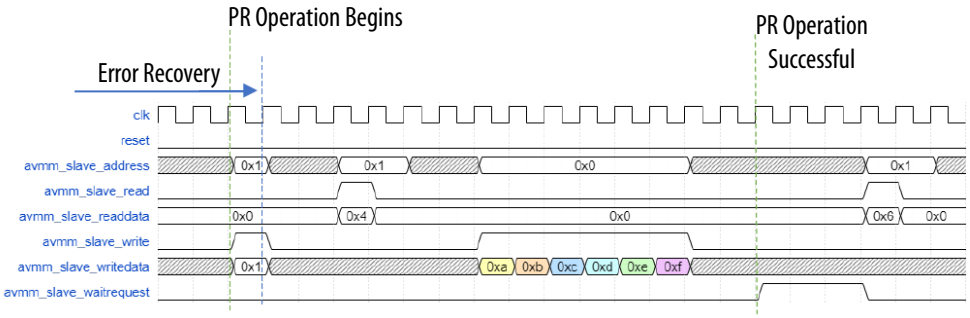


Figure 55. Timing Diagram: Error Recovery Until PR Operation Initiates - PR Controller Intel FPGA IP (Avalon Streaming)



Note: In a scenario where you provide a new, uncorrupted PR bitstream to `avst_sink_data` at the same clock cycle that PR operation initiates, the internal register captures the first beat of data (`0x1`) because of error recovery.

Figure 56. Timing Diagram: Error Recovery Until PR Operation Initiates - PR Controller Intel FPGA IP (Avalon Memory-Mapped)



2.2.6. Secure Device Manager Firmware Error Reporting

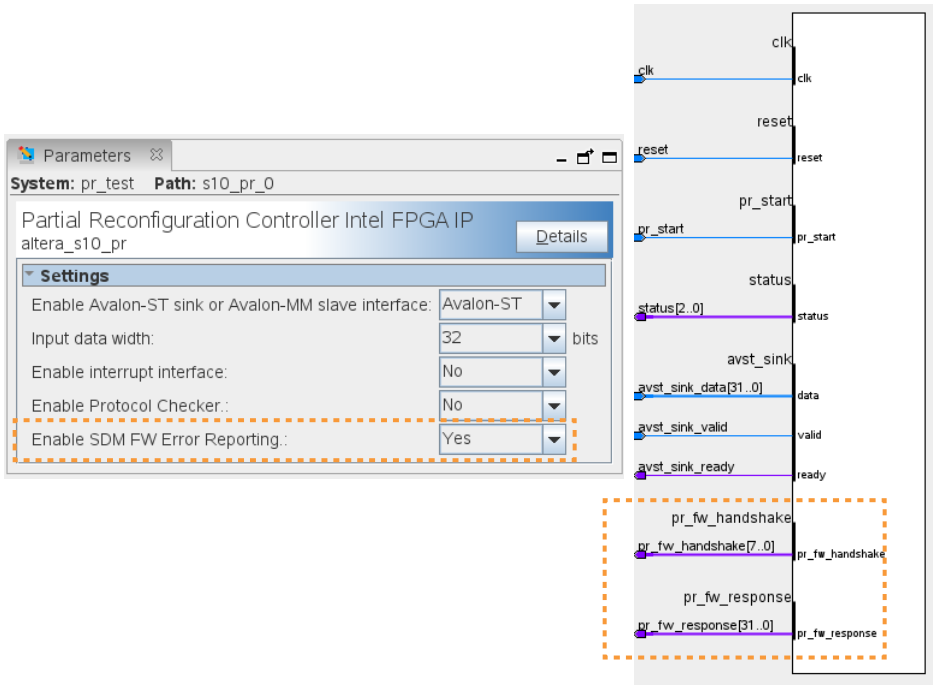
The Partial Reconfiguration Controller Intel FPGA IP notifies you of a non-specific error in PR operation by updating the 3-bit `status` CSR port to a generic PR_ERROR or "Incompatible bitstream error" in the PR controller IP. However, this error notification does not include specific details about the nature of the error.

As the secure device manager (SDM) firmware processes the PR bitstream, the SDM firmware error reporting can provide details about the error, such as the PR stage where the error occurs. You can use this information to identify and resolve any issue before re-attempting another PR operation on the PR region that remains isolated from an initial, failed PR attempt. If the SDM firmware reports no error, then the PR error is outside of the SDM firmware.

Note: SDM firmware error reporting is only supported for Agilex 7 and Agilex 5 devices.

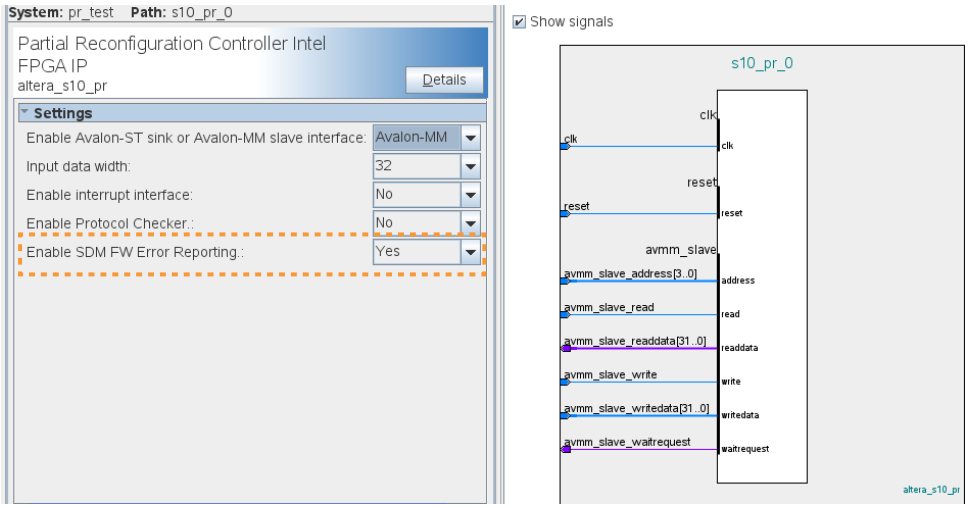
PR IP in Avalon Streaming Mode with SDM Firmware Error Reporting Enabled shows the IP in Avalon streaming mode with SDM firmware error reporting enabled. Enabling this parameter adds the `pr_fw_handshake[7:0]` and `pr_fw_response[31:0]` ports to the IP.

Figure 57. PR IP in Avalon Streaming Mode with SDM Firmware Error Reporting Enabled



Similarly, PR IP in Avalon Memory-Mapped Mode with SDM Firmware Error Reporting Enabled shows the IP in Avalon memory-mapped mode with SDM firmware error reporting enabled by the PR_FW_HANDSHAKE[0x3] and PR_FW_RESPONSE[0x4] CSR registers of the Avalon memory-mapped register map. Enabling this parameter adds no additional ports.

Figure 58. PR IP in Avalon Memory-Mapped Mode with SDM Firmware Error Reporting Enabled

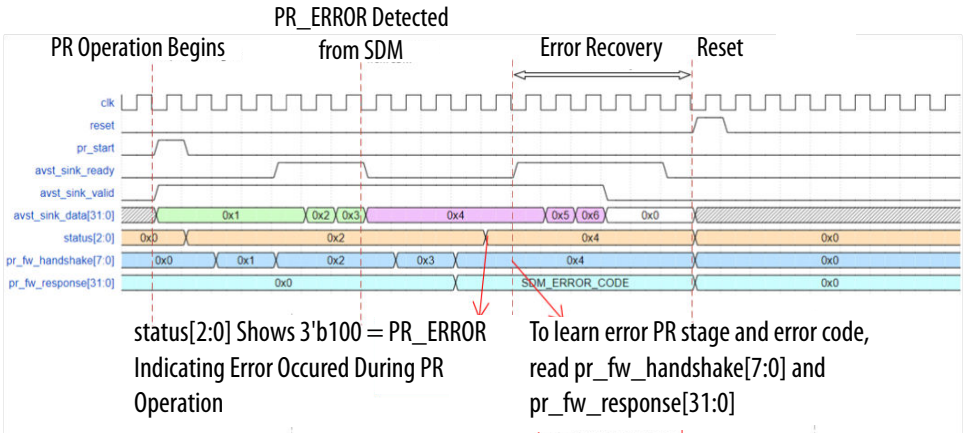


2.2.6.1. SDM Firmware Error Reporting Timing Specification

Timing Diagram: PR Operation with SDM Firmware Error Reporting Signals - PR Controller Intel FPGA IP (Avalon Streaming) shows the timing diagram of PR operation with the SDM firmware error reporting signal usage in Avalon-ST mode of the PR Controller IP. During PR operation, you can read the `pr_fw_handshake` and `pr_fw_response` signals after a PR error to learn more about the error. For example, once you detect the PR operation has ended with the `status[2:0]`, you can continue by reading the `pr_fw_handshake` and `pr_fw_response` signals.

In this example, the PR operation ends with PR_ERROR, as indicated by the `status[2:0] = 3'b100`. For error details, you can read the `pr_fw_handshake` to determine in which stage the PR operation fails. You can read `pr_fw_response` to determine the error code that the SDM returns.

Figure 59. Timing Diagram: PR Operation with SDM Firmware Error Reporting Signals - PR Controller Intel FPGA IP (Avalon Streaming)



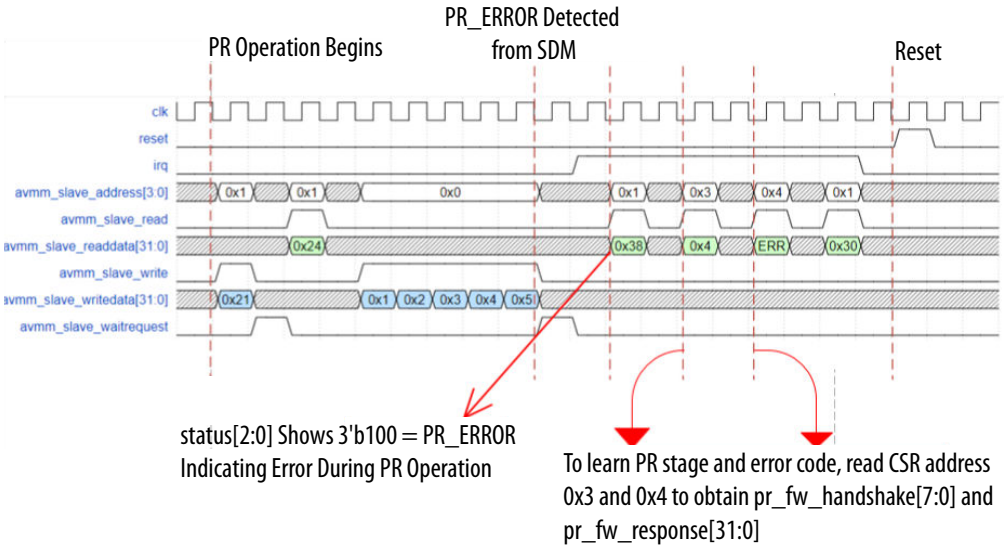
The following steps correspond to the sequence in the timing diagram:

1. Assert `pr_start` to begin the PR operation.
2. Stream the PR bitstream on `avst_sink_ready` and `avst_sink_valid` ports.
3. The SDM detects a PR_ERROR causing `avst_sink_ready` to be de-asserted to backpressure incoming PR bitstream.
4. You detect that `status[2:0]` is updated to `0x4 (3'b100)` to indicate that 'PR_ERROR is triggered'. From this status information, you can determine that an error occurred during the PR operation.
5. To learn more about the PR operation failure, read the `pr_fw_handshake[7:0]` and `pr_fw_response[31:0]` signals to determine the error code and PR stage of the failure.

Timing Diagram: PR Operation with SDM Firmware Error Reporting Signals - PR Controller Intel FPGA IP (Avalon Memory-Mapped) shows the PR operation timing diagram with the SDM firmware error reporting signal usage in Avalon memory-mapped mode of the PR Controller Intel FPGA IP. During a PR operation, you can read the CSR registers (`0x3` for PR_FW_HANDSHAKE, and `0x4` for PR_FW_RESPONSE) after a PR error to learn more about the error. For example, once you detect that the PR

operation ends with an error by IRQ assertion, and after reading CSR register 0x1 (PR_CSR) that holds the PR status[2:0] value, you can continue by reading the CSR registers 0x3 and 0x4, as .

Figure 60. Timing Diagram: PR Operation with SDM Firmware Error Reporting Signals - PR Controller Intel FPGA IP (Avalon Memory-Mapped)



The following steps correspond to the sequence in the timing diagram:

1. Start the PR operation by writing to CSR address 0x1 (PR_CSR) with 0x21.This action enables the use of the irq signal and sets the pr_start to begin the PR operation. After 1 clock cycle, status[2:0] updates to 0x2 (3'b010) to indicate 'PR operation is in progress'.
2. Read the CSR address 0x1 (PR_CSR) to obtain the status[2:0] of the PR operation. The value read back is 0x24, which translates to status[2:0] = 3'b010, indicating 'PR operation is in progress'.
3. Start writing the PR bitstream in CSR address 0x0 (PR_DATA).
4. You read PR_CSR and observe that irq was asserted and that PR_ERROR is the status.

The pr_fw_handshake register holds the current location of the mailbox handshake between the PR IP and the SDM. If the PR IP is interrupted during PR, the pr_fw_handshake register indicates the last position of the mailbox handshake. The pr_fw_handshake register also enables you to associate the value held in pr_fw_response to a specific mailbox command.

Table 18. pr_fw_handshake Register Values

Register Value	Description
0	The PR IP has not yet sent any mailbox command to the SDM to initiate the partial reconfiguration operation.
1	The PR IP has sent the mailbox command to the SDM to initiate the partial reconfiguration operation and is awaiting a response.

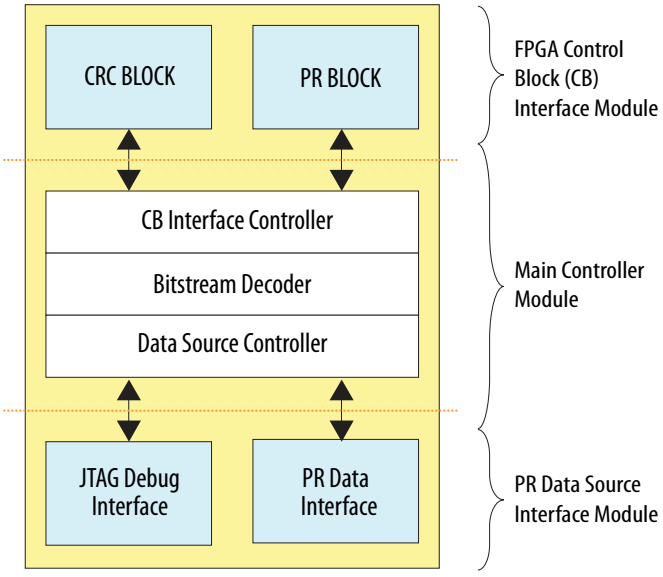
continued...

Register Value	Description
2	PR IP has received the mailbox response from the SDM on the partial reconfiguration operation initiation.
3	The PR IP has sent the mailbox command to the SDM to query the last partial reconfiguration status and is awaiting a response.
4	The PR IP has received the mailbox response from the SDM on the last partial reconfiguration status.
5	The PR IP has sent the mailbox command to the SDM to initiate the partial reconfiguration operation from the data source selected by MSEL and is awaiting a response.
6	The PR IP has received the mailbox response from the SDM on the partial reconfiguration operation from the data source selected by MSEL.
7	Reserved.

2.3. Partial Reconfiguration Controller Intel Arria® 10/Cyclone® 10 FPGA IP

The Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP provides a standard interface to the partial reconfiguration functionality in the PR control block. Use this IP core to avoid manually instantiating a PR control block interface. The Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP supports Arria 10 and Cyclone 10 GX PR designs with a maximum clock frequency of 100MHz.

Figure 61. Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP



2.3.1. Agent Interface

The Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP provides an Avalon memory-mapped agent interface to read and write to PR configuration registers.

Table 19. Data/CSR Memory Map Format

Name	Address Offset	Access	Description
PR_DATA	0x00	Write	Every data `write` to this address indicates this bitstream is sent to the IP core. Performing a `read` on this address returns all 0's.
PR_CSR	0x01	Read or Write	Control and status registers.
`Version Register`	0x02	Read-Only	Read-only SW version register. Register is currently 0xAA500003
PR Bitstream ID	0x03	Read-Only	Read-only PR POF ID register

Table 20. PR_CSR Control and Status Bits

Bit Offset	Description
0	Read and write control register for `pr_start` signal. Refer to Ports on page 95 for details on the `pr_start` signal. `pr_start = PR_CSR[0]` The IP core deasserts `PR_CSR[0]` to value 0 automatically, one clock cycle after the `PR_CSR[0]` asserts. This streamlines the flow to avoid manual assertion and de-assertion of this register to control `pr_start` signal.
1	Reserved.
2-4	Read-only status register for `status[2:0]` signal. `PR_CSR[4:2] = status[2:0]` Refer to Ports on page 95 for details on the status signals.
5	Read and clear bit for interrupt. If you enable the interrupt interface, reading this bit returns the value of the `irq` signal. Writing a 1 clears the interrupt. If you disable the interrupt interface, reading this bit always returns a value of 0.
6-31	Reserved bits. Depends on the Avalon memory-mapped data bus width.

2.3.2. Reconfiguration Sequence

Partial reconfiguration occurs through the Avalon memory-mapped agent interface in the following sequence:

1. Avalon memory-mapped host component writes `0x01` to IP address offset 0x1 to trigger PR operation.
2. Optionally poll the status register until PR `Operation in Progress`. Not polling results in `waitrequest` on first word.
3. Avalon memory-mapped host component writes PR bitstream to IP address offset 0x0, until all the PR bitstream writes complete. When enhanced decompression is on, `waitrequest` activates throughout the PR operation. Ensure that your host can handle `waitrequest` from the agent interface.
4. Avalon memory-mapped host component reads the data from IP address offset 0x1 to check the `status[2:0]` value. Optionally, the Avalon memory-mapped host component reads the `status[2:0]` of this IP during a PR operation to detect any early failure, for example, PR_ERROR.

2.3.3. Interrupt Interface

If you enable the Avalon Memory Mapped agent interface, you can use the optional interrupt interface of the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP.

The IP core asserts `irq` during the following events:

Table 21. Interrupt Interface Events

Status Code	Event
3'b001	PR_ERROR occurred.
3'b010	CRC_ERROR occurred.
3'b011	The IP core detects an incompatible bitstream.
3'b101	The result of a successful PR operation.

After `irq` asserts, the host performs one or more of the following:

- Query for the status of the PR IP core; `PR_CSR[4:2]`.
- Carry out some action, such as error reporting.
- Once the interrupt is serviced, clear the interrupt by writing a "1" to `PR_CSR[5]`.

2.3.4. Parameters

The Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP supports customization of the following parameters.

Table 22. Parameter Settings

Parameter	Value	Description
Use as partial reconfiguration internal host	On|Off	Enables the controller for use as an internal host. Enabling this option auto-instantiates `prblock` and `crcblock` WYSIWYG as part of your design. Disable this option to use the controller as an external host. Connect additional interface signals to the dedicated partial reconfiguration pins.
Enable JTAG debug mode	On|Off	Enables access to the controller by the Quartus Prime Programmer for partial reconfiguration over a JTAG interface.
Enable Avalon-MM slave interface	On|Off	Enables the controller's Avalon memory-mapped agent interface. When this setting is Off, the IP controller enables the conduit interface.
Enable interrupt interface	On|Off	Enables interrupt assertion for detection of incompatible bitstream, `CRC_ERROR`, `PR_ERROR`, or successful partial reconfiguration. Upon interrupt, query `PR_CSR[4:2]` for status. Write a `1` to `PR_CSR[5]` to clear the interrupt. Use only together with the Avalon memory-mapped agent interface.
Enable freeze interface	On|Off	Enables the controller's single-bit freeze interface. This interface identifies whether any region in the design is active or frozen for partial reconfiguration operations. Leave this interface off, and use the freeze interface from the Partial Reconfiguration Region Controller IP instead.
Enable bitstream compatibility check	On|Off	Enables bitstream compatibility checks during partial reconfiguration operation from the external host. Bitstream compatibility check automatically enables when you use partial reconfiguration by internal host. Specify the partial reconfiguration bitstream ID value if you enable this option for partial reconfiguration by external host.

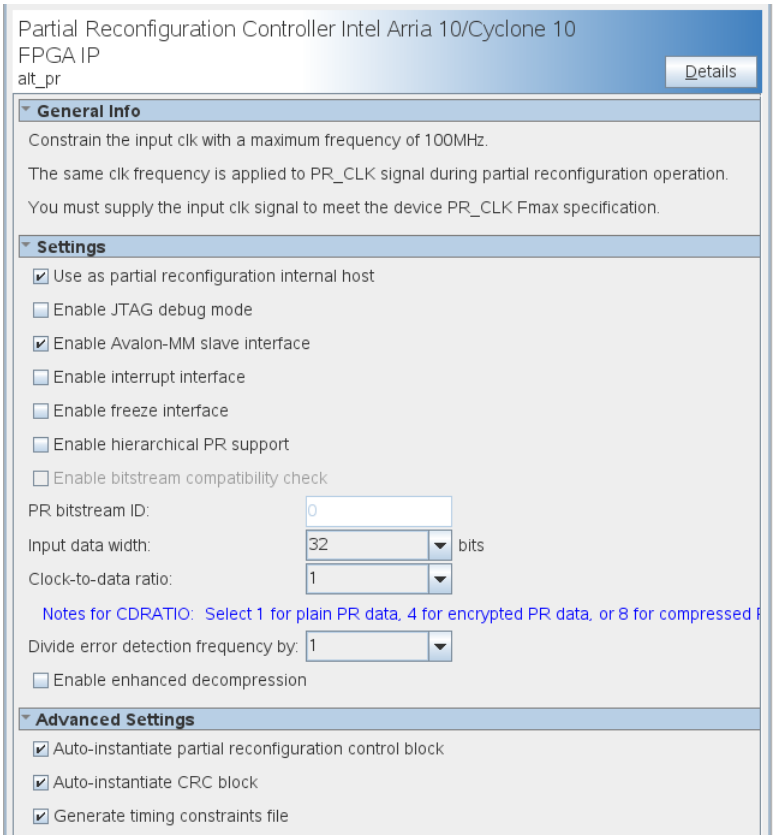
continued...

Parameter	Value	Description
PR bitstream ID	*<32-bit integer>*	Specifies a signed, 32-bit integer value of the partial reconfiguration bitstream ID for the external host. This value must match the partial reconfiguration bitstream ID that the Compiler generates for the target partial reconfiguration design. Locate the partial reconfiguration bitstream ID of the target partial reconfiguration design in the Assembler report (`.asm.rpt`).
Input data width	`1|8|16|32`	Specifies the size of the controller's data conduit interface in bits. Refer to Error Detection CRC Requirements on page 94.
Clock-to-data ratio	`1|4|8`	Specifies the clock-to-data ratio that corresponds with the partial reconfiguration bitstream data type. Refer to the Valid combinations and CD Ratio for Bitstream Encryption and Compression Table.
Divide error detection frequency by	`1..256`	Specifies the divide value of the internal clock. This value determines the frequency of the error detection CRC. The divide value must be a power of two. Refer to device documentation to determine the frequency of the internal clock for the device you select. Refer to Error Detection CRC Requirements on page 94.
Enable enhanced decompression	On|Off	Enable enhanced decompression of partial reconfiguration bitstreams. *Note:* You cannot use enhanced decompression together with encryption simultaneously. Enhanced decompression is only available with the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP.

Table 23. Advanced Settings

Parameter	Value	Description
Auto-instantiate partial reconfiguration control block	On|Off	Automatically includes the partial reconfiguration control block in the controller. When using the controller as an internal host, disable this option to share the partial reconfiguration block with other IP cores. Rather, manually instantiate the partial reconfiguration control block, and connect the relevant signals to the controller.
Auto-instantiate CRC block	On|Off	Automatically includes the CRC block within the controller. Leave this option enabled unless you plan to use single event upset (SEU) IP in the same PR design. If you disable this option, IP generation exports the `crc_error_pin` for manual connection to an external CRC block that you manually instantiate. If you disable this option and then subsequently leave the exported `crc_error_pin` floating, the PR operation is undetermined due to unexpected `crc_error_pin`.
Generate timing constraints file	On|Off	Automatically generates an appropriate Synopsys Design Constraints (`.sdc`) file to constrain the timing of the controller. Disable this option when providing timing constraints in another file.

Figure 62. Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP Parameter Editor



2.3.4.1. Error Detection CRC Requirements

The following describes requirements for enabling the error detection CRC option with various PR configuration method and parameter combinations. Click **Assignments ➤ Device ➤ Device & Pin Options ➤ Error Detection CRC ➤ Enable Error Detection Check** to enable EDCRC prior to PR bitstream generation.

Note: When using the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP with a 32-bit **Input data width**, and with Passive Parallel x1, x8, or x16 configuration, you must turn on the **Enable Error Detection Check** option, and specify a **Divide error detection frequency by** value of 2 or 4.

Note: When using the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP with a 32-bit **Input data width** and Passive Parallel x32 configuration, PR supports **Enable Error Detection Check** on or off. If **Enable Error Detection Check** is on, PR supports all values for **Divide error detection frequency by**.

Note: When using the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP with 1, 8, or 16-bit **Input data width**, and with Passive Parallel x1, x8, x16, or x32 configuration, PR supports **Enable Error Detection Check** turned on or off. If **Enable Error Detection Check** is on, PR supports all values for **Divide error detection frequency by**.

Table 24. Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP Error Detection CRC (EDCRC) Requirements Summary

PR IP Input Data Width	Configuration Mode	Enable Error Detection Check	PR Support
1, 8, 16	Passive Parallel x1, x8, x16	Off	Yes
1, 8, 16	Passive Parallel x1, x8, x16	On	Yes, for all **Divide error detection frequency by** values
32	Passive Parallel x1, x8, x16	Off	No support
32	Passive Parallel x1, x8, x16	On	Yes, for only **Divide error detection frequency by** value 2 or 4
1, 8, 16, 32	Passive Parallel x32	Off	Yes
1, 8, 16, 32	Passive Parallel x32	On	Yes, for all **Divide error detection frequency by** values

2.3.5. Ports

The Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP includes the following interface ports.

Figure 63. Partial Reconfiguration Controller Interface Ports (Internal Host)

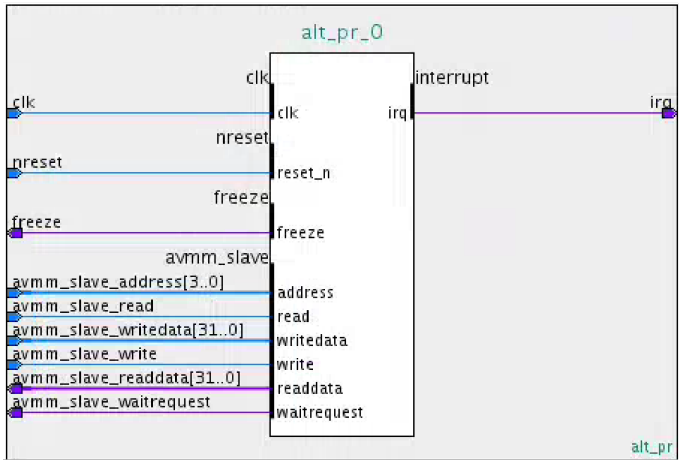


Figure 64. Partial Reconfiguration Controller Interface Ports (External Host)

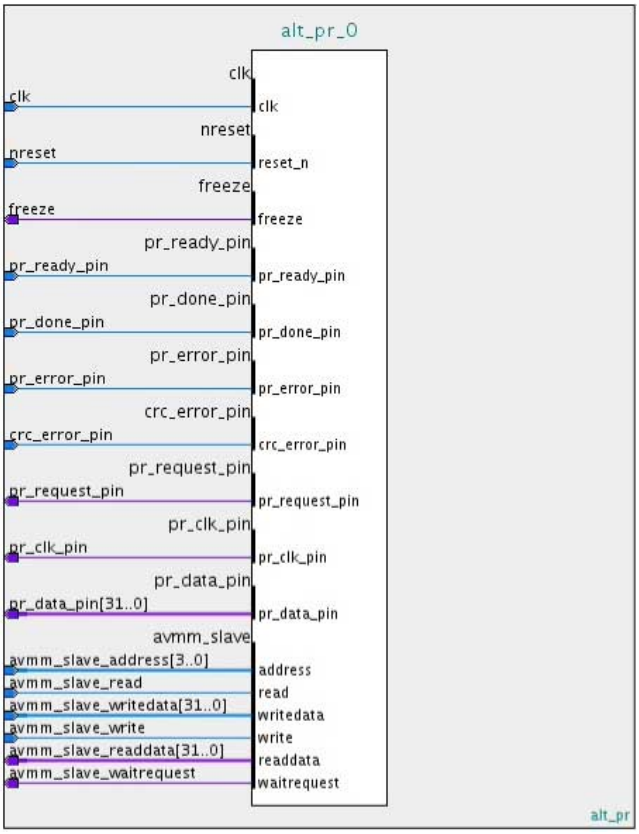


Table 25. Clock/Reset Ports

Port Name	Width	Direction	Function
`nreset`	1	Input	Asynchronous reset for the PR Controller IP core. Resetting the PR Controller IP core during a partial reconfiguration operation initiates the withdrawal sequence.
`clk`	1	Input	User input clock to the PR Controller IP core. The IP core has a maximum clock frequency of 100MHz. The IP core ignores this signal during JTAG debug operations.

Table 26. Freeze Interface Port

Port Name	Width	Direction	Function
`freeze`	1	Output	Active high signal that freezes the PR interface signals of any region undergoing partial reconfiguration. De-assertion of this signal indicates the end of PR operation. Use the Partial Reconfiguration Region Controller IP for this operation rather than the Partial Reconfiguration Controller IP `freeze` signal.

Table 27. Conduit Interface Ports

These ports are available when **Enable Avalon Memory-Mapped agent interface** is **Off**.

Port Name	Width	Direction	Function
pr_start	1	Input	A 0 to 1 transition on this port initiates a PR event. You must assert this signal high for a minimum of one clock cycle, and de-assert the signal low prior to the end of the PR operation. This operation ensures the PR Controller IP core is ready to accept the next pr_start trigger event when the freeze signal is low. The PR Controller IP core ignores this signal during JTAG debug operations.
data[]	1, 8, 16, or 32	Input	Selectable input PR data bus width, either x1, x8, x16, or x32. Once a PR event triggers, the PR event is synchronous with the rising edge of the clk signal, whenever the data_valid signal is high, and the data_ready signal is high. The PR Controller IP core ignores this signal during JTAG debug operations.
data_valid	1	Input	A 0 to 1 transition on this port indicates the data[] port contains valid data. The PR Controller IP core ignores this signal during JTAG debug operations.
data_ready	1	Output	A 0 to 1 transition on this port indicates the PR Controller IP core is ready to read the valid data on the data[] port, whenever the data_valid signal asserts high. The data sender must stop sending valid data if this port is low. This signal deasserts low during JTAG debug operations.
status[2:0]	1	Output	A 3-bit output that indicates the status of PR events. When the IP detects an error (PR_ERROR, CRC_ERROR, or incompatible bitstream error), this signal latches high. This signal only resets at the beginning of the next PR event, when pr_start is high, and freeze is low. For example: 3'b000 – power-up or nreset asserts 3'b001 – PR_ERROR triggers 3'b010 – CRC_ERROR triggers 3'b011 – Incompatible bitstream error detection 3'b100 – PR operation in progress 3'b101 – PR operation passes 3'b110 – Reserved bit 3'b111 – Reserved bit

Table 28. Avalon Memory-Mapped Slave Interface Ports

These signals are available when **Enable Avalon memory-mapped slave interface** is **On**.

Port Name	Width	Direction	Function
avmm_slave_address	4	Input	Avalon memory-mapped address bus. The address bus is in the unit of Word addressing. The PR Controller IP core ignores this signal during JTAG debug operations.
avmm_slave_read	1	Input	Avalon memory-mapped read control. The PR Controller IP core ignores this signal during JTAG debug operations.

continued...

Port Name	Width	Direction	Function
avmm_slave_readdata	32	Output	Avalon memory-mapped read data bus. The PR Controller IP core ignores this signal during JTAG debug operations.
avmm_slave_write	1	Input	Avalon memory-mapped write control. The PR Controller IP core ignores this signal during JTAG debug operations.
avmm_slave_writedata	32	Input	Avalon memory-mapped write data bus. The PR Controller IP core ignores this signal during JTAG debug operations.
avmm_slave_waitrequest	1	Output	Indicates that the IP is busy. Also indicates that the IP core is unable to respond to a read or write request. The IP core pulls this signal high during JTAG debug operations.

Table 29. Interrupt Interface Ports

These ports are available when **Enable interrupt interface** is **On**.

Port Name	Width	Direction	Function
irq	1	Output	The interrupt signal.

Table 30. CRC BLOCK Interface

These ports are available when **Use as Partial Reconfiguration Internal Host** is **Off**, or when you instantiate the CRCBLOCK manually for an internal host.

Port Name	Width	Direction	Function
crc_error_pin	1	Input	Available when you use the PR Controller IP core as an External Host. Connect this port to the dedicated CRC_ERROR pin of the FPGA undergoing partial reconfiguration.

Table 31. PR Block Interface

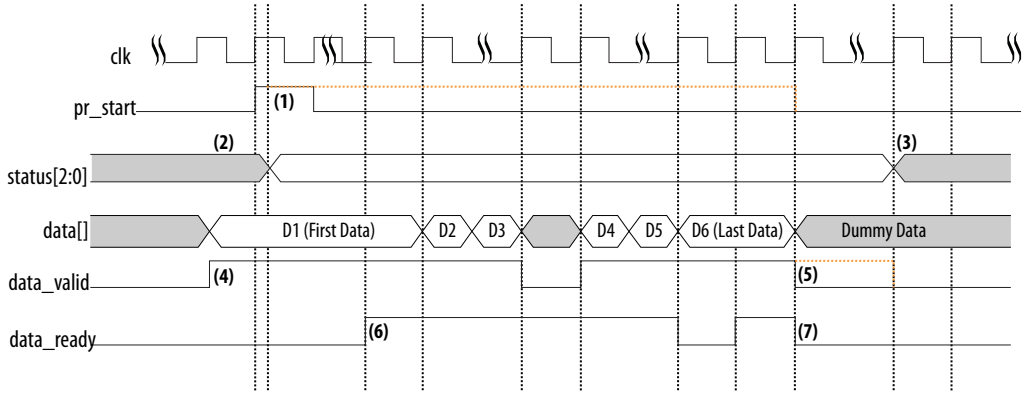
These options are available when **Use as Partial Reconfiguration Internal Host** is **Off**, or when you instantiate the PRBLOCK manually for an internal host.

Port Name	Width	Direction	Function
pr_ready_pin	1	Input	Connect this port to the dedicated PR_READY pin of the FPGA undergoing partial reconfiguration.
pr_error_pin	1	Input	Connect this port to the dedicated PR_ERROR pin of the FPGA undergoing partial reconfiguration.
pr_done_pin	1	Input	Connect this port to the dedicated PR_DONE pin of the FPGA undergoing partial reconfiguration.
pr_request_pin	1	Output	Connect this port to the dedicated PR_REQUEST pin of the FPGA undergoing partial reconfiguration.
pr_clk_pin	1	Output	Connect this port to the dedicated DCLK of the FPGA undergoing partial reconfiguration.
pr_data_pin[31..0]	16|32	Output	Connect this port to the dedicated DATA[31..0] pins of the FPGA undergoing partial reconfiguration.

2.3.6. Timing Specifications

The following timing diagram illustrates a successful PR operation with Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP. The `status[2:0]` output signal indicates whether the operations passes or fails. The PR operation initiates upon assertion of the `pr_start` signal. Monitor the `status[]` signal to detect the end of the PR operation.

Figure 65. Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP Timing Specifications



The following notes correspond to locations (1) through (7) in the timing diagram:

1. Assert `pr_start` signal high for a minimum of one clock cycle to initiate PR. Deassert `pr_start` before sending the last data.
2. `status[]` signal updates after `pr_start` is acknowledged. This signal changes during a PR operation if CRC_ERROR, PR_ERROR, or bitstream incompatibility error occurs.
3. `status[]` signal changes after a PR operation if CRC_ERROR asserts and no error occurs during the previous PR operation.
4. There is no requirement to assert the `data_valid` signal at the same time as the `pr_start` signal. Provide the `data[]`, and assert `data_valid`, when appropriate.
5. Either drive the `data_valid` signal low after sending the last data, or continue to assert `data_valid` high with dummy data on `data[]` until the IP reads the end of PR from `status[]`.
6. `data[]` transfers only when `data_valid` and `data_ready` assert on the same cycle. Do not drive new data on the data bus, when both `data_valid` and `data_ready` are not high.
7. The `data_ready` signal drives low after the PR IP Controller core receives the last data, or when the PR IP Controller cannot accept data.

2.3.7. PR Control Block and CRC Block Verilog HDL Manual Instantiation

The Partial Reconfiguration Controller Arria 10/Cyclone 10 IP includes the PR control block. However, if you create your own custom logic to perform the function of the IP core, you can manually instantiate the control block to communicate with the FPGA system.

The following example instantiates a PR control block inside a top-level Arria 10 PR project, `Chip_Top`, in Verilog HDL:

```
Chip_Top:
module Chip_Top (
 //User I/O signals (excluding PR related signals)
..
..
//PR interface and configuration signals declaration
 wire pr_request;
 wire pr_ready;
 wire pr_done;
 wire crc_error;
 wire dclk;
 wire [31:0] pr_data;

twentynm_prblock m_pr
 (
 .clk (dclk),
 .corectl (1'b1),
 .prrequest(pr_request),
 .data (pr_data),
 .error (pr_error),
 .ready (pr_ready),
 .done (pr_done)
 );

twentynm_crcblock m_crc
 (
 .clk (clk),
 .shiftnld (1'b1),
 .crcerror (crc_error)
 );
endmodule
```

For more information about port connectivity for reading the Error Message Register (EMR), refer to the *AN539: Test Methodology of Error Detection and Recovery using CRC*.

Related Information

AN539: Test Methodology of Error Detection and Recovery using CRC in Intel FPGA Devices

2.3.8. PR Control Block and CRC Block VHDL Manual Instantiation

The following example shows manual instantiation of a PR control block inside your top-level Arria 10 project, `Chip_Top`, in VHDL:

```
module Chip_Top is port (
 --User I/O signals (excluding signals that relate to PR)
 ..
 ..
 )
-- Following shows the connectivity within the Chip_Top module
Core_Top : Core_Top
```

```
 port_map (
 ..
 ..
 );
m_pr : twentynm_prblock
 port map(
clk => dclk,
 corectl =>'1', --1 - when using PR from inside
 --0 - for PR from pins; You must also enable
 -- the appropriate option in Quartus Prime settings
 prrequest => pr_request,
 data => pr_data,
 error => pr_error,
 ready => pr_ready,
 done => pr_done
 );
m_crc : twentynm_crcblock
 port map(
 shiftnld => '1', --If you want to read the EMR register when
 clk => dummy_clk, --error occurrs, refer to AN539 for the
 --connectivity for this signal. If you only want
 --to detect CRC errors, but plan to take no
 --further action, you can tie the shiftnld
 --signal to logical high.
 crcerror => crc_error
 );
```

Note: You are not required to connect a real clock source to `dummy_clk`, but you must connect `dummy_clk` to an I/O pin to avoid removal of this signal.

2.3.8.1. PR Control Block and CRC Block VHDL Component Declaration

The following example shows manual instantiation of the PR control block and the CRC block in your Arria 10 PR design:

1. Use the code sample below, containing the component declaration in VHDL. This code performs the PR function from within the core (code block within `Core_Top`).

```
module Chip_Top is port (
    --User I/O signals (excluding signals that relate to PR)
    ..
    ..
)
-- Following shows the connectivity within the Chip_Top module
Core_Top : Core_Top
port_map (
    ..
    ..
);

m_pr : twentynm_prblock
port map(
    clk => dclk,
    corectl =>'1', --1 - when using PR from inside
    --0 - for PR from pins; You must also enable
    -- the appropriate option in Quartus Prime settings
    prrequest => pr_request,
    data => pr_data,
    error => pr_error,
    ready => pr_ready,
    done => pr_done
);

m_crc : twentynm_crcblock
port map(
    shiftnld => '1', --If you want to read the EMR register when
    clk => dummy_clk, --error occurrs, refer to AN539 for the
    --connectivity forthis signal. If you only want
```

```
    --to detect CRC errors, but plan to take no
    --further action, you can tie the shiftnld
    --signal to logical high.
    crcerror => crc_error
);
```

Note: This VHDL example is adaptable for Verilog HDL instantiation.

2. Add additional ports to `Core_Top` to connect to both components.
3. Follow these rules when connecting the PR control block to the rest of your design:
   - Set the `corectl` signal to '1' (when using partial reconfiguration from core) or to '0' (when using partial reconfiguration from pins).
   - The `corectl` signal must match the **Enable PR pins** option setting in the **Device and Pin Options** dialog box (**Assignments ➤ Device ➤ Device and Pin Options**).
   - When performing partial reconfiguration from pins, the Fitter automatically assigns the PR unassigned pins. Assign all the dedicated PR pins using Pin Planner (**Assignments ➤ Pin Planner**) or Assignment Editor (**Assignments ➤ Assignment Editor**).
   - When performing partial reconfiguration from the core logic, connect the `prblock` signals to either core logic or I/O pins, excluding the dedicated programming pin, such as DCLK.

2.3.9. PR Control Block Signals

The following table lists the partial reconfiguration control block interface signals for the Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP:

Table 32. PR Control Block Interface Signals

Signal	Width	Direction	Description
pr_data	[31:0]	Input	Carries the configuration bitstream.
pr_done	1	Output	Indicates that the PR process is complete.
pr_ready	1	Output	Indicates that the control block is ready to accept PR data from the control logic.
pr_error	1	Output	Indicates a partial reconfiguration error.
pr_request	1	Input	Indicates that the PR process is ready to begin.
corectl	1	Input	Determines whether you are performing the partial reconfiguration internally, or through pins.

Note:

- You can specify a configuration width of 8, 16, or 32 bits, but the interface always uses 32 pins.
- All the inputs and outputs are asynchronous to the PR clock (`clk`), except `data` signal. `data` signal is synchronous to `clk` signal.
- PR clock must be free-running.
- `data` signal must be 0 while waiting for `ready` signal.

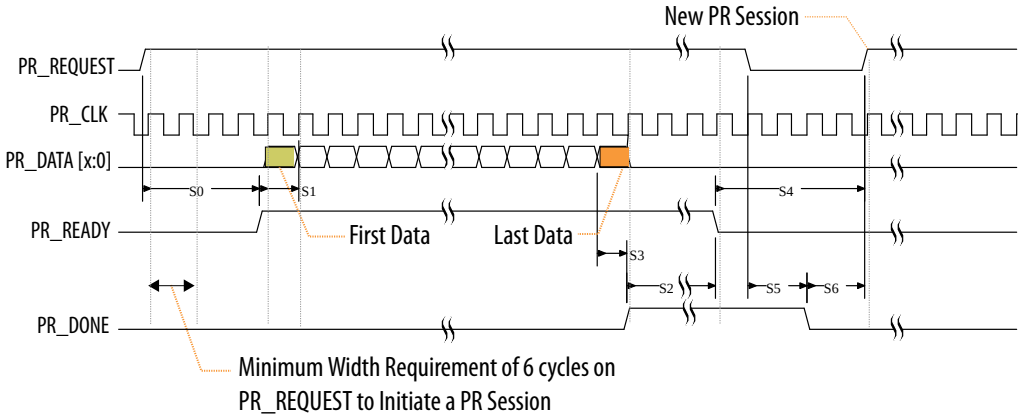
2.3.9.1. PR Control Block Signal Timing Diagrams

2.3.9.1.1. Successful PR Session (Arria 10 Example)

The following flow describes a successful Arria 10 PR session:

1. Assert PR_REQUEST and wait for PR_READY; drive PR_DATA to 0.
2. The PR control block asserts PR_READY, asynchronous to `clk`.
3. Start sending Raw Binary File (`.rbf`) to the PR control block, with 1 valid word per clock cycle. On `.rbf` file transfer completion, drive PR_DATA to 0. The PR control block asynchronously asserts PR_DONE when the control block completes the reconfiguration operation. The PR control block deasserts PR_READY on configuration completion.
4. Deassert PR_REQUEST. The PR control block acknowledges the end of PR_REQUEST, and deasserts PR_DONE. The host can now initiate another PR session.

Figure 66. Timing Diagram for Successful Arria 10 PR Session



Related Information

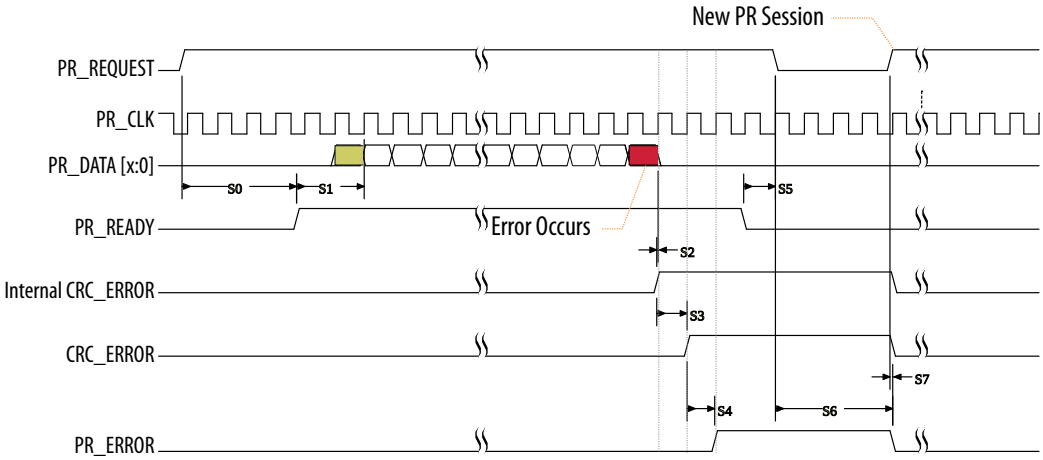
Raw Binary Programming File Byte Sequence Transmission Examples on page 33

2.3.9.1.2. Unsuccessful PR Session with Configuration Frame Readback Error (Arria 10 Example)

The following flow describes an Arria 10 PR session with error in the EDCRC verification of a configuration frame readback:

1. The PR control block internally detects a CRC error.
2. The CRC control block then asserts CRC_ERROR.
3. The PR control block asserts the PR_ERROR.
4. The PR control block deasserts PR_READY, so that the host can withdraw the PR_REQUEST.
5. The PR control block deasserts CRC_ERROR and clears the internal CRC_ERROR signal to get ready for a new PR session. The host can now initiate another PR session.

Figure 67. Timing Diagram for Unsuccessful Arria 10 PR Session with Configuration Frame Readback Error

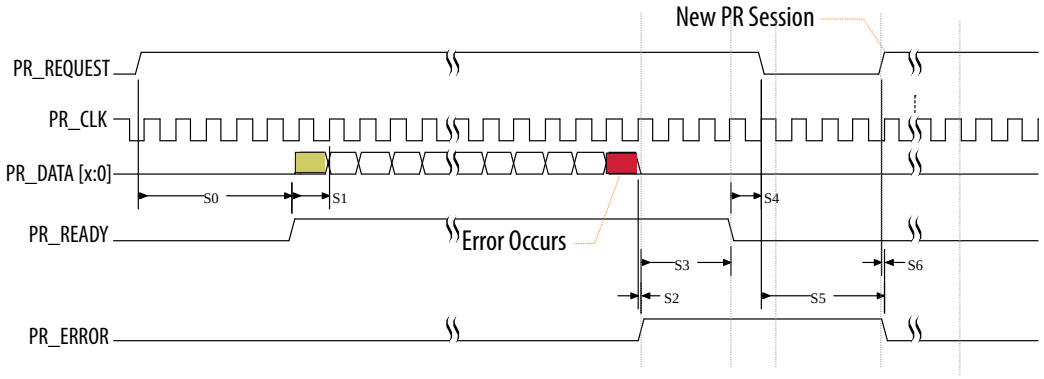


2.3.9.1.3. Unsuccessful PR Session with PR_ERROR (Arria 10 Example)

The following flow describes an Arria 10 PR session with transmission error or configuration CRC error:

1. The PR control block asserts PR_ERROR.
2. The PR control block deasserts PR_READY, so that the host can withdraw PR_REQUEST.
3. The PR control block deasserts PR_ERROR to get ready for a new PR session. The host can now initiate another PR session.

Figure 68. Timing Diagram for Unsuccessful Arria 10 PR Session with PR_ERROR

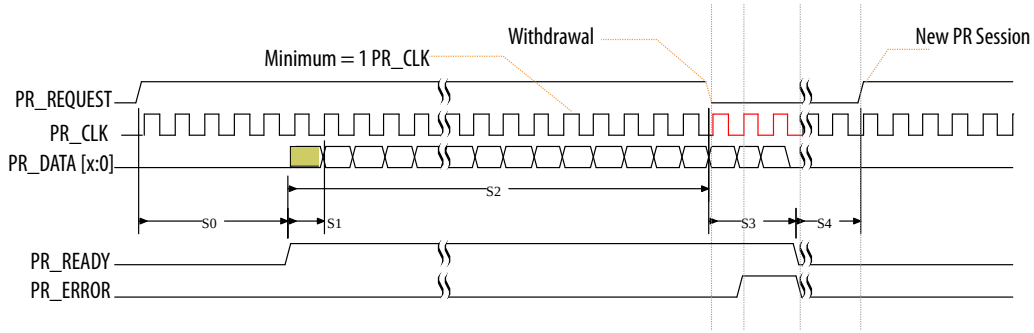


2.3.9.1.4. Late Withdrawal PR Session (Arria 10 Example)

The following flow describes a late withdrawal Arria 10 PR session:

1. The PR host can withdraw the request after the PR control block asserts PR_READY.
2. The PR control block deasserts PR_READY. The host can now initiate another PR session.

Figure 69. Timing Diagram for Late Withdrawal Arria 10 PR Session



Note: The PR host can withdraw the request any time before the PR controller asserts PR_READY. Therefore, the PR host must not return until the PR control block asserts PR_READY. Provide at least 10 PR_CLK cycles after deassertion of PR_REQUEST, before requesting a new PR session.

2.3.10. Configuring an External Host for Arria 10 or Cyclone 10 GX Designs

When using external host configuration, the external host initiates partial reconfiguration, and monitors the PR status using the external PR dedicated pins during user mode. In this mode, the external host must respond appropriately to the handshake signals for successful partial reconfiguration. The external host writes the partial bitstream data from external memory into the Arria 10 or Cyclone 10 GX device. Co-ordinate system-level partial reconfiguration by ensuring that you prepare the correct PR region for partial reconfiguration. After reconfiguration, return the PR region into operating state.

To use an external host for your design:

1. Click **Assignments ➤ Device ➤ Device & Pin Options**.
2. Select the **Enable PR Pins** option in the **Device & Pin Options** dialog box. This option automatically creates the special partial reconfiguration pins, and defines the pins in the device pin-out. This option also automatically connects the pins to PR control block internal path.

   Note: If you do not select this option, you must use an internal or HPS host. You do not need to define pins in your design top-level entity.
3. Connect these top-level pins to the specific ports in the PR control block.

The following table lists the PR pins that automatically constrain when you turn on **Enable PR Pins**, and the specific PR control block port connection to the pin:

Table 33. Partial Reconfiguration Dedicated Pins

Pin Name	Type	PR Control Block Port Name	Description
PR_REQUEST	Input	prrequest	Logic high on this pin indicates that the PR host is requesting partial reconfiguration.
PR_READY	Output	ready	Logic high on this pin indicates that the PR control block is ready to begin partial reconfiguration.
PR_DONE	Output	done	Logic high on this pin indicates that the partial reconfiguration is complete.
PR_ERROR	Output	error	Logic high on this pin indicates an error in the device during partial reconfiguration.
DATA[31:0]	Input	data	These pins provide connectivity for PR_DATA to transfer the PR bitstream to the PR controller.
DCLK	Input	clk	Receives synchronous PR_DATA.

Note:

1. PR_DATA can be 8, 16, or 32-bits in width.
2. Ensure that you connect the `corectl` port of the PR control block to 0.

Example 4. Verilog RTL for External Host PR

```
module top(
     // PR control block signals
     input  logic        pr_clk,
     input  logic        pr_request,
     input  logic [31:0] pr_data,
     output logic        pr_error,
     output logic        pr_ready,
     output logic        pr_done,

     // User signals
     input  logic i1_main,
     input  logic i2_main,
     output logic o1
```

```
);

// Instantiate the PR control block
twentynm_prblock m_prblock
(
    .clk(pr_clk),
    .corectl(1'b0),
    .prrequest(pr_request),
    .data(pr_data),
    .error(pr_error),
    .ready(pr_ready),
    .done(pr_done)
);


// PR Interface partition
pr_v1 pr_inst(
   .i1(i1_main),
   .i2(i2_main),
   .o1(o1)
);

endmodule
```

Example 5. VHDL RTL for External Host PR

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
port(
      -- PR control block signals
      pr_clk: in std_logic;
      pr_request: in std_logic;
      pr_data: in std_logic_vector(31 downto 0);

      pr_error: out std_logic;
      pr_ready: out std_logic;
      pr_done: out std_logic;

      -- User signals
      i1_main: in std_logic;
      i2_main: in std_logic;
      o1: out std_logic
);
end top;

architecture behav of top is

component twentynm_prblock is
port(
      clk: in std_logic;
      corectl: in std_logic;
      prrequest: in std_logic;
      data: in std_logic_vector(31 downto 0);
      error: out std_logic;
      ready: out std_logic;
      done: out std_logic
);
end component;

component pr_v1 is
port(
      i1: in std_logic;
      i2: in std_logic;
      o1: out std_logic
);
end component;

signal pr_gnd : std_logic;
```

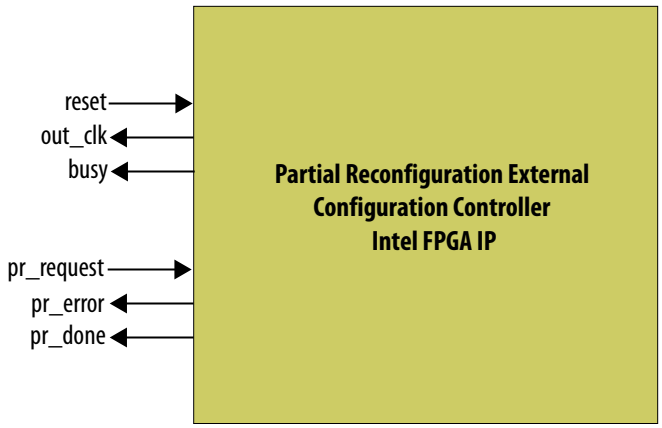
```
begin

pr_gnd <= '0';

-- Instantiate the PR control block
m_prblock: twentynm_prblock port map
(
    pr_clk,
    pr_gnd,
    pr_request,
    pr_data,
    pr_error,
    pr_ready,
    pr_done
);


-- PR Interface partition
pr_inst : pr_v1 port map
(
   i1_main,
   i2_main,
   o1
);

end behav;
```

2.4. Partial Reconfiguration External Configuration Controller Intel FPGA IP

The Partial Reconfiguration External Configuration Controller Intel FPGA IP supports partial reconfiguration via an external source.

When using external configuration, you must connect all the top-level ports of the Partial Reconfiguration External Configuration Controller Intel FPGA IP to the `pr_request` and `status` pins. These connections allow the handshaking of the host with the SDM from the Agilex 7, Agilex 5, or Stratix 10 device core. The SDM determines which types of configuration pins to use, according your MSEL setting.

Figure 70. Partial Reconfiguration External Configuration Controller Intel FPGA IP



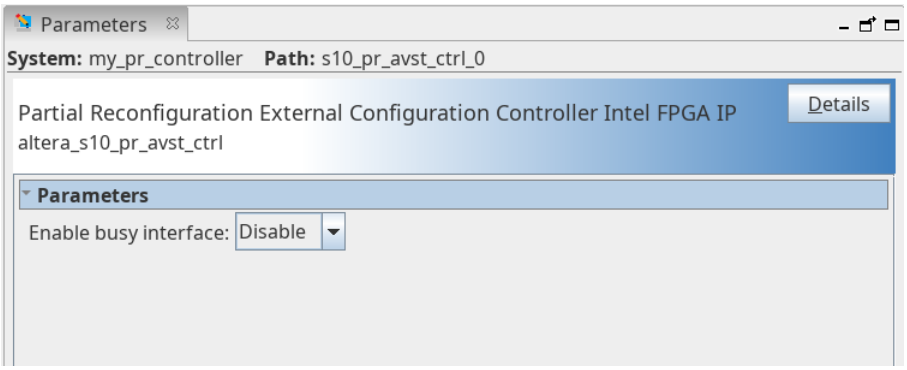
2.4.1. Parameters

The Partial Reconfiguration External Configuration Controller Intel FPGA IP supports customization of the following parameters.

Table 34. Partial Reconfiguration External Configuration Controller Parameter Settings

Parameter	Value	Description
Enable Busy Interface	**On/Off**	Allows you to Enable or Disable the Busy interface, which asserts a signal to indicate that PR processing is in progress during external configuration. Default setting is **Off**.

Figure 71. Parameter Editor



2.4.2. Ports

The Partial Reconfiguration External Configuration Controller Intel FPGA IP includes the following interface ports.

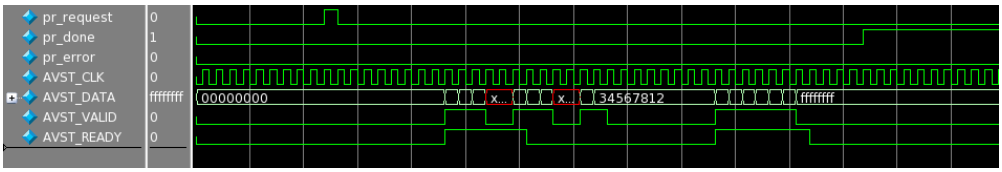
Table 35. Ports

Port Name	Width	Direction	Function
`pr_request`	1	Input	Indicates that the PR process is ready to begin. The signal is a conduit not synchronous to any clock signal.
`pr_error`	2	Output	Indicates a partial reconfiguration error.: • `2'b01`—general PR error • `2'b11`—incompatible bitstream error These signals are conduits not synchronous to any clock source.
`pr_done`	1	Output	Indicates that the PR process is complete. The signal is a conduit not synchronous to any clock signal.
`start_addr`	1	Input	Specifies the start address of PR data in Active Serial Flash. You enable this signal by selecting either **Avalon-ST** or **Active Serial** for the **Enable Avalon-ST Pins or Active Serial Pins** parameter. The signal is a conduit not synchronous to any clock signal.
`reset`	1	Input	Active high, synchronous reset signal.
`out_clock`	1	Output	Clock source that generates from an internal oscillator.
`busy`	1	Output	The IP asserts this signal to indicate PR data transfer in progress. You enable this signal by selecting **Enable** for the **Enable busy interface** parameter.

2.4.3. Partial Reconfiguration External Controller Intel FPGA IP Timing Specifications

Timing Specifications: Partial Reconfiguration External Controller Intel FPGA IP illustrates a successful PR operation with the Partial Reconfiguration External Controller Intel FPGA IP. The PR operation initiates upon assertion of the `pr_request` signal. The `avst_ready` output signal indicates whether the SDM is ready to accept data from an external host.

Figure 72. Timing Specifications: Partial Reconfiguration External Controller Intel FPGA IP



2.4.4. Configuring an External Host for Agilex 7, Agilex 5, and Stratix 10 Designs

You can optionally use an external host to write the partial bitstream data from external memory into the Agilex 7, Agilex 5, or Stratix 10 device. When using external host configuration, the external host initiates partial reconfiguration by asserting the `pr_request` signal. The external host monitors the PR status through the `pr_done` and `pr_error` signals.

The external host must respond appropriately to the handshake signals for successful partial reconfiguration. Co-ordinate system-level partial reconfiguration by ensuring that you prepare the correct PR region for partial reconfiguration. After reconfiguration, return the PR region into operating state.

To configure an external host, follow these steps:

1. Parameterize and generate the Partial Reconfiguration External Configuration Controller Intel FPGA IP, as Specifying the IP Core Parameters and Options (Quartus Prime Pro Edition) on page 133 describes.
2. Connect the Partial Reconfiguration External Configuration Controller `pr_request`, `pr_done`, and `pr_error` signals to top-level pins for control and monitor by the external host. You can assign the pin location by clicking **Assignments ➤ Pin Planner**.
3. Click **Assignments ➤ Device**, and then click the **Device & Pin Options** button.
4. In the **Category** list, click **Configuration**.
5. For the **Configuration scheme**, select the scheme that matches with your full device configuration. For example, if your full device configuration uses the **AVSTx32** scheme, the PR configuration must use **AVSTx32**.This option automatically reserves dedicated Avalon streaming configuration pins for partial reconfiguration during user mode. The pins are exactly same as the Avalon streaming pins that you use for full device configuration.

The following table describes the PR pins that the external host uses. The PR streaming to Avalon streaming pins must conform to the Avalon streaming specification for data transfer with backpressure.

Table 36. Partial Reconfiguration External Configuration Pins

Pin Name	Type	Description
pr_request	Input	User-assigned port connected to Partial Reconfiguration External Configuration Controller IP. Logic high on this pin indicates that the PR host is requesting partial reconfiguration.
pr_done	Output	User-assigned port connected to Partial Reconfiguration External Configuration Controller IP. Logic high on this pin indicates that the partial reconfiguration is complete.
pr_error	Output	User-assigned port connected to Partial Reconfiguration External Configuration Controller IP. Logic high on this pin indicates an error in the device during partial reconfiguration.
avst_data: avstx8 - [7:0] avstx16 - [15:0] avstx32 - [31:0]	Input	These pins provide connectivity for the external host to transfer the PR bitstream to the SDM. The avstx8 data pins are part of the SDM I/O. avstx16 and avstx32 data pins are from I/O 48 bank 3A.
avst_clk	Input	Clocks the Avalon streaming interfaces. avst_data and avst_valid are synchronous with avst_clk. The avstx8 clk pin is part of the SDM I/O. avstx16 and avstx32 are from I/O 48 bank 3A.
avst_valid	Input	Logic high on this pin indicates the data in avst_data is valid data. The avstx8 data pins are part of the SDM I/O. avstx16 and avstx32 data pins are from I/O 48 bank 3A.
avst_ready	Output	Logic high on this pin indicates the SDM is ready to accept data from an external host. This output is part of the SDM I/O.

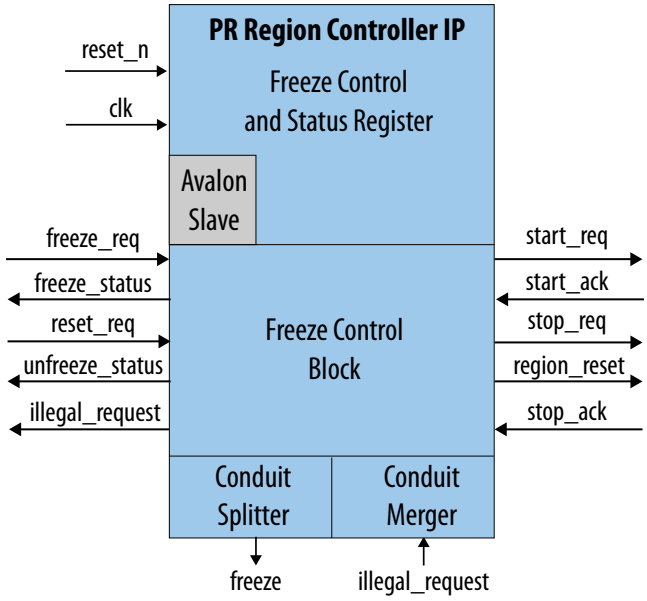
2.5. Partial Reconfiguration Region Controller Intel FPGA IP

The Partial Reconfiguration Region Controller Intel FPGA IP provides a standard interface through the Freeze Control block that controls handshaking with the PR region. The PR handshake ensures that PR region transactions complete before freeze of the interface.

Table 37. PR Region Controller Sections

IP Component	Description
Freeze Control and Status Register	Freeze status register that generates the freeze output signal.
Freeze Control Block	Performs PR handshaking and resets the PR region.
Conduit Splitter	Connects the controller's freeze signal to one or more Freeze Bridge components. Receives the freeze signal from the Freeze Control Block, and assigns the freeze input signal to one or more freeze output signals.
Conduit Merger	Connects the illegal_request signal from one or more Freeze Bridge components to the PR Region Controller. The illegal_request is a single-bit output signal from the Freeze Bridge. Conduit Merger concatenates the single-bit signal from multiple Freeze Bridges into a multi-bit bus. The Conduit Merger then connects the bus to the Freeze Control Block.

Figure 73. Partial Reconfiguration Region Controller IP Core



2.5.1. Registers

The Partial Reconfiguration Region Controller IP core performs the following operations in a partial reconfiguration:

Figure 74. Freeze Control Block PR Handshake Timing

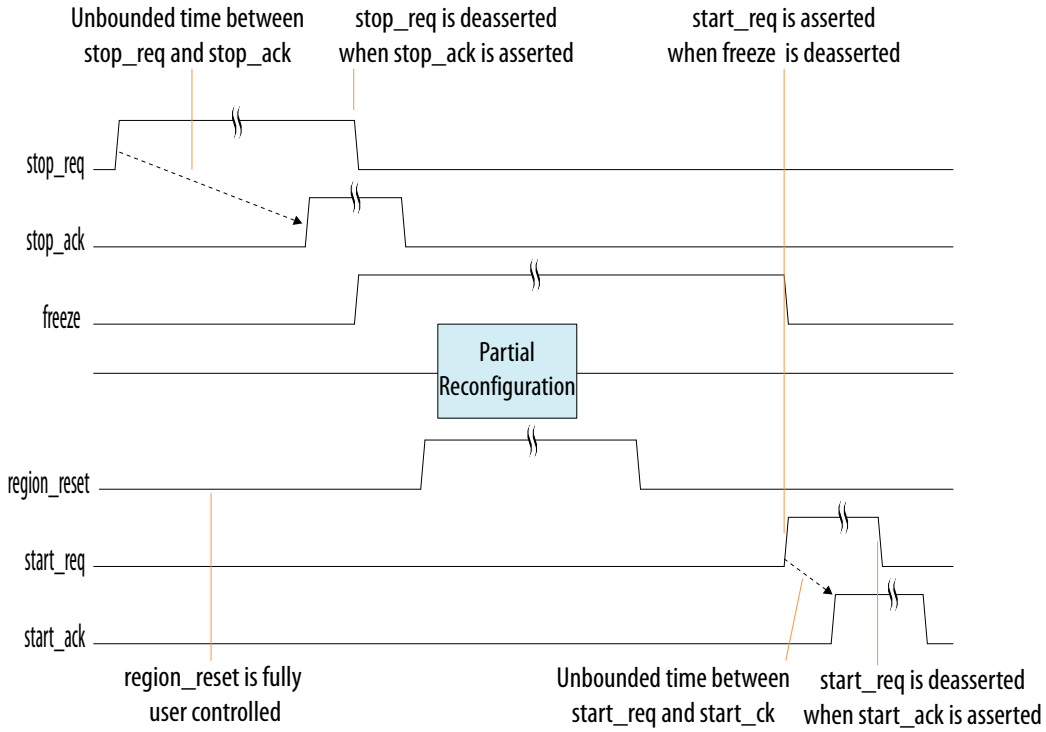


Figure 75. Register States and Programming Model

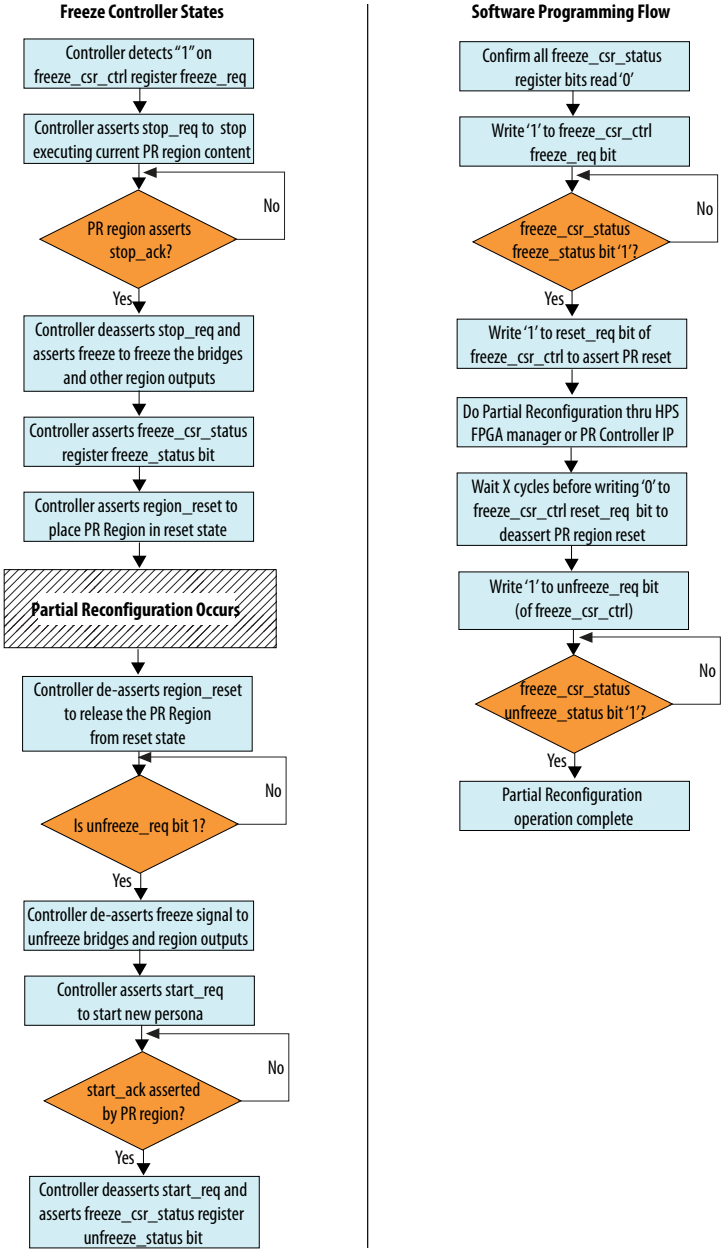


Table 38. Register Map

Name	Address Offset	Access	Description
freeze_csr_status	0x00	Read-Only	Freeze status register.
csr_ctrl	0x01	Read or Write	Control register to enable and disable freeze.
freeze_illegal_req	0x02	Read or Write	High on any bit indicates an illegal request during the freeze state.
freeze_reg_version	0x03	Read-Only	Read-only version register. This register is currently 0xAD000003.

Table 39. freeze_csr_status

Bit	Fields	Access	Default Value	Description
31:2	Reserved	N/A	0x0	Reserved bits. Reading these bits always returns zeros.
1	unfreeze_status	R	0	Hardware sets this bit to 1 after the PR region returns start_ack to indicate successful start of the persona. Hardware clears this bit to 0 when the unfreeze_req bit is low. This bit is 1 when bridges and other PR region outputs release from reset.
0	freeze_status	R	0	Hardware sets this bit to 1 after the PR region returns the stop_ack signal to indicate that the PR region is ready to enter the frozen state Hardware clears this bit to 0 when the freeze_req bit is low. This bit is 0 when bridges and other PR region outputs release from reset.

Table 40. freeze_csr_ctrl

Bit	Fields	Access	Default Value	Description
31:3	Reserved	N/A	0x0	Reserved bits. Reading these bits always returns zeros.
2	unfreeze_req	R/W	0	Write 1 to this bit to request unfreezing the PR region interfaces. Hardware clears this bit after unfreeze_status is high. Write 0 to this bit to terminate the unfreeze request. Do not assert this bit and the freeze_req bit at the same time. If both freeze_req and unfreeze_req assert at the same time, it is an invalid operation.
1	reset_req	R/W	0	Write 1 to start resetting the PR persona. Write 0 to stop resetting the PR persona.
0	freeze_req	R/W	0	Write 1 to this bit to start freezing the PR region interfaces. Hardware clears this bit after freeze_status is high.

continued...

Bit	Fields	Access	Default Value	Description
				Write 0 to this bit to terminate the freeze request if the PR region never returns `stop_ack` after this bit asserts. Do not assert this bit and the `unfreeze_req` bit at the same time. Asserting `freeze_req` and `unfreeze_req` simultaneously is an invalid operation.

Table 41. freeze_illegal_request

Bit	Fields	Access	Default Value	Description
31:n	Reserved	N/A	0x0	Reserved bits. Reading these bits always returns zeros.
n-1:0	illegal_request	R/W	0	High on any bit of this bus indicates a read or write issue by a static region master when an Avalon memory-mapped slave freeze bridge is in the freeze state. Identify which freeze bridge has an illegal request by checking each bit on the bus. For example, when `illegal_request` bit 2 is high, an illegal request occurred in the freeze bridge that connects to interface `freeze_conduit_in2` This bus triggers the interrupt signal. Write 1 to clear this bit. n is the number of bridges.

Table 42. freeze_reg_version

Bit	Fields	Access	Default Value	Description
31:0	Version Register	Read-Only	AD000003	This register bit indicates the CSR register version number. Currently the CSR register is version 0xAD000003.

2.5.2. Parameters

The Partial Reconfiguration Region Controller IP core supports customization of the following parameters.

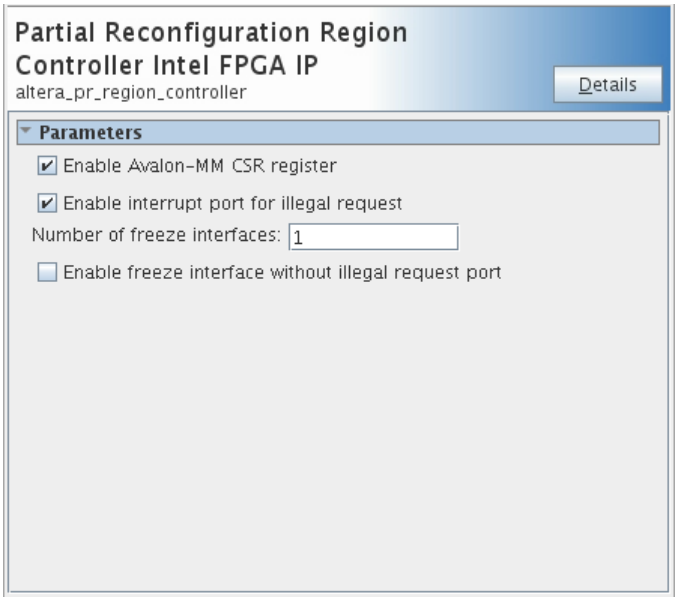
Table 43. Partial Reconfiguration Region Controller Parameter Settings

Parameter	Value	Default	Description
Enable Avalon-MM CSR register	On/Off	On	Enables Avalon memory-mapped CSR registers in the PR region controller. Disable this option to expose a conduit interface and not instantiate the CSR block.
Enable interrupt port for illegal request	On/Off	On	Enables the interrupt port for illegal operations in the PR region controller.

continued...

Parameter	Value	Default	Description
Number of freeze interfaces	*number*		Specifies the number of freeze interfaces for freeze operations. You can connect each freeze interface to a freeze bridge or you can use the interface to control other freeze logic.
Enable freeze interface without illegal request port	On/Off	Off	Enables creation of additional freeze interface, without the illegal request port.
Specify the number of freeze interfaces without illegal request port	*number*		Specifies the number of freeze interfaces without an illegal request port for freeze operations. Only available when you turn on **Enable freeze interface without illegal request port**.

Figure 76. Partial Reconfiguration Region Controller Parameter Editor



2.5.3. Ports

The Partial Reconfiguration Region Controller IP has the following ports.

Table 44. Freeze CSR Block Ports

These ports are available when **Enable Avalon Memory-Mapped CSR Register** is **On**.

Port	Width	Direction	Description
`clock_clk`	1	Input	IP core input clock.
Reset			
`reset_reset`	1	Input	Synchronous reset.
`avl_csr_addr`	2	Input	Avalon memory-mapped address bus. The address bus is in word addressing.
`avl_csr_read`	1	Input	Avalon memory-mapped read control to CSR block.
`avl_csr_write`	1	Input	Avalon memory-mapped write control to CSR.

continued...

Port	Width	Direction	Description
avl_csr_writedata	32	Input	Avalon memory-mapped write data bus to CSR.
avl_csr_readdata	32	Output	Avalon memory-mapped read data bus from CSR.
interrupt_sender_irq	1	output	Trigger by illegal read or illegal write.

Table 45. Freeze Control Block Ports

Port	Width	Direction	Description
pr_handshake_stop_req	1	Output	An assertion on this output requests that the PR persona stop executing.
pr_handshake_stop_ack	1	Input	A value of 1 on this input acknowledges that the executing PR persona stops executing and a new persona can replace it.
pr_handshake_start_req	1	Output	An assertion on this output requests that the new PR persona starts executing.
pr_handshake_start_ack	1	Input	A value of 1 on this input acknowledges that the new PR persona starts executing and can stop executing on a pr_handshake_stop_req.
conduit_control_freeze_req	1	Input	Write 1 on this bit to start freezing the PR region interfaces.
conduit_control_unfreeze_req	1	Input	Write 1 on this bit to stop freezing the PR region interfaces.
conduit_control_freeze_status	1	Output	High on this bit indicates that the PR region is successfully goes into freezing state.
conduit_control_reset	1	Input	Write 1 on this bit to reset the PR region.
conduit_control_unfreeze_status	1	Output	High on this bit indicates that the PR region successfully leaves freezing state.
conduit_control_illegal_req	n	Output	High on this bit indicates illegal data transactions occurring through a Freeze Bridge IP when freeze is active.

Table 46. Conduit Splitter and Merger Interface Ports

Signal	Width	Direction	Description
bridge_freeze0_freeze	1	Output	This output connects to the freeze input signal of a freeze bridge IP or to control other freeze logic. (Multiple interfaces generate according to the number of freeze interfaces)
bridge_freeze0_illegal_request	1	Input	This input connects to the illegal_request output signal from an instance of the Freeze Bridge IP.

Figure 77. Partial Reconfiguration Region Controller Interface Ports (Control and Status Register Block Enabled)

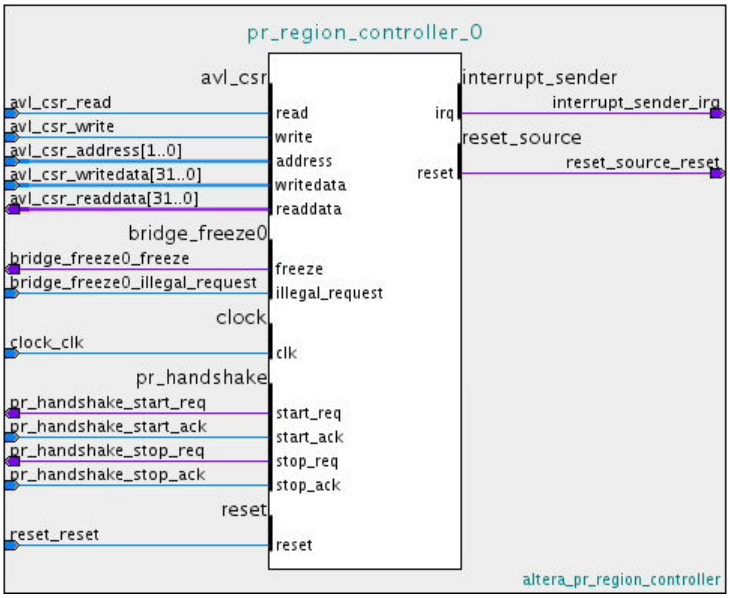
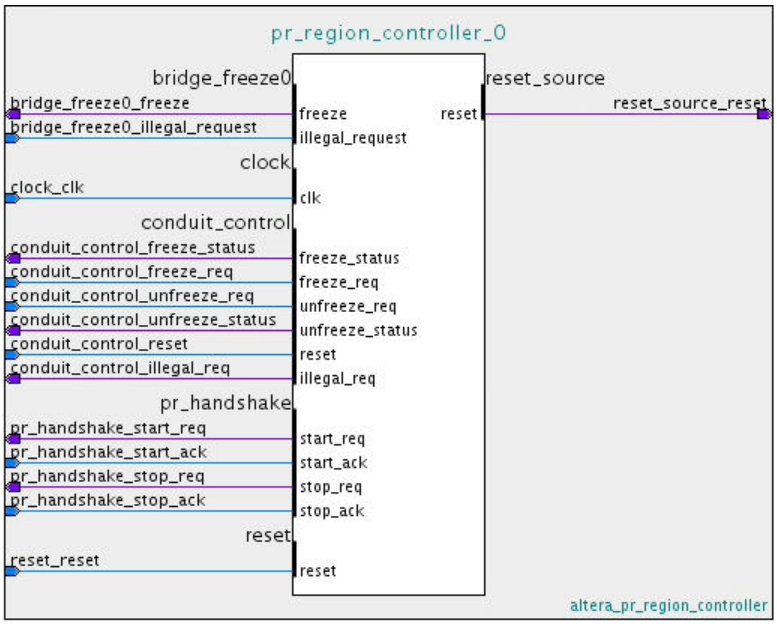


Figure 78. Partial Reconfiguration Region Controller Interface Ports (Control and Status Register Block Disabled)



2.6. Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge IP

The Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge Intel FPGA IP freezes a PR region Avalon memory-mapped interface when the `freeze` input signal is high. It is recommended that each Avalon memory-mapped interface to a PR region use an instance of the Freeze Bridge IP.

Figure 79. Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge IP

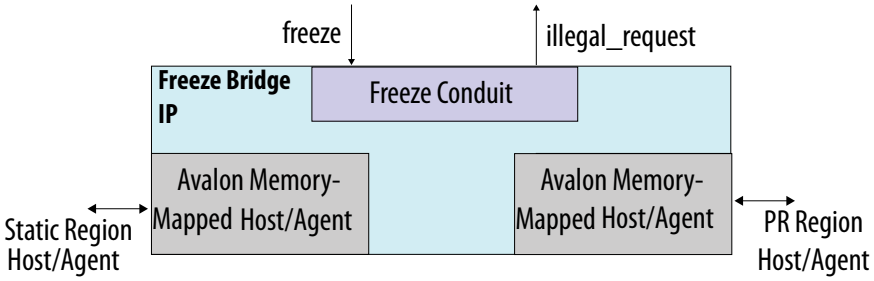


Table 47. Read and Write Request to PR Region Avalon Memory-Mapped Agent Interface

The Freeze Bridge handles read and write transactions differently for each of the following possible interface configurations. The Freeze Bridge is in the freeze state until the PR region or PR region controller asserts the `freeze` signal.

Interface Connection	Behavior
Read request to Avalon memory-mapped slave interface in PR region	1. During the `freeze` state, any read transaction responds with bogus data <h'DEADBEEF>. The corresponding `freeze_illegal_request` register bit sets. 2. During the `freeze` state, `readrequest`, `writerequest`, `waitrequest`, `beginbursttransfer`, `lock`, and `debugaccess` signals in the PR region interface tie low. 3. The Avalon memory-mapped agent response signal constantly returns 2'b10, to indicate an unsuccessful transaction from an endpoint agent. 4. If you disable **Enable Freeze port from PR region**, the IP generates no responses.
Write request to slave interface in PR region	1. The Freeze Bridge ignores any write transactions during the `freeze` state. The Freeze Bridge pulls the `waitrequest`, `beginbursttransfer`, `lock` and `debugaccess` signals low. The IP sets the corresponding `freeze_illegal_request` register bit. 2. The Avalon memory-mapped agent response signal updates with 2'b10 to indicate an unsuccessful transaction from an endpoint agent. 3. If you disable **Enable Freeze port from PR region**, the IP generates no responses.

Table 48. Read and Write Request from PR Region Avalon Memory Mapped Host Interface

Interface Connection	Behavior
Read/Write request from Avalon-MM master interface in PR region (old or new persona)	1. During the `freeze` state, the IP ignores the read and write signals from the PR region. 2. The read and write signals to the static region deassert.

Table 49. Avalon-MM Partial Reconfiguration Freeze Bridge Signal Behavior

The table below summarizes the Avalon interface output signal behavior when the Freeze Bridge is in a frozen state. When not frozen, all signals are just pass-through.

Signal	Agent Bridge	Host Bridge
write	'b0 (tie low)	'b0 (tie low)
read	'b0 (tie low)	'b0 (tie low)
address	Pass through	Pass through
writedata	Pass through	Pass through
readdata	Return <h'DEADBEEF> always	Pass through
byteenable	Pass through	Pass through
burstcount	Pass through	Pass through
beginbursttransfer	'b0 (tie low)	'b0 (tie low)
debugaccess	'b0 (tie low)	'b0 (tie low)
readdatavalid	Return 'b1 when there is a request, else 'b0	Pass through
waitrequest	Return 'b1 when there is a request, else 'b0	'b0 (tie low)
response	Return 'b10 always	Pass through
lock	'b0 (tie low)	'b0 (tie low)
writeresponsevalid	Return 'b1 when there is a request, else 'b0	Pass through

2.6.1. Parameters

The Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge IP core supports customization of the following parameters.

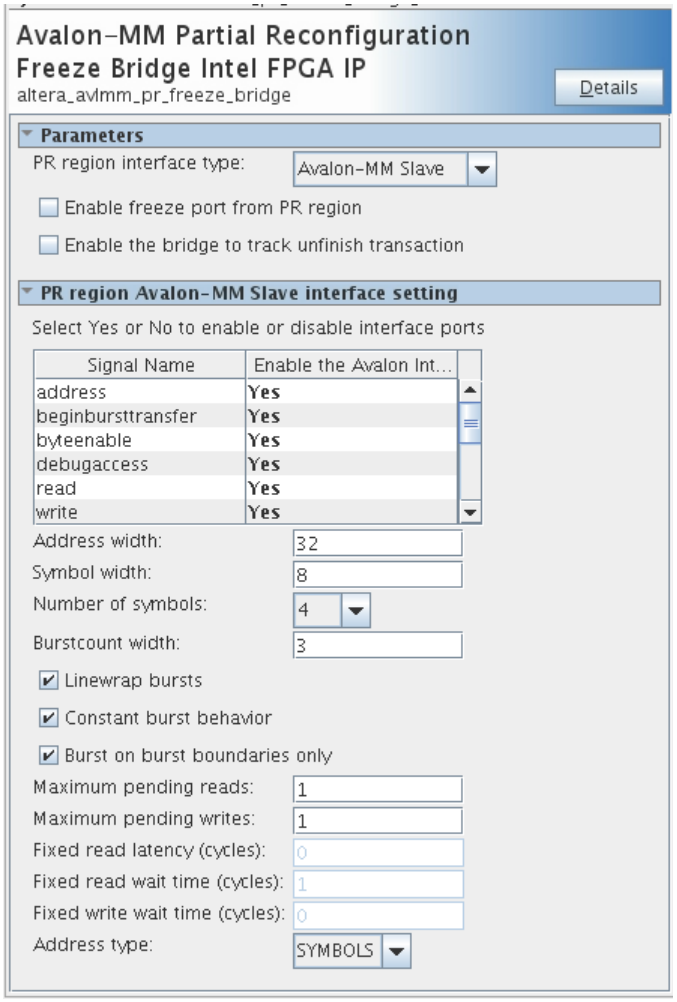
Table 50. Parameters

Parameter	Values	Description
PR region interface Type	**Avalon-MM Slave/Avalon-MM Master**	Specifies the interface type for interfacing the PR region with the Freeze Bridge.
Enable Freeze port from PR region	**On/Off**	Enables the `freeze` port that freezes all the outputs of each PR region to a known constant value. Freezing prevents the signal receivers in the static region from receiving undefined signals during the partial reconfiguration process. The freeze of a bridge is the logical OR of this signal from the PR region, and the freeze from the PR region controller.
Enable the bridge to track unfinished transaction	**On/Off**	Enables the bridge to track unfinished transactions before freezing the Avalon interface. Turn on this option when there is no custom logic to stop the Avalon transaction between the PR region and the static region. If you do not need this feature, disable this option to reduce the size of the IP.
Enabled Avalon Interface Signal	**Yes/No**	Enable (**Yes**) or disable (**No**) specific optional Freeze Bridge interface ports.
Address width	*<1-64>*	Address width in bits.

continued...

Parameter	Values	Description
Symbol width	<*number*>	Data symbol width in bits. The symbol width should be 8 for byte-oriented interfaces.
Number of symbols	<*number*>	Number of symbols per word.
Burstcount width	<*number*>	The width of the burst count in bits.
Linewrap burst	**On**/Off	When **On**, the address for bursts wraps instead of incrementing. With a wrapping burst, when the address reaches a burst boundary, the address wraps back to the previous burst boundary. Consequently, the IP uses only the low order bits for addressing.
Constant burst behavior	**On**/Off	When **On**, memory bursts are constant.
Burst on burst boundaries only	**On**/Off	When **On**, memory bursts are aligned to the address size.
Maximum pending reads	<*number*>	The maximum number of pending reads that the slave can queue.
Maximum pending writes	<*number*>	The maximum number of pending writes that the slave can queue.
Fixed read latency (cycles)	<*number*>	Sets the read latency for fixed-latency slaves. Not useful on interfaces that include the `readdatavalid` signal.
Fixed read wait time (cycles)	<*number*>	For master interfaces that do not use the `waitrequest` signal. The read wait time indicates the number of cycles before the master responds to a read. The timing is as if the master asserted `waitrequest` for this number of cycles.
Fixed write wait time (cycles)	<*number*>	For master interfaces that do not use the `waitrequest` signal. The write wait time indicates the number of cycles before the master accepts a write.
Address type	**WORDS/SYMBOLS**	Sets slave interface address type to symbols or words.

Figure 80. Parameter Editor



2.6.2. Interface Ports

The Avalon Memory-Mapped Partial Reconfiguration Freeze Bridge IP core has the following interface ports.

Table 51. Interface Ports

Port	Width	Direction	Description
`clock`	1	Input	Input clock for the IP.
`reset_n`	1	Input	Synchronous reset for the IP.

continued...

Port	Width	Direction	Description
freeze_conduit_freeze	1	Input	When this signal is high, the bridge handles any current transaction properly then freezes the Avalon memory-mapped PR interfaces.
freeze_conduit_illegal_request	1	Output	High on this bus indicates that an illegal request was issued to the bridge during the freeze state.
pr_freeze_pr_freeze	1	Input	Enabled freeze port coming from the PR region.

Table 52. Avalon Memory-Mapped Agent to PR Region Host Interface Ports

Port	Width	Direction	Description
slv_bridge_to_pr_read	1	Output	Optional Avalon memory-mapped agent bridge to PR region read port.
slv_bridge_to_pr_waitrequest	1	Input	Optional Avalon memory-mapped agent bridge to PR region waitrequest port.
slv_bridge_to_pr_write	1	Output	Optional Avalon memory-mapped agent bridge to PR region write port.
slv_bridge_to_pr_address	32	Output	Optional Avalon memory-mapped agent bridge to PR region address port.
slv_bridge_to_pr_byteenable	4	Output	Optional Avalon memory-mapped agent bridge to PR region byteenable port.
slv_bridge_to_pr_writedata	32	Output	Optional Avalon memory-mapped agent bridge to PR region writedata port.
slv_bridge_to_pr_readdata	32	Input	Optional Avalon memory-mapped agent bridge to PR region readdata port.
slv_bridge_to_pr_burstcount	3	Output	Optional Avalon memory-mapped agent bridge to PR region burstcount port.
slv_bridge_to_pr_readdatavalid	1	Input	Optional Avalon memory-mapped agent bridge to PR region readdatavalid port.
slv_bridge_to_pr_beginbursttransfer	1	Output	Optional Avalon-MM agent bridge to PR region beginbursttransfer port.
slv_bridge_to_pr_debugaccess	1	Output	Optional Avalon memory-mapped agent bridge to PR region debugaccess port.
slv_bridge_to_pr_response	2	Input	Optional Avalon memory-mapped agent bridge to PR region response port.
slv_bridge_to_pr_lock	1	Output	Optional Avalon-MM agent bridge to PR region lock port.
slv_bridge_to_pr_writeresponsevalid	1	Input	Optional Avalon memory-mapped agent bridge to PR region writeresponsevalid port.

Table 53. Avalon Memory-Mapped Agent to Static Region Master Interface Ports

Note: Same setting as Avalon memory-mapped master to PR region agent interface.

Port	Width	Direction	Description
slv_bridge_to_sr_read	1	Input	Avalon memory-mapped agent bridge to static region read port.
slv_bridge_to_sr_waitrequest	1	Output	Avalon memory-mapped agent bridge to static region waitrequest port.
slv_bridge_to_sr_write	1	Input	Avalon memory-mapped agent bridge to static region write port.
slv_bridge_to_sr_address	32	Input	Avalon memory-mapped agent bridge to static region address port.
slv_bridge_to_sr_byteenable	4	Input	Avalon memory-mapped agent bridge to static region byteenable port.
slv_bridge_to_sr_writedata	32	Input	Avalon memory-mapped agent bridge to static region writedata port.
slv_bridge_to_sr_readdata	32	Output	Avalon memory-mapped agent bridge to static region readdata port.
slv_bridge_to_sr_burstcount	3	Input	Avalon memory-mapped agent bridge to static region burstcount port.
slv_bridge_to_sr_beginbursttransfer	1	Input	Avalon memory-mapped agent bridge to static region beginbursttransfer port.
slv_bridge_to_sr_debugaccess	1	Input	Avalon-MM agent bridge to static region debugaccess port.
slv_bridge_to_sr_response	2	Output	Avalon memory-mapped agent bridge to static region response port.
slv_bridge_to_sr_lock	1	Input	Avalon memory-mapped agent bridge to static region lock port.
slv_bridge_to_sr_writeresponsevalid	1	Output	Avalon memory-mapped agent bridge to static region writereponsevalid port.

Table 54. Avalon Memory-Mapped Master to PR Region Agent Interface Ports

Port	Width	Direction	Description
mst_bridge_to_pr_read	1	Input	Optional Avalon memory-mapped master bridge to PR region read port.
mst_bridge_to_pr_waitrequest	1	Output	Optional Avalon memory-mapped master bridge to PR region waitrequest port.
mst_bridge_to_pr_write	1	Input	Optional Avalon memory-mapped master bridge to PR region write port.
mst_bridge_to_pr_address	32	Input	Optional Avalon memory-mapped master bridge to PR region address port.
mst_bridge_to_pr_byteenable	4	Input	Optional Avalon-MM master bridge to PR region byteenable port.
mst_bridge_to_pr_writedata	32	Input	Optional Avalon-MM master bridge to PR region writedata port.

continued...

Port	Width	Direction	Description
mst_bridge_to_pr_readdata	32	Output	Optional Avalon memory-mapped master bridge to PR region readdata port.
mst_bridge_to_pr_burstcount	3	Input	Optional Avalon memory-mapped master bridge to PR region burstcount port.
mst_bridge_to_pr_readdatavalid	1	Output	Optional Avalon memory-mapped master bridge to PR region readdatavalid port.
mst_bridge_to_pr_beginbursttransfer	1	Input	Optional Avalon memory-mapped master bridge to PR region beginbursttransfer port.
mst_bridge_to_pr_debugaccess	1	Input	Optional Avalon memory-mapped master bridge to PR region debugaccess port.
mst_bridge_to_pr_response	2	Output	Optional Avalon memory-mapped master bridge to PR region response port.
mst_bridge_to_pr_lock	1	Input	Optional Avalon memory-mapped master bridge to PR region lock port.
mst_bridge_to_pr_writeresponsevalid	1	Output	Optional Avalon memory-mapped master bridge to PR region writeresponsevalid port.

Table 55. Avalon Memory-Mapped Master to Static Region Agent Interface Ports

Same setting as Avalon Memory-Mapped agent to PR region master interface.

Port	Width	Direction	Description
mst_bridge_to_sr_read	1	Output	Avalon memory-mapped master bridge to static region read port.
mst_bridge_to_sr_waitrequest	1	Input	Avalon memory-mapped bridge to static region waitrequest port.
mst_bridge_to_sr_write	1	Output	Avalon memory-mapped master bridge to static region write port.
mst_bridge_to_sr_address	32	Output	Avalon memory-mapped master bridge to static region address port.
mst_bridge_to_sr_byteenable	4	Output	Avalon memory-mapped master bridge to static region byteenable port.
mst_bridge_to_sr_writedata	32	Output	Avalon memory-mapped master bridge to static region writedata port.
mst_bridge_to_sr_readdata	32	Input	Avalon memory-mapped master bridge to static region readdata port.
mst_bridge_to_sr_burstcount	3	Output	Avalon memory-mapped master bridge to static region burstcount port.
mst_bridge_to_sr_readdatavalid	1	Input	Avalon memory-mapped master bridge to static region readdatavalid port.
mst_bridge_to_sr_beginbursttransfer	1	Output	Avalon memory-mapped master bridge to static region beginbursttransfer port.
mst_bridge_to_sr_debugaccess	1	Output	Avalon memory-mapped master bridge to static region debugaccess port.

continued...

Port	Width	Direction	Description
mst_bridge_to_sr_response	2	Input	Avalon memory-mapped master bridge to static region response port.
mst_bridge_to_sr_lock	1	Output	Avalon memory-mapped master bridge to static region lock port.
mst_bridge_to_sr_writeresponsevalid	1	Input	Avalon memory-mapped master bridge to static region writeresponsevalid port.

Figure 81. Avalon Memory-Mapped Host Interface Ports

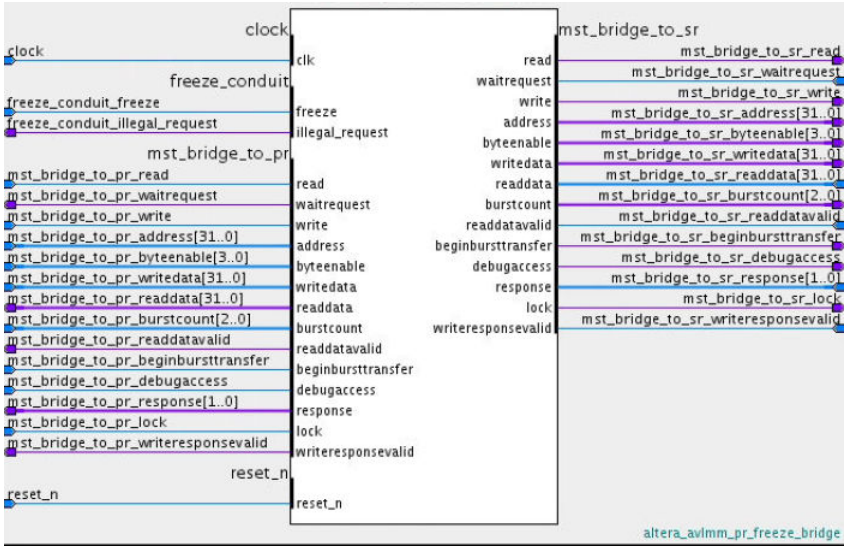
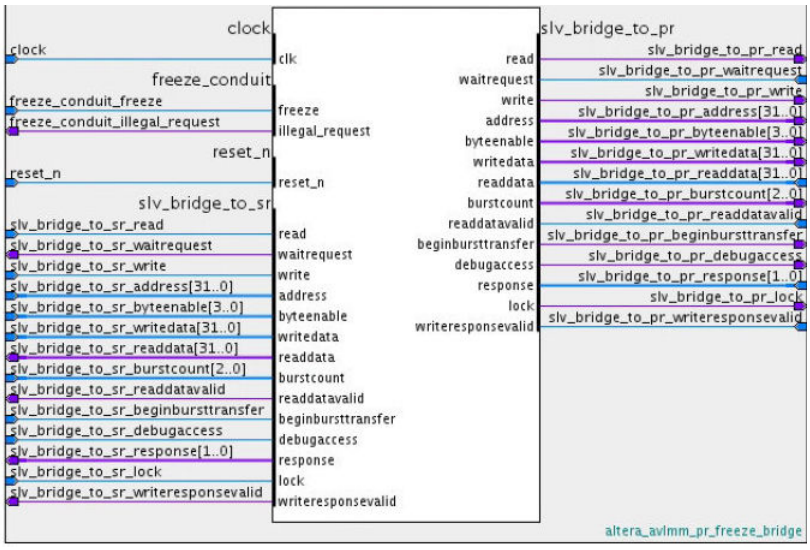


Figure 82. Avalon Memory-Mapped Agent Interface Ports



2.7. Avalon Streaming Partial Reconfiguration Freeze Bridge IP

The Avalon Streaming Partial Reconfiguration Freeze Bridge Intel FPGA IP freezes a PR region Avalon streaming interface when the `freeze` input signal is high. The Avalon Streaming Partial Reconfiguration Freeze Bridge IP ensures that any transaction is complete before freezing the connected interface. It is recommended that each Avalon streaming interface to a PR region use an instance of the Freeze Bridge IP.

Figure 83. Avalon Streaming Partial Reconfiguration Freeze Bridge

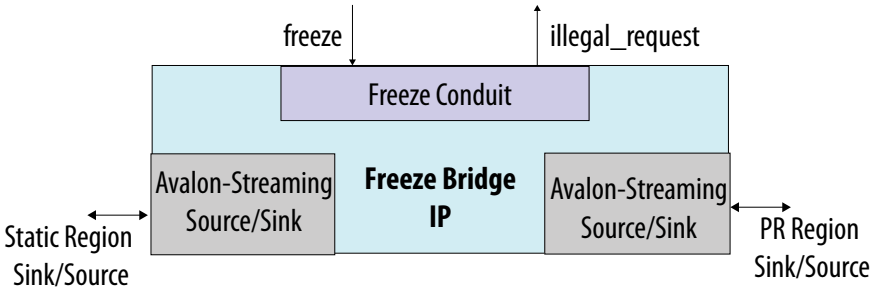


Table 56. Avalon Streaming Source Freeze Bridge Interface Behavior

Interface Type	Behavior
Source interface in the PR region with packet transfer (old or new persona)	1. When the `freeze` signal goes high, the Freeze Bridge handles the `startofpacket`, `endofpacket`, and `empty` bits and does not send transactions to the static region. 2. When the Freeze Bridge detects a `startofpacket` transaction without a corresponding `endofpacket` during the frozen state, this indicates an unfinished transaction. 3. The bridge then completes the transaction by asserting `valid` and `endofpacket` high to the static region for one clock cycle. 4. The `channel` signal remains constant, while data bits are set to `'hDEADBEEF` and error bit is set to `1'b1`. 5. The `illegal_request` output signal triggers update of the CSR register in the Partial Reconfiguration Region Controller.
Source interface in the PR region without packet transfer (old or new persona)	When the `freeze` signal is high, the Freeze Bridge does not send transactions to the static region. The Freeze Bridge remains idle until the bridge leaves the frozen state.
Source interface in the PR region with max_channel > 1 (old or new persona)	When multiple channels transfer unfinished transactions, the Freeze Bridge tracks the `channel` values to ensure that all packet transactions from different channels end by asserting the `endofpacket` bit during the frozen state.
Source interface in the PR region with ready_latency > 0 (old or new persona)	When the Freeze Bridge drives `endofpacket`, `valid`, or `channel` outputs to the static region, the Freeze Bridge reads the `ready_latency` value. The `ready_latency` value defines the actual clock cycle when the sink component is ready for data.

Figure 84. Source Bridge Handling of Unfinished Packet Transaction During Freeze

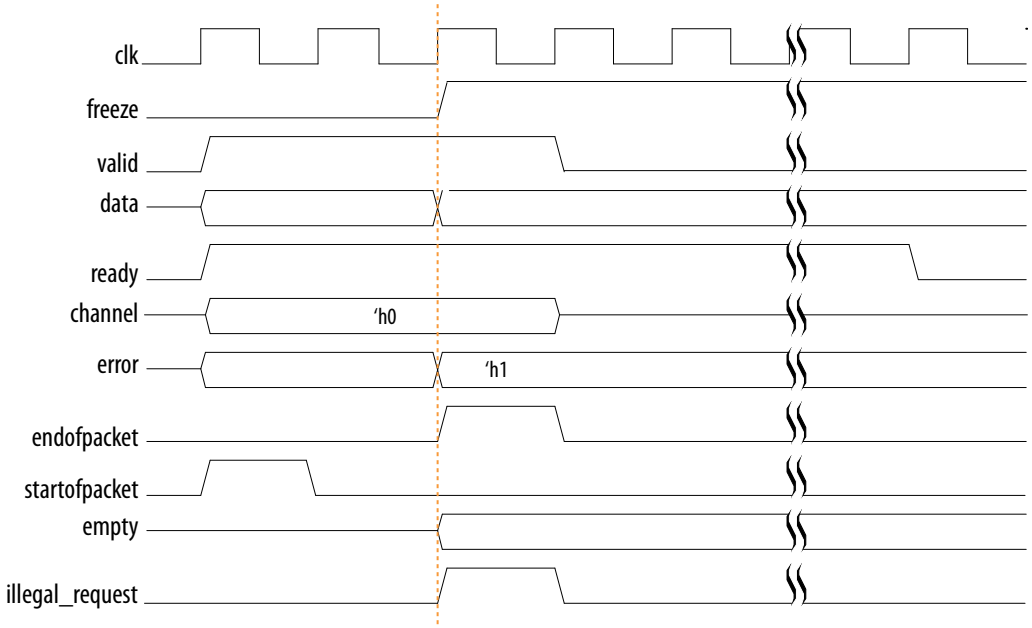


Figure 85. PR Freeze Bridge Asserting valid Signal to End Packet Transactions

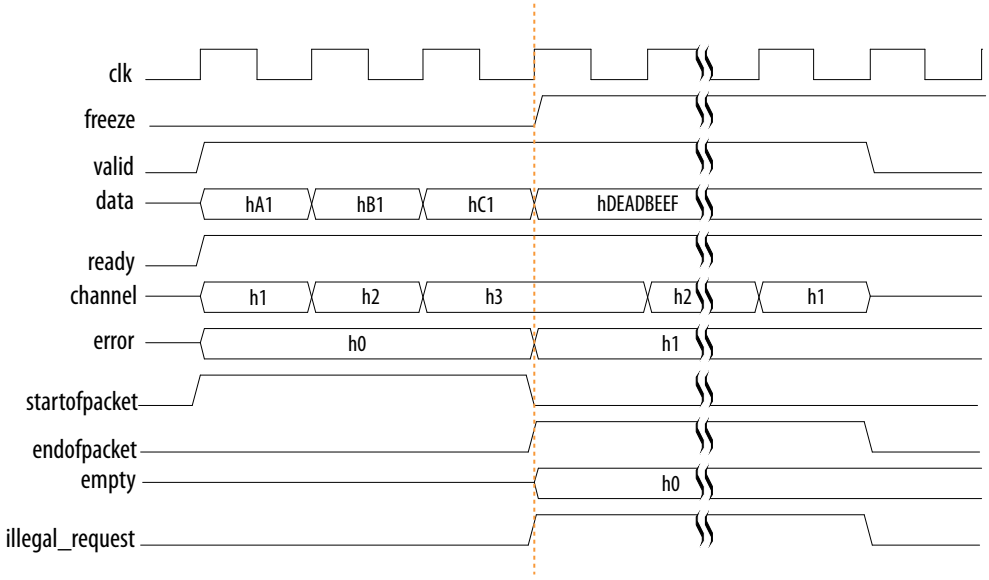


Table 57. Avalon Streaming Sink Freeze Bridge Interface Behavior

Interface Type	Behavior
Sink interface in PR region	For transactions that includes packet transfers, when the `freeze` signal goes high, the Freeze Bridge holds the `ready` signal high to the static region source until any unfinished transaction completes.

continued...

Interface Type	Behavior
	For transactions that do not include packet transfers, when the `freeze` signal goes high, the Freeze Bridge holds the `ready` signal low during the freeze period. The `illegal_request` signal asserts high to indicate that the current transaction is an error. Configure the design to stop sending transactions to the PR region after the `illegal_request` signal is high.
Sink interface in PR region with ready_latency > 0	When the Freeze Bridge drives `endofpacket`, `valid`, or `channel` outputs to the PR region, the Freeze Bridge must observe the `ready_latency` value. The `ready_latency` value defines the actual clock cycle when the sink component is ready for data.

2.7.1. Parameters

The Avalon Streaming Partial Reconfiguration Freeze Bridge IP core supports customization of the following parameters:

Figure 86. Parameter Editor

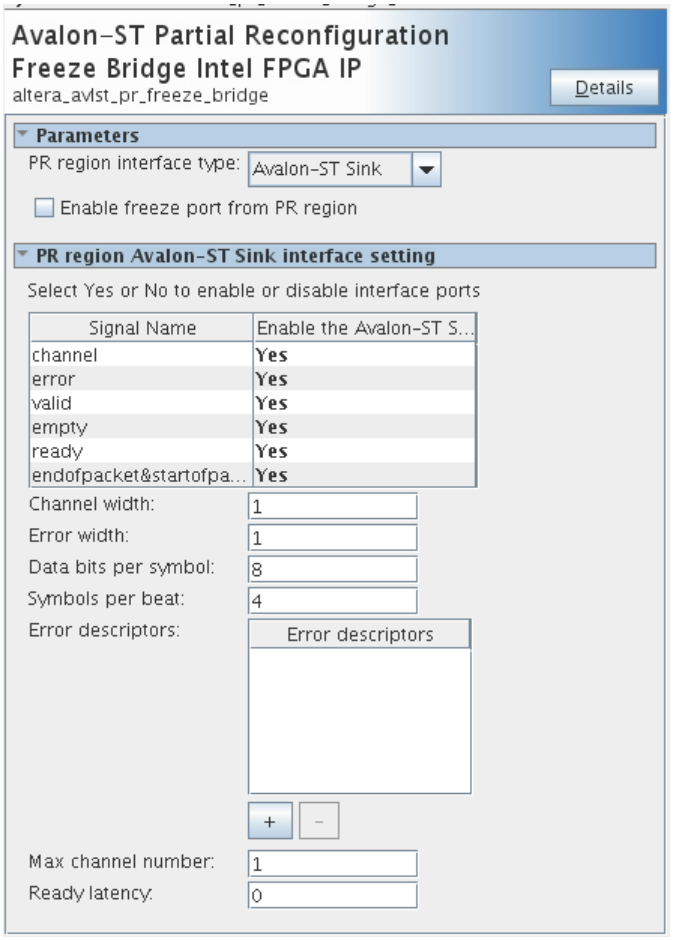


Table 58. Parameters

Parameter	Values	Description
PR region Interface Type	**Avalon-ST Source/Avalon-ST Sink**	Specifies the interface type for interfacing the PR region with the freeze bridge.
Enable Freeze port from PR region	**On/Off**	Enables the freeze port to freeze all the outputs of each PR region to a known constant value. Freezing prevents the signal receivers in the static region from receiving undefined signals during the partial reconfiguration process.
Select Yes or No to enable or disable interface ports	**Yes/No**	Enables or disables specific optional freeze bridge interface ports.
Channel width	*<1-128>*	Specifies the width of the channel signal.
Error width	*<1-256>*	Specifies the width of the error signal.
Data bits per symbol	*<1-512>*	Specifies the number of bits per symbol.
Symbols per beat	*<1-512>*	Specifies the number of symbols that transfer on every valid clock cycle.
Error descriptors	*<text>*	Specifies one or more strings to describe the error condition for each bit of the error port on the sink interface connected to the source interface. Click the plus or minus buttons to add or remove descriptors.
Max channel number	*<0-255>*	Specifies the maximum number of output channels.
Ready latency	*<0-8>*	Specifies what ready latency to expect from the source interface connected to the sink interface. The ready latency is the number of cycles from the time `ready` asserts until valid data is driven.

2.7.2. Ports

The Avalon Streaming Partial Reconfiguration Freeze Bridge IP has the following ports:

Figure 87. Avalon Streaming Sink Interface Ports

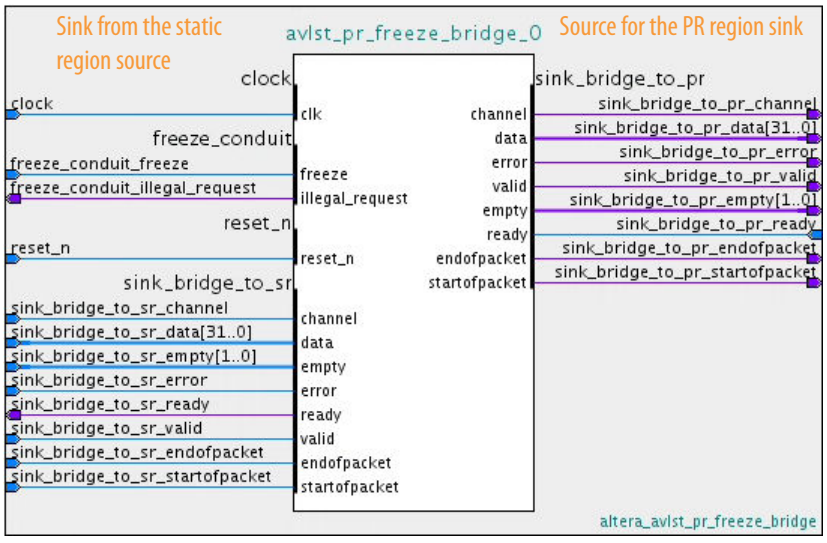


Figure 88. Avalon Streaming Source Interface Ports

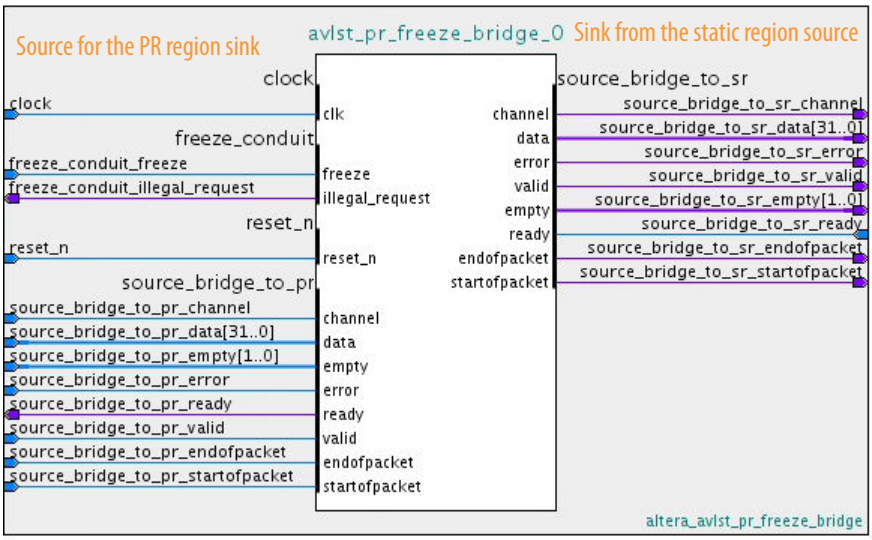


Table 59. Avalon Streaming Interface Ports

Port	Width	Direction	Description
clock	1	Input	Input clock for the IP.
freeze_conduit_freeze	1	Input	When this signal is high, the bridge handles any current transaction properly then freezes the PR interfaces.
freeze_conduit_illegal_request	1	Output	High on this bus indicates that an illegal request was issued to the bridge during the freeze state. *n* – number of freeze bridge
pr_freeze_pr_freeze	1	Input	Enabled freeze port from the PR region.
reset_n	1	Input	Synchronous reset for the IP.

Table 60. Avalon Streaming Sink to Static Region Interface Ports

Same setting as Avalon streaming sink to PR region interface.

Port	Width	Direction	Description
sink_bridge_to_sr_channel	1	Input	Avalon streaming sink bridge to static region channel port.
sink_bridge_to_sr_data	32	Input	Avalon streaming sink bridge to static region data port.
sink_bridge_to_sr_empty	2	Input	Avalon streaming sink bridge to static region empty port.
sink_bridge_to_sr_error	1	Input	Avalon streaming sink bridge to static region error port.
sink_bridge_to_sr_ready	1	Output	Avalon streaming sink bridge to static region ready port.

continued...

Port	Width	Direction	Description
sink_bridge_to_sr_valid	1	Input	Avalon streaming sink bridge to static region valid port.
sink_bridge_to_sr_endofpacket	1	Input	Avalon streaming sink bridge to static region endofpacket port.
sink_bridge_to_sr_startofpacket	1	Input	Avalon streaming sink bridge to static region startofpacket port.

Table 61. Avalon-Streaming Sink to PR Region Interface Ports

Port	Width	Direction	Description
sink_bridge_to_pr_channel	1	Output	Optional Avalon streaming sink bridge to PR region channel port.
sink_bridge_to_pr_data	32	Output	Optional Avalon streaming sink bridge to PR region data port.
sink_bridge_to_pr_empty	2	Output	Optional Avalon streaming sink bridge to PR region empty port.
sink_bridge_to_pr_error	1	Output	Optional Avalon streaming sink bridge to PR region error port.
sink_bridge_to_pr_ready	1	Input	Optional Avalon-ST sink bridge to PR region ready port.
sink_bridge_to_pr_valid	1	Output	Optional Avalon streaming sink bridge to PR region valid port.
sink_bridge_to_pr_endofpacket	1	Output	Optional Avalon streaming sink bridge to PR region endofpacket port.
sink_bridge_to_pr_startofpacket	1	Output	Optional Avalon streaming sink bridge to PR region startofpacket port.

Table 62. Avalon Streaming Source to Static Region Interface Ports

Same setting as Avalon streaming source to PR region interface.

Port	Width	Direction	Description
source_bridge_to_sr_channel	1	Output	Avalon streaming source bridge to static region channel port.
source_bridge_to_sr_data	32	Output	Avalon streaming source bridge to static region data port.
source_bridge_to_sr_empty	2	Output	Avalon streaming source bridge to static region empty port.
source_bridge_to_sr_error	1	Output	Avalon streaming source bridge to static region error port.
source_bridge_to_sr_ready	1	Input	Avalon streaming source bridge to static region ready port.
source_bridge_to_sr_valid	1	Output	Avalon streaming source bridge to static region valid port.
source_bridge_to_sr_endofpacket	1	Output	Avalon streaming source bridge to static region endofpacket port.
source_bridge_to_sr_startofpacket	1	Output	Avalon streaming source bridge to static region startofpacket port.

Table 63. Avalon Streaming Source to PR Region Interface Ports

Port	Width	Direction	Description
source_bridge_to_pr_channel	1	Input	Optional Avalon streaming source bridge to PR region channel port.
source_bridge_to_pr_data	32	Input	Optional Avalon streaming source bridge to PR region data port.
source_bridge_to_pr_empty	2	Input	Optional Avalon streaming source bridge to PR region empty port.
source_bridge_to_pr_error	1	Input	Optional Avalon-ST source bridge to PR region error port.
source_bridge_to_pr_ready	1	Output	Optional Avalon streaming source bridge to PR region ready port.
source_bridge_to_pr_valid	1	Input	Optional Avalon streaming source bridge to PR region valid port.
source_bridge_to_pr_endofpacket	1	Input	Optional Avalon streaming source bridge to PR region endofpacket port.
source_bridge_to_pr_startofpacket	1	Input	Optional Avalon streaming source bridge to PR region startofpacket port.

2.8. Generating and Simulating Intel FPGA IP

Use the following information to generate and simulate an IP core variation.

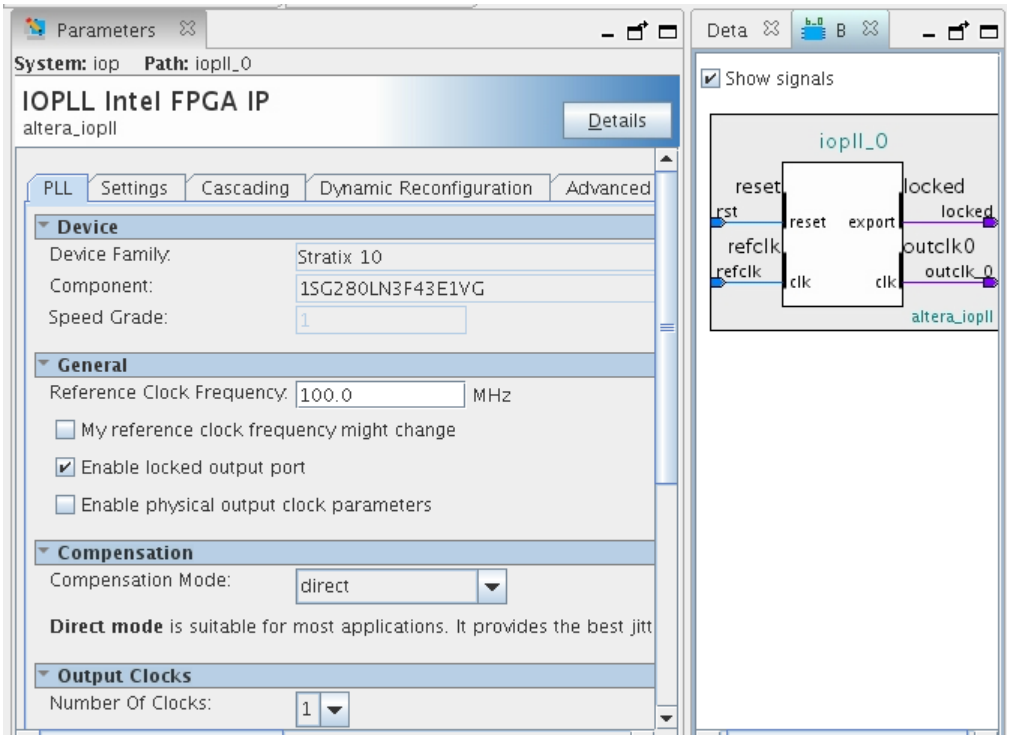
2.8.1. Specifying the IP Core Parameters and Options (Quartus Prime Pro Edition)

Quickly configure Intel FPGA IP cores in the Quartus Prime parameter editor. Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to define a custom variation of the IP core. The parameter editor generates the IP variation synthesis and optional simulation files, and adds the `.ip` file representing the variation to your project automatically.

Follow these steps to locate, instantiate, and customize an IP core in the parameter editor:

1. Create or open an Quartus Prime project (`.qpf`) to contain the instantiated IP variation.
2. In the IP Catalog (**Tools ➤ IP Catalog**), locate and double-click the name of the IP core to customize. To locate a specific component, type some or all of the component's name in the IP Catalog search box. The New IP Variation window appears.
3. Specify a top-level name for your custom IP variation. Do not include spaces in IP variation names or paths. The parameter editor saves the IP variation settings in a file named <*your_ip*>.ip. Click **OK**. The parameter editor appears.

Figure 89. IP Parameter Editor (Quartus Prime Pro Edition)



4. Set the parameter values in the parameter editor and view the block diagram for the component. The **Parameterization Messages** tab at the bottom displays any errors in IP parameters:
   - Optionally, select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for processing the IP core files in other EDA tools.

   Note: Refer to your IP core user guide for information about specific IP core parameters.
5. Click **Generate HDL**. The **Generation** dialog box appears.
6. Specify output file generation options, and then click **Generate**. The synthesis and simulation files generate according to your specifications.
7. To generate a simulation testbench, click **Generate ➤ Generate Testbench System**. Specify testbench generation options, and then click **Generate**.
8. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate ➤ Show Instantiation Template**.
9. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project.
10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

Note: Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script*.

Related Information

Introduction to Intel FPGA IP Cores

2.8.2. Running the Freeze Bridge Update script

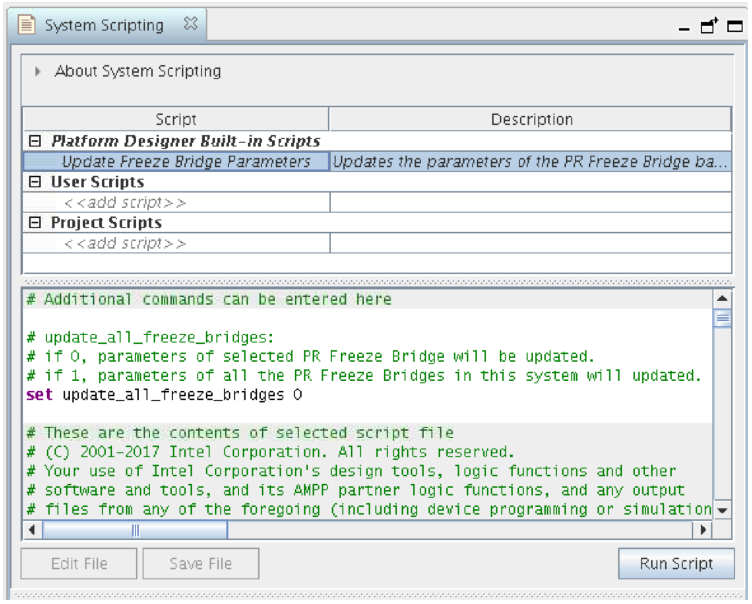
When instantiating the Freeze Bridge as a Platform Designer system component, the interface connections between the Freeze Bridge and the PR region must match, so that Platform Designer inserts no extra interconnect during system generation. Rather than manually matching the Avalon interface properties individually in the parameter editor, you can run the provided Update Freeze Bridge Parameters script to update Freeze Bridge Avalon interface properties automatically.

Running this script updates the host and agent interfaces or the sink and source interfaces of the Freeze Bridge, according to the Avalon property settings of the connecting PR region component.

To run the Update Freeze Bridge Parameters script:

1. Open a Platform Designer system containing one or more instances of the Freeze Bridge component.
2. In Platform Designer, click **View ➤ System Scripting**. The **System Scripting** tab displays **Platform Designer Built-in Scripts**.
3. To update all freeze bridges in your Platform Designer system, set `update_all_freeze_bridges` to 1 in the Additional Commands section of the script. To update only a single freeze bridge, click the freeze bridge instance.
4. Click **Run Script**. The script runs and updates the freeze bridge parameters.

Figure 90. Platform Designer System Scripting Tab



2.8.3. IP Core Generation Output (Quartus Prime Pro Edition)

The Quartus Prime software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.

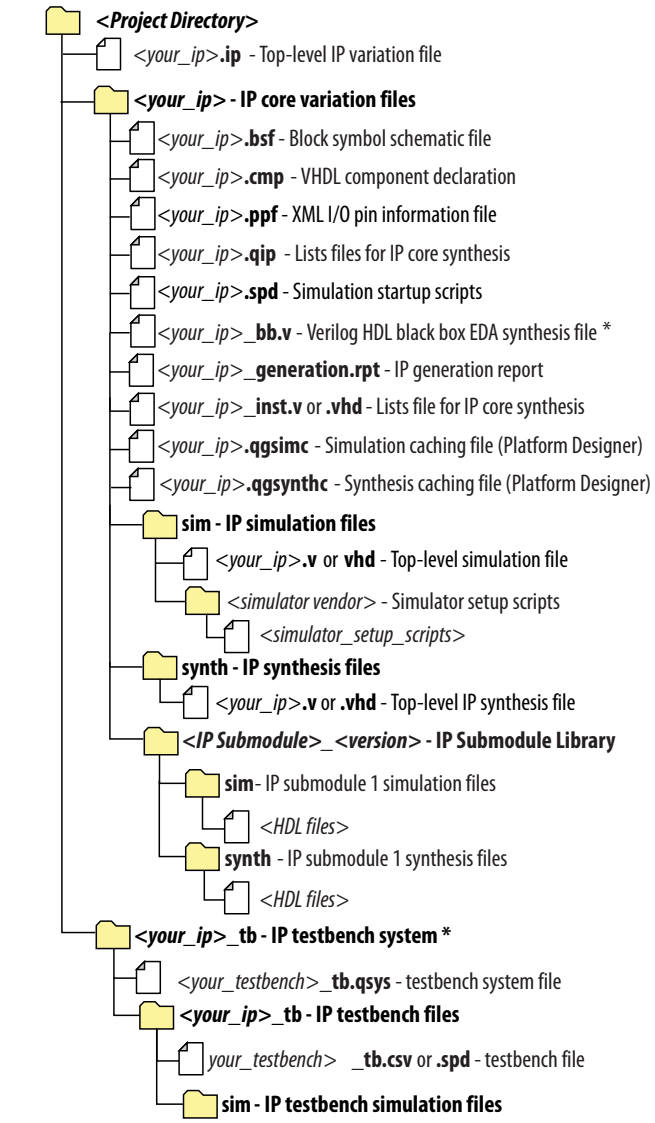
Table 64. Output Files of Intel FPGA IP Generation

File Name	Description
<*your_ip*>.ip	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a .qsys file.
<*your_ip*>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you use in VHDL design files.
<*your_ip*>_generation.rpt	IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.
<*your_ip*>.qgsimc (Platform Designer systems only)	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qgsynth (Platform Designer systems only)	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.csv	Contains information about the upgrade status of the IP component.
<*your_ip*>.bsf	A symbol representation of the IP variation for use in Block Diagram Files (.bdf).
<*your_ip*>.spd	Input file that ip-make-simscript requires to generate simulation scripts. The .spd file contains a list of files you generate for simulation, along with information about memories that you initialize.
<*your_ip*>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components you create for use with the Pin Planner.

continued...

File Name	Description
<*your_ip*>_bb.v	Use the Verilog blackbox (_bb.v) file as an empty module declaration for use as a blackbox.
<*your_ip*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*your_ip*>.regmap	If the IP contains register information, the Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of host and agent interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*your_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Quartus Prime software stores the .svd files for agent interface visible to the System Console hosts in the .sof file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system agents, Platform Designer accesses the registers by name.
<*your_ip*>.v <*your_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a msim_setup.tcl script to set up and run a simulation with a supported Siemens EDA simulator, such as the QuestaSim simulator.
aldec/	Contains a Riviera-PRO* script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS* simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file to set up and run a VCS MX simulation.
/xcelium	Contains an Xcelium* Parallel simulator shell script xcelium_setup.sh and other setup files to set up and run a simulation.
/submodules	Contains HDL files for the IP core submodule.
<*IP submodule*>/	Platform Designer generates /synth and /sim sub-directories for each IP submodule directory that Platform Designer generates.

Figure 91. Individual IP Core Generation Output (Quartus Prime Pro Edition)



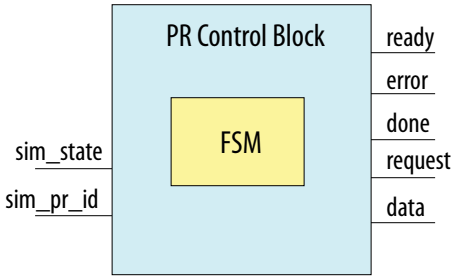
* If supported and enabled for your IP core variation.

2.8.4. Arria 10 and Cyclone 10 GX PR Control Block Simulation Model

The Quartus Prime Pro Edition software supports simulating the delivery of a partial reconfiguration bitstream to the PR control block. This simulation allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

The Arria 10 and Cyclone 10 GX PR control blocks support PR simulation. Sending a simulation RBF (PR bitstream) allows the PR control block to behave accordingly, to PR simulation success or PR simulation failure. To activate simulation of a specific PR persona in your PR region simulation wrapper, use a PR ID encoded in the simulation RBF, in conjunction with the PR control block. Simulate the PR control block either as standalone, or as part of the simulation file set for the Partial Reconfiguration Controller IP core.

Figure 92. PR Control Block Simulation Model



The PR control block simulation model contains two additional simulation-only ports—`sim_state` and `sim_pr_id`. Connect these simulation ports, and the other ports, to the `twentynm_prblock_if` SystemVerilog interface. This connection allows monitoring of the PR control block using your testbench's PR control block monitor. The Quartus Prime software automatically instantiates the `twentynm_prblock_if` interface when generating the simulation file set of the Partial Reconfiguration IP core. Obtain a reference to the `twentynm_prblock_if` that the IP instantiates by using the `alt_pr_test_pkg::twentynm_prblock_if_mgr` singleton, as shown in the following example:

```
virtual twentynm_prblock_if prblock_if;

alt_pr_test_pkg::twentynm_prblock_if_mgr cb_mgr;

// Get the PR control block from the prblock manager
cb_mgr = alt_pr_test_pkg::twentynm_prblock_if_mgr::get();
prblock_if = cb_mgr.if_ref;
```

The code for the `twentynm_prblock_if` interface is as follows:

```
interface twentynm_prblock_if(input logic pr_clk, input logic clk);

    logic prrequest;
    logic [31:0] data;
    wire error;
    wire ready;
    wire done;
    logic [31:0] sim_only_state;
    wire [31:0] sim_only_pr_id;

    // All signals are async except data
    clocking cb1 @(posedge pr_clk);
        output data;
    endclocking

endinterface : twentynm_prblock_if
```

For more information on the `twentynm_prblock_if` interface, refer to the *<installation directory>*`/eda/sim_lib/altera_lnsim.sv` file.

The simulation state of the PR control block simulation model represents the PR_EVENT_TYPE enumeration state of the control block. The `twentynm_prblock_test_pkg` SystemVerilog package defines these enumerations. These states represent the different allowed states for the control block. The defined control block enumerations are:

```
package twentynm_prblock_test_pkg;
    typedef enum logic [31:0] {
        NONE,
        IDLE,
```

```
        PR_REQUEST,
        PR_IN_PROGRESS,
        PR_COMPLETE_SUCCESS,
        PR_COMPLETE_ERROR,
        PR_INCOMPLETE_EARLY_WITHDRAWL,
        PR_INCOMPLETE_LATE_WITHDRAWL
    } PR_EVENT_TYPE;
```

When the simulation state is PR_IN_PROGRESS, the affected PR region must have its simulation output multiplexes driven to X, by asserting the pr_activate signal. This action simulates the unknown outputs of the PR region during partial reconfiguration. In addition, you must assert the pr_activate signal in the PR simulation model to load all registers in the PR model with the PR activation value.

Once the simulation state reaches PR_COMPLETE_SUCCESS, activate the appropriate PR persona using the appropriate PR region simulation wrapper mux sel signals. You can decode the region, as well as the specific select signal from the sim_only_pr_id signal of the PR control block. This ID corresponds to the encoded ID in the simulation RBF.

Table 65. Required Sequence of Words in Simulation RBF

Step 1 writes zero or more of the following words. All other steps write only 1 word.

1	zero padding blocks	0x00000000
2	PR_HEADER_WORD	0x0000A65C
3	PR_ID	32-bit user ID
4	PRDATA_COUNT_0	0x01234567
5	PRDATA_COUNT_1	0x89ABCDEF
6	PRDATA_COUNT_2	0x02468ACE
7	PRDATA_COUNT_3	0x13579BDF

Note: The PR_ID word is output on the sim_only_pr_id word, starting at PRDATA_COUNT_0. Using a different value for the header or data count results in PR simulation errors.

Related Information

Simulating PR Persona Replacement on page 53

2.8.5. Generating the PR Persona Simulation Model

Use the Quartus Prime EDA Netlist Writer to create the simulation model for a PR persona. The simulation model represents the post-synthesis, gate-level netlist for the persona.

When using the PR simulation model for the persona, the netlist includes a new altera_sim_pr_activate top-level signal for the model. You can asynchronously drive this signal to load all registers in the model with X. This feature allows you to verify the reset sequence of the new persona on PR event completion. Verify the reset sequence through inspection, using SystemVerilog assertions, or using other checkers.

By default, the PR simulation model asynchronously loads X into the register's storage element on `pr_activate` signal assertion. You can parameterize this behavior on a per register basis, or on a simulation-wide default basis. The simulation model supports four built-in modes:

- `load X`
- `load 1`
- `load 0`
- `load rand`

Specify these modes using the SystemVerilog classes:

- `dffeas_pr_load_x`
- `dffeas_load_1`
- `dffeas_load_0`
- `dffeas_load_rand`

Optionally, you can create your own PR activation class, where your class must define the `pr_load` variable to specify the PR activation value.

Follow these steps to generate the simulation model for a PR design:

1. Open the base revision of a PR project in Quartus Prime Pro Edition, and then click **Processing ➤ Start ➤ Start Analysis & Synthesis**. Alternatively, run this command-line equivalent:

```
quartus_syn <project name> -c <base revision name>
```

2. After synthesis is complete, click **Project ➤ Export Design Partition**, and then select the **root partition** for the **Partition name**, and select **synthesized** for the **Snapshot**. Click **OK**. Alternatively, run this command-line equivalent:

```
quartus_cdb <project name> -c <base revision name> \
     "--export_block root_partition --snapshot synthesized \
     --file <static qdb name>
```

3. Click **Project ➤ Revisions** and switch the current revision to that of the persona you want to export.
4. Click **Processing ➤ Start ➤ Start Analysis & Synthesis**. Alternatively, run this command-line equivalent:

```
quartus_syn <project name> -c <persona revision name>
```

5. After synthesis of the persona revision completes, execute the following at the command line to generate the PR simulation model:

```
quartus_eda <project name> –c <persona revision name> "--pr --simulation \
     --tool=modelsim --format=verilog --partition=<pr partition name> \
     --module=<partition name>=<persona module name>
```

6. Repeat steps 3 through 5 for all personas that you want to simulate.

Example 6. Complete PR Simulation Model Generation Script

```
quartus_syn <project name> -c <base revision name>
quartus_cdb <project name> -c <base revision name> \
     "--export_block root_partition --snapshot synthesized \
```

```
     --file <static qdb name>
quartus_syn <project name> -c <persona revision name>
quartus_eda <project name> –c <persona revision name> \
     "--pr --simulation --tool=modelsim --format=verilog \
     --partition=<pr partition name> --module=<partition name>=\
     <persona module name>
```

You can use the PR mode of the EDA netlist writer to generate the gate level netlist of a PR region. Refer to the "EDA Netlist Writer and Gate Level-Netlists" section of the *Quartus Prime Pro Edition User Guide: Third Party Simulation*.

Related Information

Quartus Prime Pro Edition User Guide: Third Party Simulation

2.8.6. Secure Device Manager Partial Reconfiguration Simulation Model

The Quartus Prime Pro Edition software supports simulating the delivery of a partial reconfiguration bitstream to the Agilex 7, Agilex 5, and Stratix 10 FPGA's secure device manager (SDM).

The SDM is a self-contained system-on-chip that securely manages the boot and configuration process, provides secure key storage, enforces security policies, and provides security services during runtime for Agilex 7, Agilex 5, and Stratix 10 devices. Simulation with the SDM allows you to observe the resulting change and the intermediate effect in a reconfigurable partition.

When you send a simulation RBF (PR bitstream) to the SDM, the SDM can respond with a PR simulation success or PR simulation failure message.

To activate simulation of a specific PR persona in your PR region simulation wrapper, you use a PR ID encoded in the simulation RBF, in conjunction with the SDM.

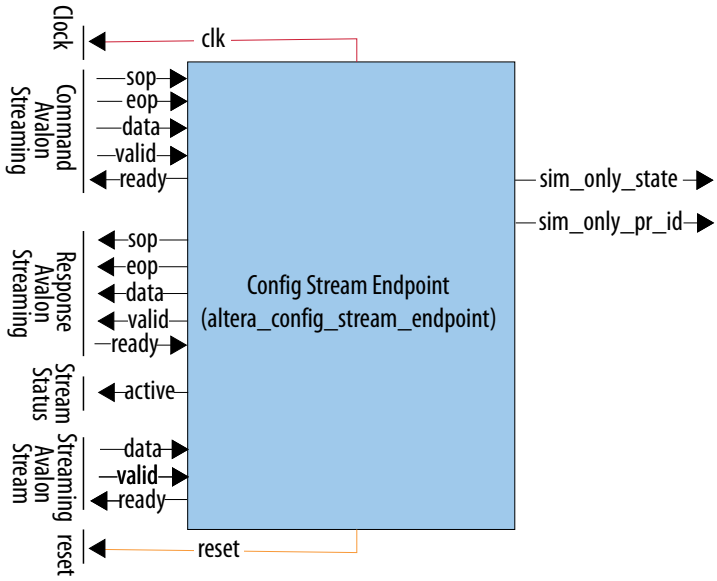
You simulate the SDM as part of the simulation file set for the Partial Reconfiguration Controller Intel FPGA IP and the Partial Reconfiguration External Configuration Controller IP that is available for Agilex 7, Agilex 5, and Stratix 10 devices.

2.8.6.1. Monitoring the SDM

The SDM simulation model exposes the following additional simulation-only ports through the `altera_config_stream_endpoint` module:

- `sim_only_state`
- `sim_only_pr_id`

Figure 93. SDM Partial Reconfiguration Simulation Model - altera_config_stream_endpoint Module



You connect these simulation ports to the `config_stream_endpoint_pr_if` SystemVerilog interface. This connection allows monitoring of the SDM using your testbench's SDM monitor.

The Quartus Prime software automatically instantiates the `config_stream_endpoint_pr_if` interface when generating the simulation file set of the Partial Reconfiguration Controller and the Partial Reconfiguration External Configuration Controller IP.

You can obtain a reference to the `config_stream_endpoint_pr_if` that the IP instantiates by using the following singleton:

```
intel_pr_mailbox_test_pkg::config_stream_endpoint_pr_if_mgr
```

The following example shows this reference:

```
virtual config_stream_endpoint_pr_if pr_mailbox_if

intel_pr_mailbox_test_pkg::config_stream_endpoint_pr_if_mgr pr_mbox_mgr;

// Get the PR Config Stream Endpoint from the pr_mbox manager
pr_mbox_mgr = intel_pr_mailbox_test_pkg::config_stream_endpoint_pr_if_mgr\
   ::get();
pr_mailbox_if = pr_mbox_mgr.if_ref;
```

The following is the code for the `config_stream_endpoint_pr_if` interface:

```
interface config_stream_endpoint_pr_if (input logic clk);
    wire [31:0] sim_only_state;
    wire [31:0] sim_only_pr_id;
endinterface : config_stream_endpoint_pr_if
```

Refer to the following file for more information on the `config_stream_endpoint_pr_if` interface:

```
<installation directory>/eda/sim_lib/altera_lnsim.sv
```

The simulation state of the SDM simulation model represents the PR_EVENT_TYPE enumeration state of the SDM. The `config_stream_endpoint_pr_test_pkg` SystemVerilog package defines these enumerations. These states represent the different allowed states for the SDM. The following are SDM enumeration definitions:

```
package config_stream_endpoint_pr_test_pkg;
    typedef enum logic [31:0] {
        NONE,
        IDLE,
        PR_REQUEST,
        PR_IN_PROGRESS,
        PR_COMPLETE_SUCCESS,
        PR_COMPLETE_ERROR,
        PR_INCOMPLETE_SYS_BUSY,
        PR_INCOMPLETE_BAD_DATA
    } PR_EVENT_TYPE;
```

2.8.6.2. Simulating Unknown Outputs and Persona Activation

To simulate the unknown outputs and persona activation that occurs during the partial reconfiguration process, follow these steps:

1. Ensure that the affected PR region has the simulation output multiplexes driven to X by asserting the `pr_activate` signal when the simulation state is `PR_IN_PROGRESS`.
2. In addition, you must assert the `pr_activate` signal in the PR simulation model to load all registers in the PR model with the PR activation value.
3. Once the simulation state reaches `PR_COMPLETE_SUCCESS`, activate the appropriate PR persona using the appropriate PR region simulation wrapper mux sel signals.
4. Decode the region, as well as the specific select signal from the `sim_only_pr_id` signal of the SDM. This PR ID corresponds to the encoded ID in the simulation RBF.

The SDM simulation model checks encoded instructions in the following simulation RBF locations:

- 1st (`0x97566593`)
- 2nd (`0x4422XXXX`)
- 3rd (`0x5056XXXX`)

If any encoded instructions do not stream into the model in the specified location, the simulation model triggers a PR error, and the error state is reflected in the Partial Reconfiguration Controller Intel FPGA IP or in the Partial Reconfiguration External Configuration Controller Intel FPGA IP.

If the exact number of dummy data that `NNNN` specifies does not stream into the simulation model, the model outputs an info message (`PR warning: Exceed expected length of data!`) to indicate the mismatch number of data sent. No PR error triggers in this case.

2.8.6.3. Simulation RBF Required Word Sequence

The simulation RBF requires the following sequence of words:

RBF Location	RBF Instruction	OpCode	Description
1st	Start of RBF	`97566593`	Identifier word indicates the start of the RBF.
2nd	Expected length (*N*) of data in RBF	`4422NNNN`	Specifies the length of dummy data `NNNN`.
3th	PR region ID (*R*) to activate	`5056RRRR`	Specifies the PR region ID RRRR, which reflects in the `sim_only_pr_id` signal. The PR region ID must be unique, design wide, across all PR partitions.
4th till 4th + `NNNN`	Dummy data	`XXXXXXXX`	Streams `NNNN` number of dummy data into the simulation model. `NNNN` is specified in the 2nd location of the RBF.
4th + `NNNN` + 1	End of RBF	`00001011`	Identifier word that indicates the end of the RBF.

Note: The PR region ID word outputs on the `sim_only_pr_id` word, starting after the first data word. Using a different value for the header or data count results in PR simulation errors.

2.9. Quartus Prime Pro Edition User Guide: Partial Reconfiguration Archive

For the latest and previous versions of this user guide, refer to Quartus Prime Pro Edition User Guide: Partial Reconfiguration. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

2.10. Partial Reconfiguration Solutions IP User Guide Revision History

Document Version	Quartus Prime Version	Changes
2023.04.01	24.1	• Applied initial Altera rebranding throughout. • Updated throughout to reflect support for Agilex 5 devices.
2023.04.03	23.1	• Updated product family name to "Intel Agilex 7." • Added note to *PR Error Recovery* topic about device family support.
2022.01.11	21.4	• Revised *Partial Reconfiguration Controller Intel FPGA IP* section to reflect the new error recovery mechanism that flushes out any remaining corrupted PR bitstream that might be left in user logic pipeline after sending a corrupted bitstream. • Revised Partial Reconfiguration Controller Intel FPGA IP *Parameters* topic for new Enable Protocol Checker and Enable SDM FW Error Reporting parameters that report additional error handshake status and response for detailed error analysis in PR Controller FPGA IP for internal host and debug SDM JTAG command for external host. • Added new *PR Error Recovery* topic. • Added new *PR Error Recovery Timing Specifications* topic. • Added new *Secure Device Manager Firmware Error Reporting* topic. • Added new *SDM Firmware Error Reporting Timing Specifications* topic. • Added new *Partial Reconfiguration External Controller Intel FPGA IP Timing Specifications* topic. • Added new *Secure Device Manager Partial Reconfiguration Simulation Model* section. • Corrected typo in *Agent Interface* topic.
2021.10.04	21.3	• Updated Partial Reconfiguration External Configuration Controller Intel FPGA IP *Ports* topic status bit information. • Updated Partial Reconfiguration Controller Intel FPGA IP *Ports* topic status bit information. • Updated non-inclusive terms with "host" and "agent" for Avalon Memory Mapped interface references throughout.
2021.08.02	21.2	• Updated *Partial Reconfiguration Controller Intel FPGA IP* topic SEU note. • Revised *Register States and Programming Model* figure to correct sequence.
2020.09.28	20.3	• Replaced references to Avalon-MM and Avalon-ST with Avalon memory-mapped and Avalon streaming for legal compliance.
2020.08.07	20.2	• Corrected signal name typos in "Interface Ports" topic for Avalon-MM Partial Reconfiguration Freeze Bridge Intel FPGA IP.
2019.12.16	19.4.0	• Added "Error Detection CRC Requirements" topic.

continued...

Document Version	Quartus Prime Version	Changes
2019.09.30	19.3.0	• Updated the name of "Intel Stratix 10 Partial Reconfiguration Controller FPGA IP" to "Partial Reconfiguration Controller Intel FPGA IP" to encompass support for Agilex 7 devices. • Added note about connection of dummy_clk in "PR Control Block and CRC Block VHDL Module." • Added note to "PR Bitstream Compression and Encryption" topic about support for enhanced decompression.
2019.06.07	19.1.0	• Added note and reference to *Stratix 10 Configuration User Guide*.
2019.04.22	19.1.0	• Indicated support for POF generation support for Intel Cyclone GX devices.
2019.01.04	18.1.0	• Clarified statement about configuration width in "Control Block Signals" topic.
2018.12.07	18.1.0	• Corrected typographical error in "Partial Reconfiguration IP Cores" table. • Corrected typographical error in "Avalon-MM Slave to PR Region Master Interface Ports" table.
2018.09.24	18.1.0	• Updated specification for Partial Reconfiguration Controller Stratix 10 FPGA IP from 250 MHz to 200 MHz. • Stated PR compilation flow support for Cyclone 10 GX devices. • Updated Partial Reconfiguration Controller Arria 10 FPGA IP name to Partial Reconfiguration Controller Arria 10/Cyclone 10 FPGA IP.
2018.06.27	18.0.0	Updated `freeze_status` signal description in *Registers: Partial Reconfiguration Region Controller*.
2018.06.18	18.0.0	• Corrected syntax error in *Generating the PR Persona Simulation Model*.
2018.05.07	18.0.0	• Added description of new Partial Reconfiguration External Configuration Controller Stratix 10 FPGA IP. • Updated names of Partial Reconfiguration Controller Arria 10 FPGA IP and Partial Reconfiguration Controller Stratix 10 FPGA IP. • Enhanced explanation of **Auto-instantiate CRC block** Partial Reconfiguration Controller Arria 10 parameter. • Added as chapter in new *Partial Reconfiguration User Guide*. • Added note about recovery after PR error when using SEU detection in Stratix 10 designs.
2017.11.06	17.1.0	• Added support for Stratix 10 Partial Reconfiguration Controller IP core. • Updated for latest Intel product naming conventions.
2017.05.08	17.0.0	Initial public release.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Third-party Simulation

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

1. FPGA Simulation Basics

This chapter is a high-level explanation of FPGA simulation basic concepts and workflows for all simulators that the Quartus® Prime software supports. An understanding of these basic concepts provides a foundation for performing simulation using your supported simulator of choice.

While the details of using a particular simulator vary, the basic foundational concepts and tasks of FPGA design simulation are common to all supported simulators.

Related Information

Supported Simulators on page 19

1.1. FPGA Simulation Essential Elements

The following describes the essential elements required for FPGA design simulation.

Design

An Quartus Prime design typically consists of a top-level design module containing a hierarchy of module instances, defined in one or more HDL files. The design that you intend to simulate is known as the Design Under Test (DUT).

Testbench

To simulate the DUT (that is, a design), you must also provide a separate HDL module (referred to as the testbench module) that instantiates the DUT and additional logic to stimulate the DUT and to capture the output from the DUT. The testbench module can include a hierarchy of module instances related to the testbench, but that are not part of the design. You define the testbench modules in one or more HDL files.

Top-Level Testbench

A top level testbench module is the testbench module that instantiates all other design and testbench related modules. This is the module you simulate.

HDL Design and Testbench Files

Simulating a design requires HDL design files, and one or more HDL testbench files. Quartus Prime designs typically consist of several modules that you define in multiple HDL files. These files can include HDL files generated by Quartus Prime tool, such as Quartus Prime Platform Designer.

Some of the modules instantiated in the design may be common to many designs. Examples of some common modules are low-level primitives, like AND and OR gates, and more complex blocks, such as multipliers and FIFOs.

The low level modules common to many designs are known as *simulation library modules*, and the files defining those modules are known as *simulation library files*. The Quartus Prime software installation provides various simulation library files, as The Quartus Prime Simulation Library describes.

The combination of design and testbench files includes all the modules that are instantiated in the top-level testbench module hierarchy, including all of the modules for the design, because the design is instantiated within the testbench hierarchy.

Executable Simulation Model

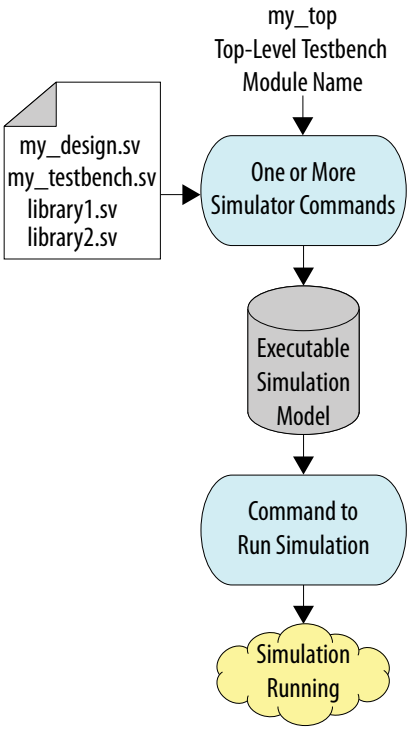
In order to simulate a design you must first generate an executable simulation model of the top-level testbench by running a set of simulator specific commands. You must then run the executable model to perform simulation. Running the executable model may require simulator specific commands. The executable model is typically a set of binary files specific to a simulator.

Simulator Commands

You must run one or more simulator commands to generate the executable simulation model and then to run the executable simulation model. The commands require the following inputs to generate an executable model of the top-level testbench module that you can simulate, as the Inputs and Commands to Generate and Run the Executable Model figure shows:

- The name of the top-level testbench module.
- The HDL design files, including files generated by tools such as Platform Designer, simulation library files, and testbench files.

Figure 1. Inputs and Commands to Generate and Run the Executable Simulation Model



Since you must run several commands to create and run the executable model to perform simulation, you can place the calls to the commands into one or more simulation scripts for convenience. These scripts can be Linux shell scripts, Tcl scripts, Perl, or Python scripts.

1.2. Overview of Simulation Tool Flow

The various simulator commands that you use to generate and run the executable model are all part of the simulation tool flow. A simulation tool flow consists of executing the following three stages of the simulation, in that order:

1. Compilation
2. Elaboration
3. Simulation

You run simulator specific commands at each stage in the flow.

1.2.1. Compilation Stage

In the first stage of simulation you run compilation commands.

Inputs to a Compilation Command

The compilation command takes as input one or more design files, testbench files, and simulation library files.

What Does a Compilation Command Do?

A compilation command does the following:

1. Reads the files that you specify as arguments to the command.
2. Analyzes the content of the files, which includes checking for syntax errors and other issues.
3. Stores the analyzed content (such as the module definitions) in a directory in a simulator specific proprietary format. The directory is known as a library directory. You can also specify the directory as an input argument.

   This step of storing the analyzed content of HDL files in a library directory is known as *compiling the files into a library directory*, or simply *compiling a file*.

Compiling a file is similar to running Quartus Prime Analysis & Synthesis, in that the analyzed file content is stored in design database directory. The compilation is also loosely analogous to compiling a C/C++ file into an object file, where the object file is stored in a separate directory.

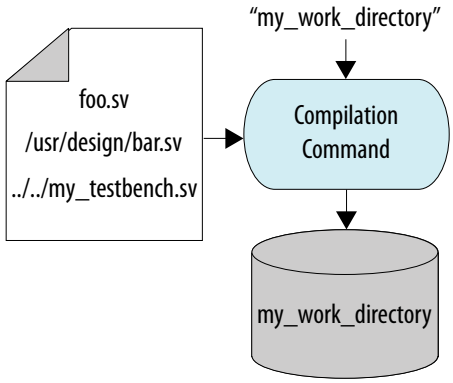
The library directory contains the definitions of all modules that are defined in the files that you compiled into the library. You use these module definitions in the elaboration stage. You may need to run multiple compilation commands to compile all the files into library directories.

Compilation Command Example

Consider a design example that has two design files, `foo.sv` and `bar.sv`, and one testbench file, `my_testbench.sv`.

To compile these files, you first create a new directory, for example `my_work`. Next, you run the simulator specific compilation command that takes in the file names and directory name as inputs, as the File and Directory Name Input to Simulator Specific Compilation Command figure shows. Once the command runs successfully, one or more files appear in the `my_work` library directory. The directory contains the definitions of all modules the three HDL files define, in a proprietary format that only the simulator understands.

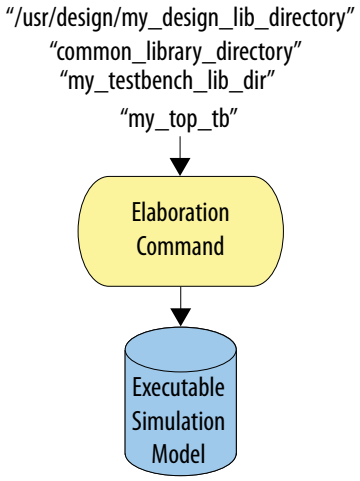
Figure 2. File and Directory Name Input to Simulator Specific Compilation Command



1.2.2. Elaboration Stage

The elaboration stage follows the compilation stage. In the elaboration stage you typically run just one elaboration command. This elaboration command can take several inputs. At the minimum, elaboration requires as input the top-level testbench module name, and the list of library directories that the compilation stage creates.

Figure 3. Elaboration Stage Inputs and Output



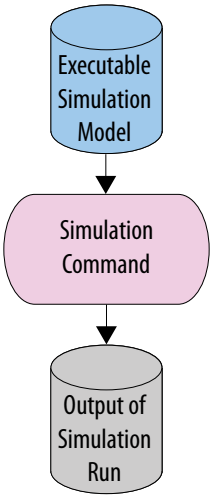
The output of the elaboration command is the executable simulation model for the top-level testbench. The executable simulation model comprises one or more simulator-specific files and directories. For more details about the elaboration stage, refer to Understanding Elaboration.

1.2.3. Simulation Stage

The commands that apply to the simulation stage actually run the simulation. The input to simulation stage commands is the executable simulation model that you generate during the elaboration stage, along with other inputs, such as how long to simulate, and which signals to capture for waveform viewing and dumping.

As the Files and Directories for Executable Model of my_top_tb figure shows, the output of the simulation stage is simply the output from a simulation run, which can include messages issued by HDL modules, files written out by the simulator such as waveform dumps, and any GUI display of simulation in progress.

Figure 4. Files and Directories for Executable Model of my_top_tb



1.3. Simulation Tool Flow

The simulation tool flow begins with the compilation stage that compiles files into logical libraries using simulator specific compilation commands.

The next stage is elaboration, where you run the elaboration command to generate an executable simulation model. In the final stage, you run the executable simulation model to run the simulation.

The following topics describe these simulation tool flow concepts in more detail:

1. Specifying Logical Libraries
2. Compiling Files into Library Directories
3. Understanding Elaboration
4. Commands to Configure and Run Simulation

1.3.1. Specifying Logical Libraries

Some simulator commands for compilation and elaboration require you to specify a *logical library* name as input.

A *Logical library* is simply a name (typically short and readable) that represents a physical library directory. For example, logical library name `foo` can represent physical directory `/users/jsmith/design1/bar`. The simulator commands translate the logical library name to a physical directory name by reading in a library mapping file. The simulator commands require only the physical directory names.

The mapping of logical library names to physical directory names is known as *library mapping*, which you must define. You typically store the library mapping in a separate text file in a proprietary text format, with each line containing a single logical library name and the corresponding library directory path. You can either update the file manually, or by using a simulator specific command (if available). This library mapping file often has a fixed simulator specific name and a fixed location. Therefore, you do not generally specify the library mapping file as an argument to simulator commands, even though the file is read by the commands.

A logical library name is an optional argument to many simulator commands. If you do not specify a logical library name for such commands, the default value is `work`. Therefore, you must map the logical library name `work` to a physical directory name. Some simulators add a default library mapping for the `work` library if you do not specify a mapping in a library mapping file. It is legal to map multiple logical library names to a single library directory.

Example library mapping file with logical library names `foo_lib` and `common_sim_lib`:

```
foo_lib  :   /users/john/designs/foo_dir
common_sim_lib  : /usr/sim/common/libraries
```

Note: syntax of library mapping file varies with simulator

1.3.1.1. Why Do We Need Logical Library Names?

Using logical library names instead of physical directory names in command invocations and in HDL files (especially VHDL files) simplifies some aspects of simulation. Use of logical library names makes it easier to port simulation scripts when moving the scripts across machines and disks because you only need to update the library mapping to reflect any new library directory paths in the new environment.

For example, Intel recommends compiling Quartus Prime simulation library files into fixed logical library names. You can then map the logical library names to appropriate library directory paths.

1.3.2. Compiling Files Into Library Directories

Many simulators include commands to compile one or more files, specified in some order, into a single library directory. You specify the library directory by specifying its logical library name. Some simulators have one command for compiling Verilog HDL or SystemVerilog files, and a different command for compiling VHDL files.

The following section describes the various commands for compiling files into library directories.

1.3.2.1. Inputs to Compilation Commands

Compilation commands accept the following inputs:

- An ordered list of one or more HDL file names, usually file names separated by spaces.

  Note: The file order is important in some cases, as Order of Files for Compilation Commands explains.
- (Optional) Command line options to configure the compilation behavior, as Compilation Command-Line Options describes.
- (Optional) The name of the logical library or a library directory name. When not specified, the logical library default value is the `work` library, as Specifying Logical Libraries describes.

Note:

1. You can compile two or more files using a single compilation command if you can compile them into the same library, and they require the same compilation options. The compilation command can take a list of HDL files as input.
2. You can compile files defining modules that are not part of the design or testbench. The elaboration stage ignores such modules. In fact, in practice, you typically compile many more modules than are required to simulate the top-level testbench module.

The compilation command generates outputs, as Compilation Stage describes.

1.3.2.2. Order of Files for Compilation Commands

The order of files that you specify to compilation commands is irrelevant for Verilog and SystemVerilog files in many instances. The main exception is when there are files defining SystemVerilog packages, or other files that import or otherwise refer to those SystemVerilog packages.

Important: You must compile the files defining the SystemVerilog packages before compiling the files that import or refer to those packages. Otherwise, the compilation command errors out when compiling files that import or refer to those SystemVerilog packages.

For example, suppose file `multp_pkg.sv` defines the SystemVerilog package `multp`, and the file `my_design.sv` imports package `multp`:

- If you compile both `multp_pkg.sv` and `my_design.sv` with a single compilation command, you must ensure that `multp_pkg.sv` occurs before `my_design.sv`.
- If you compile `multp_pkg.sv` and `my_design.sv` using separate compilation commands, you must ensure that you run the command that is compiling `multp_pkg.sv` first.

VHDL has stricter requirements for ordering the files. For example, when a VHDL file `foo.vhd` refers to a logical library name `lib1`, you must compile the files into `lib1` first, before compiling `foo.vhd` into another library.

1.3.2.3. Compilation Command Line Options

Some of the optional command-line arguments for the compilation command (not including HDL file names and library names) include:

- The type of file for compilation (Verilog HDL, SystemVerilog, or VHDL).
- The values of the Verilog macros to pass in.
- The directories containing Verilog "include" files. These are files included in a Verilog HDL file using the `` `include `` construct.
- Simulator-specific optimization switches.

1.3.2.4. Module Definitions in Library Directories

A library directory can contain one or more module definitions, as well as other elements, such as SystemVerilog package definitions.

A library directory can store only one module definition per module. For example, if the `adder.sv` and `adder_fast.sv` files define the same module `adder`, but have different implementations (perhaps `adder_fast.sv` implements a fast adder), then compiling both files into the same library directory with a single compilation command results in a compilation error. However, you can compile the `adder.sv` and `adder_fast.sv` files into different library directories.

You can also replace an existing module definition in a library with another module definition with the same module name. For example, if a library directory already includes a module definition for `adder` (from compiling file `adder.sv`), and you compile the `adder_fast.sv` file into that library directory, the existing module definition in the library directory is replaced with the module definition from `adder_fast.sv`.

1.3.3. The Quartus Prime Simulation Library

The Quartus Prime software includes the Quartus Prime simulation library. This library is comprised of Verilog HDL and VHDL files in the following directory:

```
<quartus_installation>/quartus/eda/sim_lib
```

This library includes simulation models for all low-level blocks that you instantiate in your design. The library includes the following different types of low level blocks:

Table 1. Low Level Blocks in Simulation Library

Low-Level Blocks	Description
Gate-Level Primitives	Gate-level primitives include simple, non-parameterized modules, such as AND gates and flip-flops. `altera_primitives.v` and `altera_primitives.vhd` define the gate-level primitives. These primitives are only used in RTL designs. Post-synthesis and post-fit netlists do not include these primitives. Rather, these netlists include ATOMs.
Basic IP Function Blocks	Previously known as "megafuctions," these are basic parameterized blocks for functions such as FIFOs and multipliers. Only RTL designs use these blocks. Post-synthesis and post-fit netlists do not include these blocks.
ATOMs	Also known as WYSIWYGs, ATOMs are the lowest level primitives in an Quartus Prime design. There are different ATOM primitives, all of them parameterized modules with varying complexity. They represent the hardware blocks on the FPGA. For example there are ATOM modules that represent the I/O pins and buffers, FPGA lookup tables, DSP blocks, RAM blocks, and periphery

continued...

Low-Level Blocks	Description
	blocks, such as high speed transceivers and hardened Ethernet and PCIe blocks. You are not expected to instantiate ATOMs directly in your RTL. Rather, the ATOMs are instantiated in the RTL files that the Quartus Prime software generates. Since the Quartus Prime synthesis maps the design to ATOMs, the post-synthesis and post-fit netlists are netlists of ATOMs, known as ATOM netlists. The Fitter places and routes the ATOM netlist.
HDL Library Files	You compile the HDL library files into fixed logical locations, as Compiling Files into Library Directories describes. You must not compile the libraries for Questa* Intel® FPGA Edition. Instead use the included precompiled libraries.

1.3.3.1. The Quartus Prime Simulation Library Compiler

The Quartus Prime Simulation Library Compiler is an Quartus Prime software GUI and command-line tool that generates simulation scripts. You can use these scripts to automatically compile the Quartus Prime software simulation libraries for a given simulator, device family, and hardware description language (Verilog HDL or VHDL).

Note: For Questa Intel FPGA Edition, do not use the Simulation Library Compiler to compile the libraries in Questa Intel FPGA Edition. Instead, you must use the Questa Intel FPGA Edition precompiled libraries included with this simulator.

Related Information

Questa Intel FPGA Edition Simulation User Guide

1.3.3.2. Running the Simulation Library Compiler in a Terminal

You can run the Quartus Prime Simulation Library Compiler in a terminal without launching the Quartus Prime software GUI.

The following example command generates the Questasim `compile.do` simulation script that compiles all Verilog HDL simulation files for the specified Agilex™ 7 device family.

```
quartus_sh –simlib_comp -family agilex7 -tool questasim \
     -language verilog -gen_only -cmd_file compile.do
```

To view all available command-line options, you can run the following command:

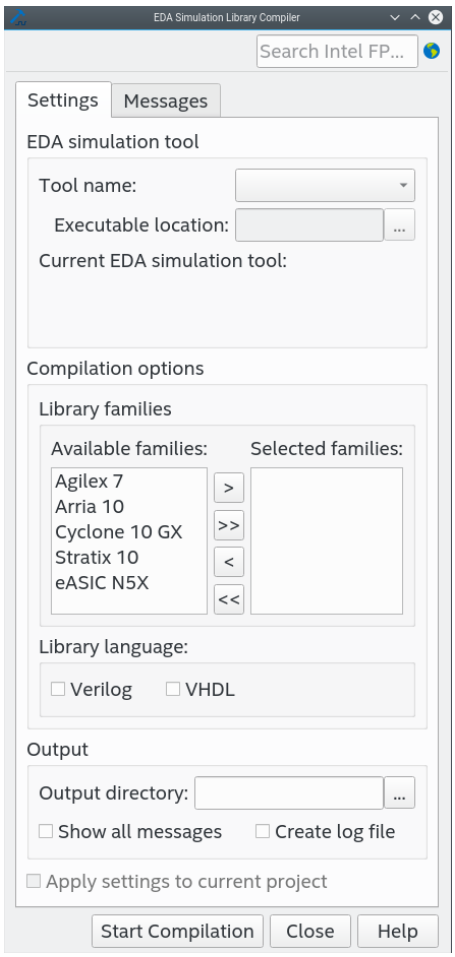
```
quartus_sh --help=simlib_comp
```

1.3.3.3. Running the Simulation Library Compiler in the GUI

To automatically compile all required simulation model libraries for your design in your supported simulator using the Simulation Library Compiler GUI, follow these steps:

1. In the Quartus Prime software, click **Tools ➤ Launch Simulation Library Compiler**.
2. Specify options for your simulation tool, language, target device family, and output location, and then click **OK**. Simulation model compilation may require up to an hour, depending on your system. Although the compilation messages may appear paused or incomplete, compilation is still running correctly.
3. Use the compiled simulation model libraries to simulate your design. For information about running simulation, refer to your supported EDA simulator's documentation.

Figure 5. Simulation Library Compiler GUI



1.3.3.4. Finding Logical Library Names in Simulation Library Compiler Output

After you generate the simulation script using the Simulation Library Compiler, you may need to inspect the script to identify the logical library names for use with your elaboration command (`vsim`).

To identify the logical library names for Quartus Prime simulation libraries in the generated script, search for all of the lines that begin with `vmap`, such as the following line:

```
vmap altera_ver "./verilog_libs/altera_ver"
```

The first argument to `vmap` is the logical library name (`altera_ver`). The second argument is the physical directory where the library content is stored. This second argument is irrelevant for Questa Intel FPGA Edition because you do not run the command.

1.3.4. Understanding Elaboration

Simulator elaboration is analogous to the linking step in C/C++ programming that produces an executable binary file.

You can run elaboration with a single command that accepts the following inputs and generates an executable model for the top-level testbench module name:

- An ordered list of logical library names. You can specify the ordered list of logical library names either explicitly on the elaboration command line, or by ordering them in the library mapping file. If reading from the library mapping file, the simulator uses the order of logical libraries in the library mapping file.
- (Optional) Elaboration options.
- Top-level testbench module name.
- (Optional) The name of the logical library containing the top-level testbench module definition. If omitted, the top-level testbench module defaults to the `work` library.

The elaboration command does not read any HDL files. The elaboration command only reads the library directories containing the module definitions.

An important part of elaboration is to find the module definitions for all the module instances in the top-level testbench module hierarchy. This identification is described as binding the module instances to their module definitions, or linking the module instances to their module definitions. Understanding the binding process during elaboration is important when debugging common elaboration errors, as Elaboration Binding Phase describes.

1.3.4.1. Elaboration Binding Phase

Elaboration works in a top-down manner to bind module instances in the following order:

1. Elaboration finds the top-level testbench module definition, given the module name and the library that contains the module definition as input. Typically, you compile the top-level testbench module into the `work` library. For example, specifying the top-level testbench module as `foo` with no library name, is equivalent to specifying the top-level testbench module as `work.foo`.
2. Elaboration reads the module definition, and identifies all the module instances in the top-level testbench module.
3. Elaboration attempts to find the module definitions for all instances in the top-level testbench, one instance at a time.

   For example, for an instance `inst1` of module `foo` in the top-level testbench module `tb`, elaboration attempts to find the definition of module `foo` by searching for `foo` in the first library in the ordered list of library directories. If elaboration cannot find the module definition in the first library directory, it searches in the second library directory, and so on.

   Once elaboration finds the definition of `foo` in a library directory, it stops searching for the definition. Therefore, if `foo` is defined in multiple library directories, elaboration uses only the first instance, and ignores any other instances. In this way, elaboration binds `inst1` to `foo`.

4. Elaboration attempts to find all of the module instances within `foo`, and then to find the module definitions for those instances using the same process that elaboration followed for binding `foo`.
5. Elaboration recursively attempts to bind all the module instances within the `foo` module's hierarchy before processing other instances in the top-level testbench `tb`.
6. The elaboration stage ends in one of the following ways:
   - All instances in the top-level testbench hierarchy are bound to modules, and elaboration succeeds.
   - An error is generated because elaboration cannot bind one or more instances in the top-level testbench module hierarchy to modules.

1.3.4.2. Elaboration Checks

The elaboration command performs several checks. For example, elaboration verifies that the module definitions are consistent with their instantiations. This check confirms that a module's ports and parameter definitions match the corresponding module instances.

1.3.4.3. Elaboration Options

There are many simulator specific elaboration options that you can specify. One common elaboration option preserves specific signal names so that their waveforms (a record of how the signals change with time) can be recorded during simulation. The rationale behind this option is explained below.

The elaboration stage generally includes an optimization step. The optimization step attempts to build an optimized executable simulation model that can run faster or consume less memory during simulation.

There are many signals (defined as wire, reg, or logic variables) in a typical design and testbench hierarchy. At the end of a simulation, any signal can produce a simulation waveform.

The optimization step may be unable to fully optimize the executable simulation model if most of the signals in the testbench hierarchy are preserved. Therefore, it is best to limit the signals that you preserve to those that require waveforms during simulation. You can specify the signals to preserve at varying level of granularity. For example, you can specify specific signal names, or all signals within a module instance.

1.3.5. Commands To Configure and Run Simulation

Once you generate the executable simulation model during elaboration, you can run the executable simulation model to simulate the top-level testbench module.

There are several different methods to configure and run simulation. The following are some of the typical simulator commands and options that you can use for simulation:

- You can specify which signals that you want the simulator to record during simulation. You must ensure that those signals are preserved during the elaboration stage, as Elaboration Options explains.

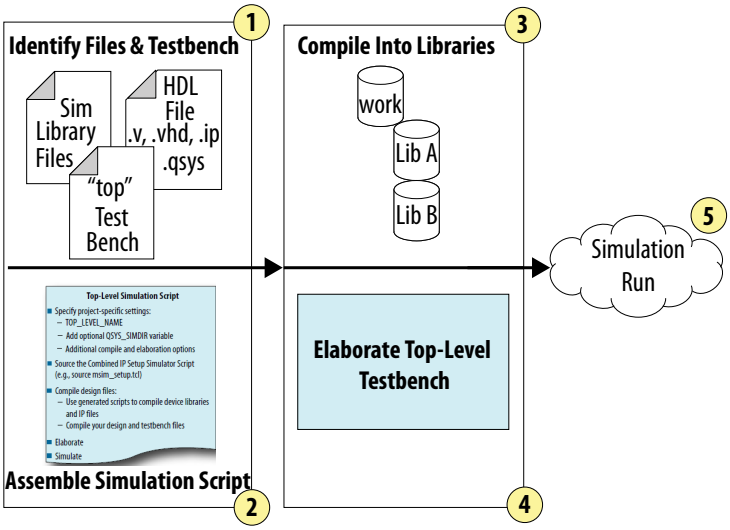
  The simulator writes the waveforms of these signals to a simulator proprietary database during simulation. You can view the waveforms in a GUI after simulation.

  Note: You cannot record or display signals in encrypted HDL files with the simulator.
- You can specify the amount of simulation time to simulate the top-level testbench module. For example, you can specify a simulation time of 1 milliseconds.
- You can specify an option to wait for simulation licenses. This option is applicable when using floating simulation licenses. Some simulators exit immediately if there are no available floating licenses for simulation.

1.3.6. FPGA Simulation Generic Workflow

The following describes the high level workflow for simulation of any Quartus Prime design using any supported simulator:

Figure 6. Generic FPGA Simulation Workflow



1. Identify all of the HDL simulation files, including design files, simulation library files, and HDL testbench files.
2. Identify the top-level test bench module for simulation.
3. For each HDL simulation file, determine the logical library for compilation, and any compilation options for compiling the file.(1)
4. Determine any simulator-specific elaboration options required for elaborating the top- level testbench module, as Understanding Elaboration describes.
5. Use the information gathered in previous steps to assemble a simulation script to compile, elaborate, and simulate the design. This script must include commands to perform the following:

- Compile the simulation files into libraries, as Compiling Files into Library Directories describes.
- Elaborate the Top-Level testbench, as Understanding Elaboration describes.
- Run the executable simulation model to simulate the testbench and the design, using the appropriate commands for your simulator to configure and run simulation.

The Quartus Prime software can generate simulator-specific simulation scripts to automate some of the simulation processing in your preferred simulation environment.

The Quartus Prime software can generate a simulator specific simulation script for an IP core, or a Platform Designer system, for use in RTL simulation. The script includes commands to compile all the IP RTL files, as well as an elaboration command with any simulator specific options.

The Quartus Prime software can generate a simulation library compilation script for a given simulator, device family, and language. This script includes commands to compile the simulation library files for the specified simulator, device family, and language. You can use this script for RTL simulation and gate-level simulation.

1.4. Supported Simulation Flows

The Quartus Prime software supports scripted and specialized simulation flows.

Table 2. Simulation Flows

Simulation Flow	Description
Scripted Simulation Flows	Scripted simulation supports custom control of all aspects of simulation, such as custom compilation commands, or multipass simulation flows. Use a version-independent top-level simulation script that sources Quartus Prime-generated IP simulation setup scripts. The Quartus Prime software can generate a combined simulator setup script for all IP cores, for each supported simulator.
Run Simulation	You can use the Run Simulation feature in the Quartus Prime Pro Edition software to integrate your supported third-party EDA simulator and automate generation of simulator-specific files and setup scripts, compilation of simulation libraries, and launch of your simulation.
Qrun Flow	The Qrun flow optionally creates simulation files, including the functional simulation model, and any testbench (or example design) for the QuestaSim* and Questa Intel FPGA Edition simulators only. The Qrun flow can automatically combine the compile, optimize, and simulate functions into a single step.
Specialized Simulation Flows	Specialized simulation flows support various design scenarios: • For simulation of example designs, refer to the example design pr IP documentation. • For simulation of Platform Designer designs, refer to *Simulating Platform Designer Systems* in *Quartus Prime Standard Edition User Guide: Platform Designer*. • For simulation of the Nios® V processor, refer to the *Nios V Embedded Processor Design Handbook*.

1.5. Supported Hardware Description Languages

The Quartus Prime software provides the following hardware description language (HDL) support for EDA simulators.

(1) In general, you can compile most HDL simulation files into the default work library.

Table 3. HDL Support

Language	Support Description
VHDL	• For VHDL simulation, you compile design files, testbench files, and Platform designer generated RTL files using simulator commands. • For all supported simulators other than Questa Intel FPGA Edition, you must also compile simulation models from the Quartus Prime simulation libraries. • Many of the Quartus Prime simulation models and IP RTL files are implemented in Verilog or SystemVerilog only. Therefore, you may require a simulator that is capable of VHDL and Verilog HDL mixed language simulation.
Verilog / SystemVerilog	• For Verilog or SystemVerilog simulation, you compile design files, testbench files, and Platform Designer generated RTL files using simulator commands. • For all supported simulators other than Questa Intel FPGA Edition, you must also compile simulation models from the Quartus Prime simulation libraries. • There are some IP RTL files that are implemented in VHDL only. Therefore, you may require a simulator that is capable of VHDL and Verilog HDL mixed language simulation.
Mixed HDL	• If your design is a mix of VHDL, Verilog HDL, and SystemVerilog files, you must use a mixed language simulator. • The Questa Intel FPGA Edition software supports native, mixed-language (VHDL/Verilog HDL/SystemVerilog) simulation. If you have a VHDL-only simulator and need to simulate Verilog HDL modules and IP cores, you can either acquire a mixed-language simulator license from the simulator vendor, or use the Questa Intel FPGA Edition simulator.
Schematic	• You cannot simulate a schematic in any of the simulators that the Quartus Prime software supports. • To perform RTL simulation of the schematic, you must convert the schematic to HDL format and run RTL simulation on the HDL. The Quartus Prime Pro Edition software cannot perform schematic conversion. • To perform post-synthesis or post-fit simulation, you must first compile the schematic based design in the Quartus Prime software, generate a gate-level Verilog HDL or VHDL simulation netlist, and perform simulation on the gate-level netlist.

1.6. Supported Simulation Types

You can run different types of simulation, depending on the stage of the Quartus Prime design flow:

Table 4. Supported Simulation Types

Simulation Type	Description	Occurs
RTL	Simulation of an RTL design consisting of one or more RTL files that you provide as input to the Quartus Prime software. These RTL files typically also include the files that the Quartus Prime Platform Designer generates for Intel FPGA IP and systems. You can only simulate HDL RTL files.(2). The RTL files can instantiate low level blocks, such as primitives, basic IP functions, and ATOMs, as Intel Quartus Prime Simulation Library describes.	Can perform before Quartus Prime Synthesis
Post-Synthesis Simulation (Gate-Level)	The Quartus Prime software can generate a Verilog HDL or VHDL gate-level netlist after the synthesis stage completes, but before the Fitter stage runs. The resulting netlist is the post-synthesis netlist. The	Must perform after Quartus Prime synthesis

continued...

(2) You must first convert the non-HDL files to HDL files prior to simulation

Simulation Type	Description	Occurs
	Quartus Prime EDA Netlist Writer tool generates the post-synthesis netlist. The post-synthesis netlist is a netlist of low level blocks called ATOMs. The post-synthesis netlist is a purely functional netlist.	
Post-Fit Simulation (Gate-Level)	The Quartus Prime EDA Netlist Writer can generate a Verilog HDL or VHDL gate-level netlist after the Fitter stage completes. The resulting netlist is the post-fit netlist. The post-fit netlist is a netlist of ATOMs that the Fitter placed and routed on the FPGA device. The post-fit netlist is a purely functional netlist. *Note:* The post-fit netlist includes chip locations of ATOM instances in commented lines. The post-synthesis netlist does not include this data.	Must perform after Quartus Prime Fitter

Note: the Quartus Prime software supports post-fit functional simulation, but does not support post-fit timing simulation.

1.7. Supported Simulators

The Quartus Prime software supports the following EDA simulator versions for RTL and gate-level simulation.

Table 5. Quartus Prime Pro Edition Supported Simulators

Vendor	Simulator	Version	Platform	Supports Siemens EDA Verification IP
Aldec	Active-HDL*	14.0	Windows* 64-bit	No
Aldec	Riviera-PRO*	2023.10	Windows, Linux, 64-bit	No
Cadence	Xcelium* Parallel Simulator	23.09.004	Linux 64-bit	Yes
Altera	Questa Intel FPGA Edition	2023.4	Windows, Linux, 64-bit	Yes
Siemens EDA	QuestaSim Simulator(3)	2023.4	Windows, Linux, 64-bit	Yes
Synopsys*	VCS*, VCS MX	U-2023.03-SP2-1	Linux 64-bit	Yes

Table 6. Quartus Prime Standard Edition Supported Simulators

Vendor	Simulator	Version	Platform
Aldec	Active-HDL	14.0	Windows
Aldec	Riviera-PRO	2023.04	Windows, Linux
Cadence	Xcelium	23.03.003	Linux
Altera	Questa Intel FPGA Edition	2023.3	Windows, Linux
Siemens EDA	QuestaSim	2023.2	Windows, Linux
Synopsys	VCS VCS MX	U/2023.03-1	Linux

(3) QuestaSim is the generic name for Questa Core and Questa Prime simulators from Siemens EDA.

Related Information

- Questa Intel FPGA Edition Simulation User Guide
- IBIS Models for Intel FPGA Devices

1.8. Post-Fit Simulation Support by FPGA Family

The current version of the Quartus Prime Pro Edition software provides the following post-fit functional simulation support per FPGA family. In the table, Core fabric is the main FPGA die that includes lookup tables, M20K memories, and DSP blocks. Basic I/Os do not include complex interfaces, such as transceivers or external memory interfaces.

Table 7. Post-Fit Simulation Support by FPGA Family

FPGA Family	Post-Fit Simulation Support Level
Agilex 7 F-Series FPGAs	Limited support for designs that contain only core fabric logic and basic I/Os.
Agilex 7 M-Series FPGAs	Limited support for designs that contain only core fabric logic and basic I/Os.
Agilex 7 I-Series FPGAs	Limited support for designs that contain only core fabric logic and basic I/Os.
Stratix® 10 FPGAs	Limited support for designs that contain only core fabric logic and basic I/Os.
Cyclone® 10 GX FPGAs	Limited support for designs that contain only core fabric logic and basic I/Os.
Arria® 10 FPGAs	Limited support for designs that contain only core fabric logic and basic I/Os.

Note: For more information, refer to the specific FPGA device user guide.

1.9. Automating Simulation with the Run Simulation Feature

You can use the Run Simulation feature in the Quartus Prime Pro Edition software to integrate your supported third-party EDA simulator and automate the following steps of the simulation flow:

- Automatic generation of simulator-specific files and setup scripts.
- Automatic compilation of simulation libraries.
- Automatic launch of your simulator after running Quartus Prime Analysis & Elaboration.

You can use the simulation scripts that Run Simulation generates as a starting point to create more custom simulation scripts that may be required for large or complex FPGA designs.

Note: The current version of the Quartus Prime Pro Edition software supports the Run Simulation feature for only RTL simulation, not gate-level simulation.

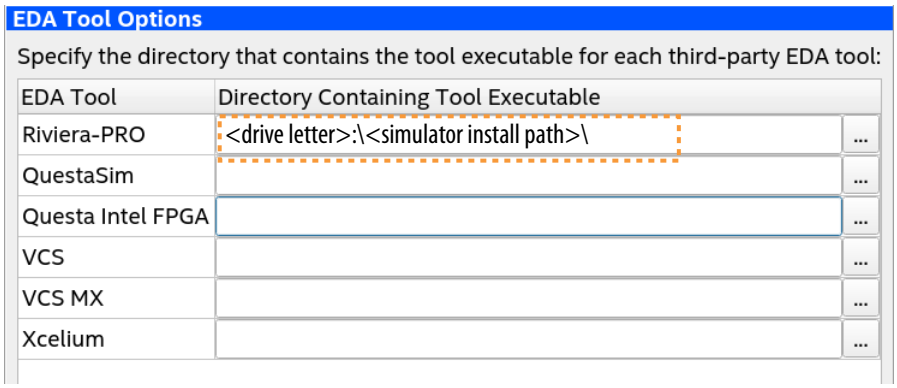
1.9.1. Setting Up the Run Simulation Feature

You must first setup the Run Simulation feature before using it to automate portions of the simulation flow.

To setup the Run Simulation feature by specifying the settings that identify your simulator, output path, and other options, follow these steps:

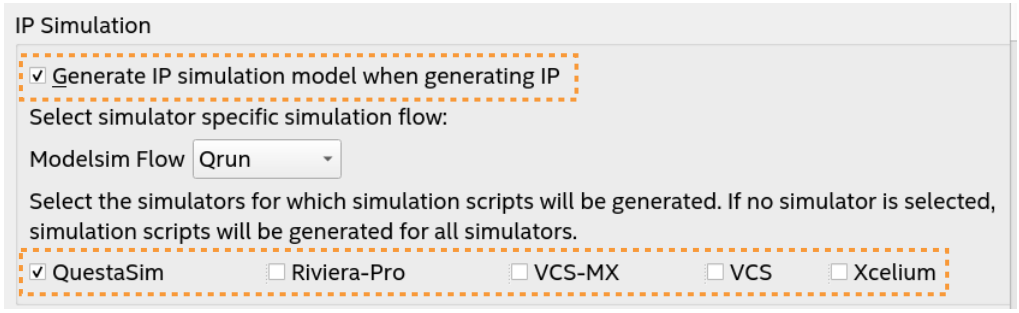
1. Open a project in the Quartus Prime software.
2. Click **Tools ➤ Options ➤ EDA Tool Options** and specify the location of your simulator executable file, as Execution Paths for Supported EDA Simulators describes in detail.

Figure 7. Specifying Simulator Install Path



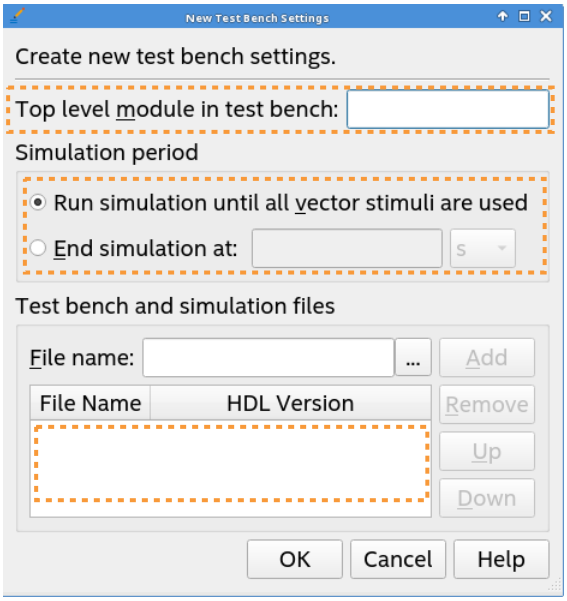
3. To enable automated generation of the IP simulation models whenever you generate HDL for IP in Platform Designer, click **Tools ➤ Options ➤ Board and IP Settings ➤ IP Simulation**. Make sure **Generate IP simulation model when generating IP** option is turned on, as Simulation Options describes in detail.

Figure 8. Specifying Automated IP Simulation Model Generation



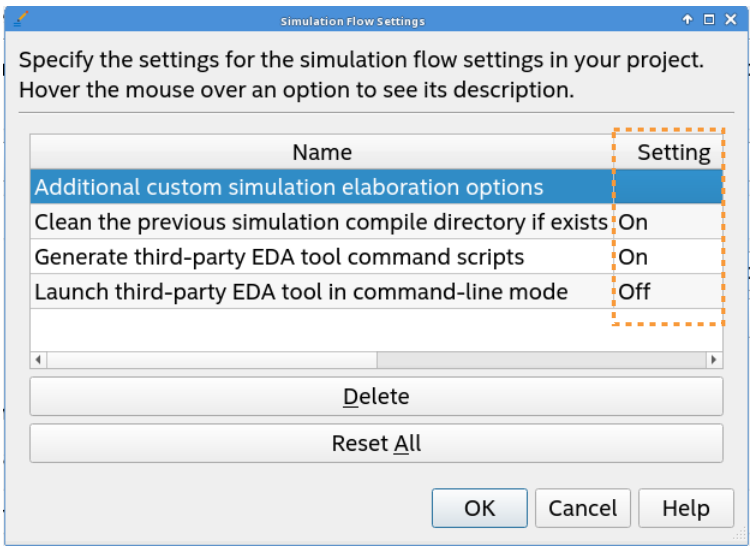
4. Click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ Simulation** and specify the following simulation settings:
   a. For **Testbench Specification**, click the **New** button and enter the testbench information, including the **Top level module in testbench**, **Simulation period**, and **Testbench and simulation files** options.

Figure 9. Defining Testbench Specification



b. Click the **Simulation Flow Settings** button to specify additional options for the automated simulation flow, as Simulation Flow Settings describes in detail.

Figure 10. Simulation Flow Settings



1.9.1.1. Installation Paths for Supported EDA Simulators (EDA Tool Options Page)

The following are execution paths for supported EDA simulators. Click **Tools ➤ Options ➤ EDA Tool Options** to specify the execution path to automatically launch your simulator.

Table 8. Installation Directory Paths for Supported EDA Simulators

Simulator	Path
Questa Intel FPGA Edition	Windows: *<drive letter>*:*<simulator install path>*\ Linux: *<simulator install path>*/
Siemens EDA QuestaSim	Windows: *<drive letter>*:*<simulator install path>*\ Linux: *<simulator install path>*/
Synopsys VCS/VCS MX(4)	Linux: *<simulator install path>*/bin
Aldec Active-HDL	Windows: *<drive letter>*:*<simulator install path>*\bin
Aldec Riviera-PRO	Windows: *<drive letter>*:*<simulator install path>*\bin Linux: *<simulator install path>*/bin
Cadence Xcelium(4)	Linux: *<simulator install path>*/tools/bin/64bit

1.9.1.2. Simulation Options (Board and IP Settings Page)

The following options impact automated IP simulation model generation. Click **Tools ➤ Options ➤ Board and IP Settings ➤ IP Simulation** to specify the whether to always generate IP simulation models when generating IP for one or more simulator..

Table 9. Simulation Options (Board and IP Settings Page) Options Dialog Box

Option	Allowed Values	Description
Tool name	<None>(5) Active-HDL(6) Riviera-PRO QuestaSim Questa Intel FPGA Edition VCS/VCS MX(4) Custom(6) Xcelium(4)	Specifies the supported simulator to automatically run.
Format for output netlist	Verilog HDL VHDL	Specifies Verilog or VHDL as the format for the output netlist. The setting does not apply to RTL simulation.
Output directory	Any valid path	Specifies the directory to store all output files for simulation. Default is `simulation/<simulator>`.
Map illegal HDL characters	Disabled (Default) Enabled	When enabled, directs the EDA Netlist Writer to map illegal characters for VHDL or Verilog HDL. The setting does not apply to RTL simulation. If you select **VHDL** for **Format for output netlist**, the EDA Netlist writer maps non-alphanumeric characters, including brackets ([]), parentheses, (( )), angle brackets (<>), and braces ({}) to (_a) in VHDL Output Files. This option generates VHDL 1987 compatible names.

continued...

(4) Only supported for Linux OS.

(5) Not supported by Run Simulation

(6) Only supported for Windows OS.

Option	Allowed Values	Description
		If you select **Verilog HDL** for **Format for output netlist**, the EDA Netlist writer maps the vertical bar (|), tilde (~), and colon (:) characters in hierarchical node names to the legal Verilog HDL characters z, x, and underscore (_) in Verilog Output Files. This option also maps other illegal non-alphanumeric characters, including brackets ([]), parentheses, (( )), angle brackets (<>), and braces ({}) to underscore (_).

1.9.1.3. Simulation Flow Settings (EDA Tool Settings Page)

The simulation flow settings allow you to specify additional options for the automated simulation flow. Click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ Simulation ➤ Simulation Flow Settings** to specify any of the following additional options.

Table 10. Simulation Flow Settings (EDA Tool Settings Page) Settings Dialog Box

Name	Setting	Description
Additional custom simulation elaboration options		Allows you to specify additional custom simulation elaboration options for one or more simulators. For example: `questa=-suppress 2732 -suppress 14408 -suppress 16154` `vcs=+define+IP7521SERDES_ UX_SIMSPEED`
Clean the previous simulation compile directory if exists	Off On (Default)	Allows you to clean (On) or retain (Off) the simulation directory created by the previous simulation run.
Generate third-party EDA tool command scripts	Off On (Default)	Allows you to generate only the command scripts for the third-party EDA tool without launching the simulator itself. Select Off to launch the simulator using the Run Simulation feature.
Launch third-party EDA-tool in command-line mode	Off (Default) On	Allows you to launch a third-party EDA tool in command-line mode (On) rather than opening the GUI (Off).

1.9.1.4. More EDA Netlist Writer Settings (EDA Tool Settings Page)

The **More EDA Netlist Writer Settings** dialog box allows you to specify settings that control how the Compiler generates and formats the gate-level netlist for gate-level simulation. These setting do not apply to RTL simulation.

Click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ Simulation ➤ More EDA Netlist Writer Settings** to specify any of the following additional options.

Table 11. More EDA Netlist Writer Settings (EDA Tool Settings Page)

Name	Setting	Description
Architecture name in VHDL output netlist	structure	Specify the name of the architecture in the generated VHDL simulation netlist.
Bring out device-wide set/reset signals as ports	Off (Default) On	Add the `devpor`, `devclrn`, and `devoe` signals in the design as input ports in the top-level design hierarchy in the Verilog or VHDL simulation netlist for the project.
Do not write top level VHDL entity	Off (Default) On	Do not write top-level entity in VHDL Output File (`.vho`).

continued...

Name	Setting	Description
Flatten busses into individual nodes	Off (Default) On	Flattens all busses when creating the VHDL Output File (`.vho`). Turn on this option if your third-party EDA environment does not support buses.
Force Gate Level Simulation Registers to initialize to X (don't care) and propagate X	Off (Default) On	Modifies output gate level simulation netlist to force all registers to initialize to X (don't care) and propagate X.
Generate Power Estimate Scripts	Off (Default) On	Write scripts for simulation tool to generate `.vcd` file for outputs for power estimation.
Truncate long hierarchy paths	Off (Default) On	Truncate hierarchical node names to 80 characters.

1.9.2. Run RTL Simulation using Run Simulation in Batch Mode (Command-Line)

You can run RTL simulation using the Run Simulation feature in batch mode by using any of the following methods:

- By entering Tcl commands in the Quartus Prime Tcl Console.
- By entering global settings directly in the Quartus Prime Settings File (`.qsf`) (for simulation environment setup only).
- By entering commands in the Quartus Prime command-line shell:

```
quartus_sh -t <script file> \
[<script args>]
```

Note: Cadence Xcelium only supports Run Simulation in batch mode. Aldec Active-HDL only supports Run Simulation in the GUI mode.

1.9.2.1. Specifying Required Simulation Settings for Run Simulation (Batch Mode)

There are both required and optional setting for use of Run Simulation in batch mode.

The following steps describe specifying the required simulation settings for use of Run Simulation in batch mode:

1. Set the compatible EDA Simulator executable path. For example:

```
set_user_option -name EDA_TOOL_PATH_ QUESTA_INTEL /<simulator install
 path>/questa_fe_tag/24.1/92/linux64/linux_x86_64

set_user_option -name EDA_TOOL_PATH_ QUESTASIM /<simulator install
 path>/eda/mentor/questasim/2023.4/linux64/linux_x86_64

set_user_option -name EDA_TOOL_PATH_ VCS /<simulator install
 path>/eda/synopsys/vcsmx/U-2023.03-1/linux64/suse/bin

set_user_option -name EDA_TOOL_PATH_ VCS_MX /<simulator install
 path>/eda/synopsys/vcsmx/U-2023.03-1/linux64/suse/bin

set_user_option -name EDA_TOOL_PATH_ACTIVEHDL <drive letter>:\<simulator
install
 path>\eda\aldec\activehdl\13.0\windows64\bin

set_user_option -name EDA_TOOL_PATH_RIVIERAPRO /<simulator install
path>/eda/aldec/riviera/2023.04.082/linux64/bin
```

```
set_user_option -name EDA_TOOL_PATH_ XCELIUM /<simulator install
  path>/eda/cadence/xcelium/23.03.003/linux64/suse/tools.lnx86/bin
```

2. Specify your supported EDA Simulator and HDL:

```
set_global_assignment -name EDA_SIMULATION_TOOL
"<simulator> (HDL)"
```

If not set, the following is the default setting:

```
set_global_assignment -name EDA_SIMULATION_TOOL
"Questa Intel FPGA (Verilog)"
```

Table 12. Settings for EDA Simulator and HDL

Simulator	Verilog	VHDL
Questa Intel FPGA Edition	`Questa Intel FPGA (Verilog)`	`Questa Intel FPGA (VHDL)`
QuestaSim	`QuestaSim (Verilog)`	`QuestaSim (VHDL)`
VCS	`VCS`	N/A
VCS MX	`VCS MX (Verilog)`	`VCS MX (VHDL)`
Active-HDL	`Active-HDL (Verilog)`	`Active-HDL (VHDL)`
Riviera-PRO	`Riviera-PRO (Verilog)`	`Riviera-PRO (VHDL)`
Xcelium	`Xcelium (Verilog)`	`Xcelium (VHDL)`

3. Set the testbench and simulation file (if any) names, as well as the section ID (arbitrary). This is a multi-value assignment. The following shows an example with the testbench `file 1` as the top-level testbench file:

```
set_global_assignment -name EDA_TEST_BENCH_FILE <testbench file 1>
-section_id testbenchSet
set_global_assignment -name EDA_TEST_BENCH_FILE <testbench file 2>
-section_id testbenchSet
set_global_assignment -name EDA_TEST_BENCH_FILE <testbench file 3>
-section_id testbenchSet
```

4. Set the top-level module name in the top testbench file.

```
set_global_assignment -name EDA_TEST_BENCH_TOP_MODULE <testbench top module
name>
 -section_id testbenchSet
```

1.9.2.2. Optional Simulation Settings for Run Simulation (Batch Mode)

There are both required and optional setting for use of Run Simulation in batch mode.

The following examples show how to specify optional simulation settings for use of Run Simulation in batch mode:

Optional Setting To Run Simulation for Specific Time Interval

```
set_global_assignment -name EDA_TEST_BENCH_RUN_SIM_FOR “1 ns” \
-section_id testbenchSet
```

Optional Setting for Custom Wave File

```
set_global_assignment -name EDA_SIMULATION_WAVE_FILE_QUESTA_INTEL wave.do \
-section_id testbenchSet
```

Optional Setting for Elaboration Options

For Questa Intel FPGA Edition or QuestaSim:

```
set_global_assignment -name EDA_EXTRA_ELAB_OPTION
"questa=-suppress 2732 -suppress 14408 -suppress 16154" -section_id
eda_simulation
```

For VCS (Linux-only):

```
set_global_assignment -name EDA_EXTRA_ELAB_OPTION
"vcs=+define+IP7521SERDES_UX_SIMSPEED\ " -section_id eda_simulation
```

For VCS MX (Linux-only):

```
set_global_assignment -name EDA_EXTRA_ELAB_OPTION
"vcsmx=+define+IP7521SERDES_UX_SIMSPEED\ " -section_id eda_simulation
```

For Active-HDL (Windows* only):

```
set_global_assignment -name EDA_EXTRA_ELAB_OPTION
"activehdl=<third-party elab options>" -section_id eda_simulation
```

For Riviera-PRO:

```
set_global_assignment -name EDA_EXTRA_ELAB_OPTION
"rivierapro=<third-party elab option(s)>" -section_id eda_simulation
```

For Xcelium (Linux-only):

```
set_global_assignment -name EDA_EXTRA_ELAB_OPTION
"xcelium=<third-party elab option(s)>" -section_id eda_simulation
```

Simulator GUI or Batch Mode Operation Optional Setting

The default mode of this option is GUI mode. For simulator batch mode:

```
set_global_assignment -name EDA_LAUNCH_CMD_LINE_TOOL ON
-section_id eda_simulation
```

1.9.2.3. Launching Simulation with the Run Simulation Feature

After providing all the required and optional simulation settings, you can launch simulation with the Run Simulation feature in the GUI (simulator window launched) or batch (simulator window not launched) mode.

- To run RTL simulation in GUI mode (simulator window launched):

```
set_global_assignment -name EDA_LAUNCH_CMD_LINE_TOOL OFF -section_id
eda_simulation
execute_flow -simulation
```

- To run RTL simulation in batch mode (simulator window not launched):

```
set_global_assignment -name EDA_LAUNCH_CMD_LINE_TOOL ON -section_id
eda_simulation
execute_flow -simulation
```

- To generate top-level simulation and `.do` scripts only and *not* run simulation:

```
set_global_assignment -name EDA_SIMULATION_GENERATE_SCRIPT_ONLY ON -
section_id eda_simulation
execute_flow -simulation
```

1.9.2.4. Running RTL Simulation using Run Simulation

To run RTL Simulation using the Run Simulation feature, follow these steps:

1. Set up the Run Simulation feature, as Setting Up the Run Simulation Feature describes.
2. If your design includes Intel FPGA IP, you generate the simulation model and setup scripts for IP components and Platform Designer systems when generating HDL for these IP. Click the **Generate HDL** button and specify **Simulation** options for model generation. For system and IP generation details, refer to *Quartus Prime Pro Edition User Guide: Platform Designer*.
3. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**. The Compilation Dashboard indicates when Analysis & Elaboration completes successfully.
4. In Quartus Prime Tcl Console, type the following command:

   ```
   execute_flow -simulation
   ```

   The Run Simulation feature compiles the simulation libraries and runs your simulator automatically, according to your settings when Setting Up the Run Simulation Feature. While the simulator is open, you are unable to make changes in the Quartus Prime software. This prevents changes to the simulation settings during simulation. The Quartus Prime software displays simulation output messages from the EDA simulator in the Messages window.
5. Analyze the simulation results in your simulator. Correct any functional errors in your design, testbench, or simulation script files. If necessary, re-simulate your design to verify correct behavior.

Related Information

Quartus Prime Pro Edition User Guide: Platform Designer

1.9.2.5. Output Directories and Files for Run Simulation

Run Simulation generates subdirectories and files inside the following output directories, according to your specifications in the Simulation Options.

```
./simulation/<simulator/arbitrary>/rtlsim/
   <project>_run_msim_rtl_<hdl>.do
      A .do file containing the simulator settings
   run_sim_command.sh (or .bat for Windows)
      A .sh/.bat file to rerun the simulation only (standalone, i.e., without
      rerunning Quartus Prime compilation and elaboration)
   <simulator/arbitrary>_transcript.log
      A simulator transcript file
```

```
./simulation/<simulator or arbitrary>/rtlsim/<project>_iputf_input
   ./aldec
      rivierapro_setup.tcl
   ./common
      modelsim_files.tcl
      riviera_files.tcl
      vcs_files.tcl
      vcsmx_files.tcl
      xcelium_files.tcl
   ./mentor
      msim_setup.tcl
   ./synopsys
      ./vcs
         vcs_setup.sh
      ./vcsmx
         synopsys_sim.setup
         vcsmx_setup.sh
   ./xcelium
      cds.lib
      hdl.var
      xcelium_setup.sh
      ./cds_libs
         Project library files
```

1.10. Intel FPGA Simulation Basics Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Updated simulator versions supported in *Supported Simulators* topic. • Revised *Supported Simulation Flows* topic to include the Run Simulation feature. • Added new *Automating Simulation with the Run Simulation Feature* section. • Applied phase I rebranding throughout.
2023.12.07	23.4	• Added new *Post-Fit Simulation by Intel FPGA Family* topic. • Updated simulator versions supported in *Supported Simulators* topic.
2022.12.21	22.4	• Replaced all content in chapter with newly developed content more suitable for basic understanding of FPGA design simulation. • Updated chapter to reflect end of support for ModelSim and relocation of Questa Intel FPGA Edition information to new *Questa Intel FPGA Edition Simulation User Guide*.
2022.04.13	22.1	• Updated simulator versions supported in *Simulator Support* topic. • Added note about longer compilation times to *Compiling Simulation Module Libraries* topic.

continued...

Document Version	Quartus Prime Version	Changes
2021.10.05	21.3	• Updated simulator versions supported in *Simulator Support* topic. • Revised name of Questa Intel FPGA Edition and QuestaSim for latest guidelines throughout. • Updated default output directory name to questa in *Using the EDA Netlist Writer* topic.
2021.10.04	21.3	• Revised *Generating IP Simulation Files* steps to include generation of all IP in the design at once.
2021.06.21	21.2	• Changed chapter title to *FPGA Simulation Basics* from *Simulating Intel FPGA Designs*. • Added support for Questa Intel FPGA Edition simulator throughout. • Removed support for ModelSim - Intel FPGA Edition simulator throughout. • Updated simulator versions supported in *Simulator Support* topic. • Added precompiled libraries footnote to *Supported Hardware Description Languages* and *Compiling Simulation Model Libraries* topics. • Revised *Running a Simulation (Custom Flow)* topic to add missing EDA Netlist Writer step and related links. • Replaced "Mentor Graphics" with "Siemens EDA" to reflect current company name. • Updated *Supported Hardware Description Languages* for note on schematic conversion. • Revised *Scripting IP Simulation* to correct typo in step 1. • Added links to *Incorporating Simulator Setup Scripts from the Generated Template* topic.
2021.03.29	21.1	• Added note about X propagation limit of the Quartus Prime-provided clock divider simulation model to the *Compiling Simulation Model Libraries* topic.
2020.10.10	20.1	• Revised *Generating IP Simulation Files* topic for new simulation file output options. • Updated supported simulator versions and removed support for Cadence Incisive Enterprise* in *Simulator Support* topic.

2. Siemens EDA QuestaSim Simulator Support

This chapter provides guidelines for simulation of Quartus Prime designs with the supported Siemens EDA QuestaSim simulators.

Note: Intel also provides the Questa Intel FPGA Edition simulator, a version of the Questa Advanced simulator targeted for Intel FPGA devices. The Questa Intel FPGA Edition simulator supports the Intel FPGA gate-level simulation libraries, and includes behavioral simulation, HDL test benches, and Tcl scripting support. Refer to the *Questa Intel FPGA Edition Simulation User Guide* for complete information.

Related Information

Questa Intel FPGA Edition Simulation User Guide

2.1. Quick Start Example (QuestaSim with Verilog)

You can adapt the following RTL simulation example to get started quickly with QuestaSim:

1. To specify your EDA simulator and executable path, type the following Tcl package command in the Quartus Prime tcl shell window:

   `set_user_option -name EDA_TOOL_PATH_QUESTASIM <questasim executable path>`

   `set_global_assignment -name EDA_SIMULATION_TOOL "QuestaSim (Verilog)"`

2. Compile simulation model libraries using one of the following methods:
   - To automatically compile all required simulation model libraries for your design in your supported simulator, click **Tools ➤ Launch Simulation Library Compiler**. Specify options for your simulation tool, language, target device family, and output location, and then click **OK**.
   - Type the following commands to create and map Intel FPGA simulation libraries manually, and then compile the models manually:

     ```
     vlib <lib1>_ver
     vmap <lib1>_ver <lib1>_ver
     vlog -work <lib1> <lib1>
     ```

   Use the compiled simulation model libraries during simulation of your design. Refer to your EDA simulator's documentation for information about running simulation.

3. Compile your design and testbench files:

   ```
   vlog -work work <design or testbench name>.v
   ```

4. Load the design:

```
vsim -L work -L <lib1>_ver -L <lib2>_ver work.<testbench name>
```

2.2. QuestaSim Simulator Guidelines

The following guidelines apply to simulation of Quartus Prime designs with QuestaSim.

2.2.1. Passing Parameter Information from Verilog HDL to VHDL

You must use in-line parameters to pass values from Verilog HDL to VHDL.

By default, the **x_on_violation_option** logic option is enabled for all design registers, resulting in an output of "X" at timing violation. To disable "X" propagation at timing violations on a specific register, disable the **x_on_violation_option** logic option for the specific register, as shown in the following example from the Quartus Prime Settings File (`.qsf`).

```
set_instance_assignment -name X_ON_VIOLATION_OPTION OFF -to \
<register_name>
```

Example 1. In-line Parameter Passing Example

```
lpm_add_sub#(.lpm_width(12), .lpm_direction("Add"),
.lpm_type("LPM_ADD_SUB"),
.lpm_hint("ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO" ))

lpm_add_sub_component (
        .dataa (dataa),
        .datab (datab),
        .result (sub_wire0)
);
```

Note: The sequence of the parameters depends on the sequence of the GENERIC in the VHDL component declaration.

2.2.2. Viewing Simulation Messages

QuestaSim simulator error and warning messages are tagged with a `vsim` or `vcom` code. To determine the cause and resolution for a `vsim` or `vcom` error or warning, use the `verror` command.

For example, QuestaSim may return the following error:

```
# ** Error: C:/altera_trn/DUALPORT_TRY/simulation/questa/DUALPORT_TRY.vho(31):
 (vcom-1136) Unknown identifier "stratixiv"
```

In this case, type the following command:

`verror 1136`
The following description appears:

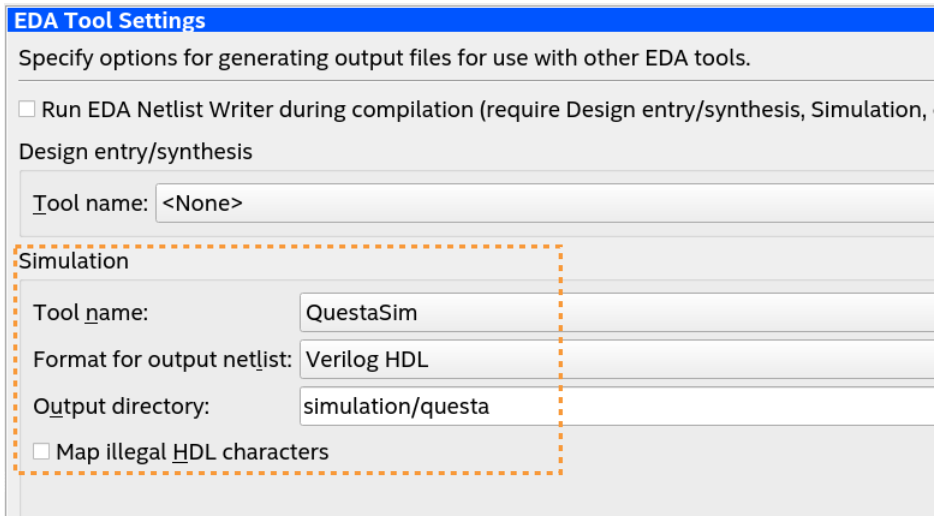
```
# vcom Message # 1136:
# The specified name was referenced but was not found. This indicates
# that either the name specified does not exist or is not visible at
# this point in the code.
```

2.2.3. Generating Signal Activity Data for Power Analysis

To generate and use simulation signal activity data for power analysis:

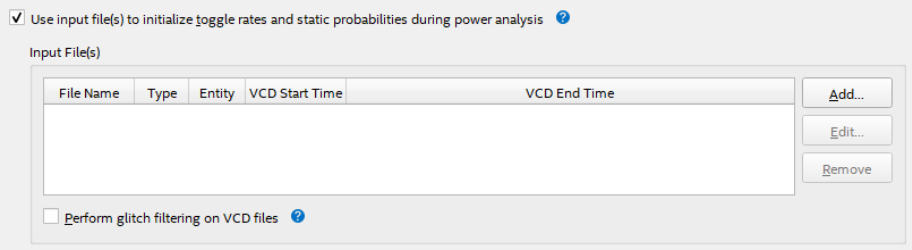
1. To run full compilation on your design, click **Processing ➤ Start Compilation**.
2. To specify settings for simulation output, click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ Simulation**. Select your simulator in **Tool name** and the **Format for output netlist** and **Output directory**.

Figure 11. EDA Tool Settings for Simulation



3. Turn on **Map illegal HDL characters**. This setting directs the EDA Netlist Writer to map illegal characters for VHDL or Verilog HDL, and results in more accurate data for power analysis.
4. Click the **Power Analyzer Settings** page.
5. Under **Input file**, turn on **Use input files to initialize toggle rates and static probabilities during power analysis**.

Figure 12. Specifying Power Analysis Input Files



6. To specify a `.vcd` for power analysis, click **Add** and specify the **File name**, **Entity**, and **Simulation period** for the `.vcd`, and click **OK**.
7. To enable glitch filtering during power analysis with the `.vcd` you generate, turn on **Perform glitch filtering on VCD files**.
8. To run the power analysis, click **Start** on the **Power Analysis** step in the Compilation Dashboard. View the toggle rates in the power analysis results.

2.2.4. Viewing Simulation Waveforms

QuestaSim automatically generates a Wave Log Format File (`.wlf`) following simulation. You can use the `.wlf` to generate a waveform view.

To view a waveform from a `.wlf` through QuestaSim, perform the following steps:

1. Type `vsim` at the command line. The **QuestaSim** dialog box appears.
2. Click **File ➤ Datasets**. The **Datasets Browser** dialog box appears.
3. Click **Open** and select your `.wlf`.
4. Click **Done**.
5. In the Object browser, select the signals that you want to observe.
6. Click **Add ➤ Wave**, and then click **Selected Signals**.
   You must first convert the `.vcd` to a `.wlf` before you can view a waveform in QuestaSim.
7. To convert the `.vcd` to a `.wlf`, type the following at the command-line:

   ```
   vcd2wlf <example>.vcd <example>.wlf
   ```

8. After conversion, view the `.wlf` waveform in QuestaSim.

2.3. Using the Qrun Flow

The Quartus Prime Pro Edition software now supports a new Qrun flow for IP generation. The Qrun flow optionally creates simulation files, including the functional simulation model, and any testbench (or example design).

The Qrun flow, for use with only the QuestaSim and Questa Intel FPGA Edition simulators, is an enhancement over the traditional flow that can automatically combine the compile, optimize, and simulate functions into a single step.

This section describes how to specify the Qrun settings, generate the system or component simulation model and simulator setup scripts, and generate the testbench and example design.

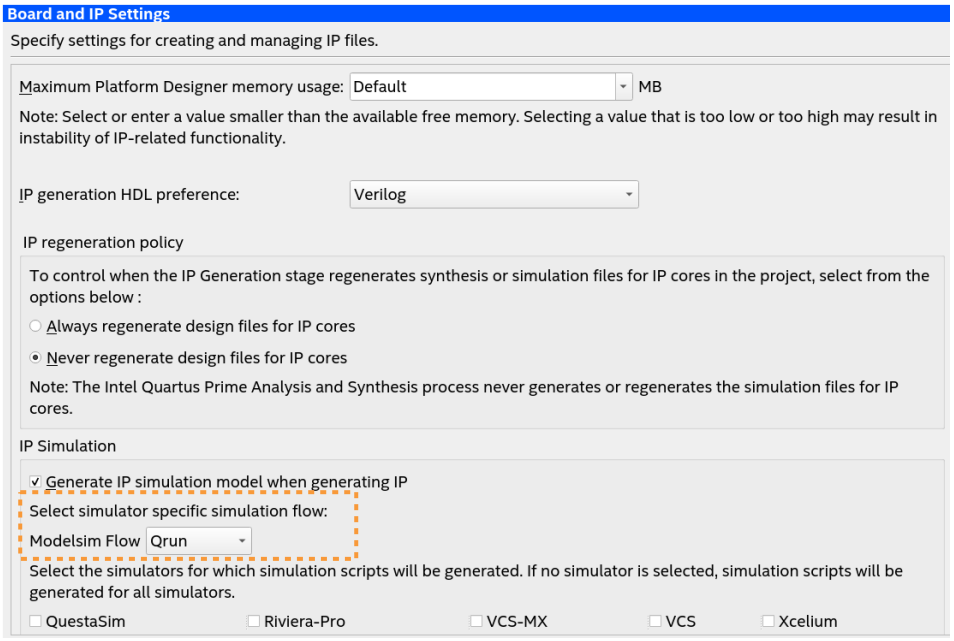
2.3.1. Specifying Simulation File Generation Settings

Before generating simulation files using the Qrun flow, you specify your supported simulator and other options for simulation file generation. These setting impact the generation of simulation files when generating HDL for IP in your project.

To specify simulation file generation settings, follow these steps:

1. In the Quartus Prime Pro Edition software, click **Assignments ➤ Settings ➤ Board and IP Settings**. The **Board and IP Settings** dialog box appears.
2. Under **IP Simulation**, turn on **Generate IP simulation model when generating IP**. Turning on this option enables the remaining settings.
3. For **Select simulator specific simulation flow**, make sure **Qrun** is selected to enable the Qrun flow. The alternative setting runs the **Traditional** flow.

Figure 13. Board and IP Settings Page of Settings Dialog Box



4. To specify one or more specific simulators for which to generate simulation files, enable the checkbox for those simulators. To enable generation for all supported simulators, leave all checkboxes disabled (default setting).

To generate the simulation model and simulator setup scripts for your Platform Designer system or IP component in batch mode, use this command:

```
ip-make-simscript [args] --modelsim_flow=QRUN
```

Type `ip-make-simscript -help` for all available arguments (`[args]`).

2.3.2. Generating the Simulation Model and Setup Scripts

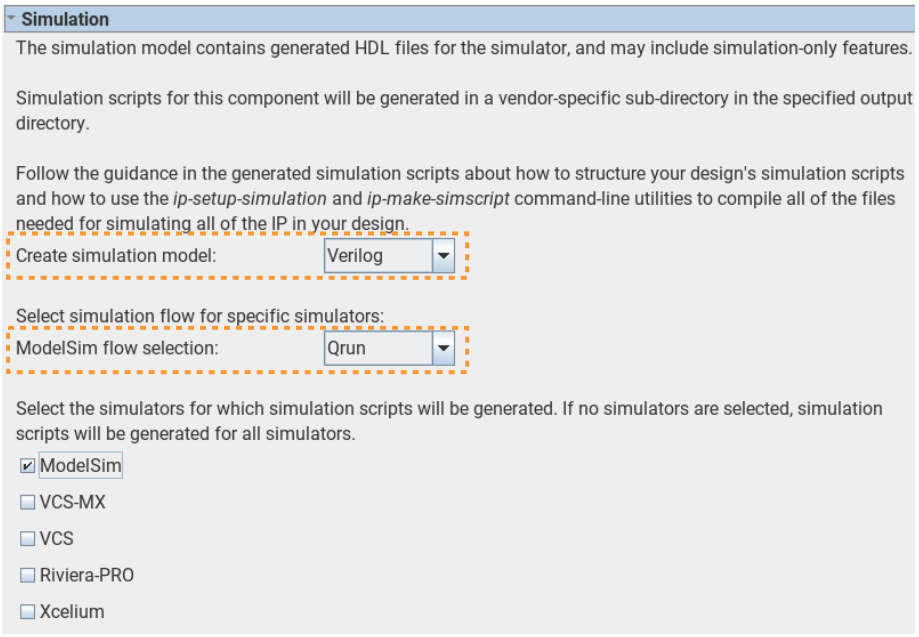
You generate the simulation model and setup scripts for IP components and Platform Designer systems when generating HDL for these IP in your project.

Platform Designer generates the simulation model and setup scripts according to your specifications in Specifying Simulation File Generation Settings.

To generate the simulation model and simulator setup scripts for your Platform Designer system or IP component, follow these steps:

1. In the Quartus Prime Pro Edition software, click **Tools ➤ Platform Designer** and open or create an IP variant or Platform Designer system.
2. After specifying any IP component or system parameters in the parameter editor, click the **Generate HDL** button. The **Generation** dialog box appears.
3. Under **Simulation**, select either **Verilog** or **VHDL** for **Create Simulation Model**. Selecting one of these options makes the **Modelsim flow selection** setting editable.
4. For **Modelsim flow selection**, make sure **Qrun** is selected to enable the Qrun flow. The alternative setting runs the **Traditional** flow.

Figure 14. Generation Dialog Box Settings



5. To specify one or more specific simulators for which to generate simulation files, enable the checkbox for those simulators. To enable generation for all supported simulators, leave all checkboxes disabled (default setting).
6. Click **Generate**. Platform Designer generates the simulation models and setup scripts for your system or IP component in the *<project>/<IP name>*/sim/<vendor> directory.

To generate the simulation model and simulator setup scripts for your Platform Designer system or IP component in batch mode, use this command:

```
qsys-generate <file> [args] --modelsim_flow=QRUN
```

2.3.3. Generating the Testbench System

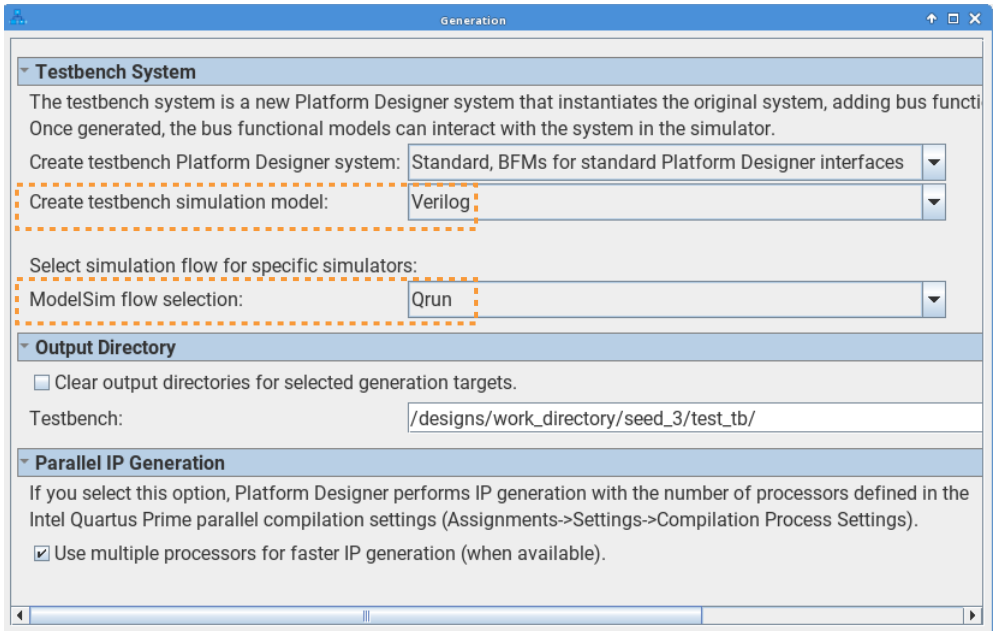
You can optionally generate a testbench system that instantiates the original system, adding bus functional models to drive the top-level interfaces. Once generated, the bus functional models can interact with the system or IP in the simulator.

Platform Designer generates the simulation model and setup scripts according to your specifications in Specifying Simulation File Generation Settings.

To generate the testbench system for a Platform Designer system or IP component, follow these steps:

1. In the Quartus Prime Pro Edition software, click **Tools ➤ Platform Designer** and open or create an IP variant or Platform Designer system.
2. After specifying any IP component or system parameters in the parameter editor, click the **Generate ➤ Generate Testbench System** button. The **Generation** dialog box appears.
3. Under **Testbench System**, select either **Verilog** or **VHDL** for **Create testbench simulation model**. Selecting one of these options makes the **Modelsim flow selection** setting editable.
4. For **Modelsim flow selection**, make sure **Qrun** is selected to enable the Qrun flow. The alternative setting runs the **Traditional** flow.

Figure 15. Generation Dialog Box Settings



5. Click **Generate**. Platform Designer generates the simulation models and setup scripts for your system or IP component under the specified **Output Directory**.

2.3.4. Generating Example Design Simulation Files

When you run **Generate ➤ Generate Example Design**, Platform Designer automatically generates the simulator setup script `msim_setup.tcl` containing `qrun` commands.

2.3.5. Recommendations for Using Qrun

- Mixing of traditional and Qrun generated simulation scripts is not supported.
- Although mixing of vlog and vcom and qrun commands is allowed, you can write testbench scripts using vlog and vcom and generated scripts with qrun or vice versa.

2.4. QuestaSim Simulation Setup Script Example

The Quartus Prime software can generate a `msim_setup.tcl` simulation setup script for IP cores in your design. The script compiles the required device library models, compiles the design files, and elaborates the design with or without simulator optimization. To run the script, type source `msim_setup.tcl` in the simulator Transcript window.

Alternatively, if you are using the simulator at the command line, you can type the following command:

```
vsim -c -do msim_setup.tcl
```

In this example the `top-level-simulate.do` custom top-level simulation script sets the hierarchy variable `TOP_LEVEL_NAME` to `top_testbench` for the design, and sets the variable `QSYS_SIMDIR` to the location of the generated simulation files.

```
# Set hierarchy variables used in the IP-generated files
set TOP_LEVEL_NAME "top_testbench"
set QSYS_SIMDIR "./ip_top_sim"
# Source generated simulation script which defines aliases used below
source $QSYS_SIMDIR/mentor/msim_setup.tcl
# dev_com alias compiles simulation libraries for device library files
dev_com
# com alias compiles IP simulation or Platform Designer model files and/or
Platform Designer model files in the correct order
com
# Compile top level testbench that instantiates your IP
vlog -sv ./top_testbench.sv
# elab alias elaborates the top-level design and testbench
elab
# Run the full simulation
run - all
```

In this example, the top-level simulation files are stored in the same directory as the original IP core, so this variable is set to the IP-generated directory structure. The `QSYS_SIMDIR` variable provides the relative hierarchy path for the generated IP simulation files. The script calls the generated `msim_setup.tcl` script and uses the alias commands from the script to compile and elaborate the IP files required for simulation along with the top-level simulation testbench. You can specify additional simulator elaboration command options when you run the `elab` command, for example, `elab +nowarnTFMPC`. The last command run in the example starts the simulation.

2.5. Sourcing QuestaSim Simulator Setup Scripts

Follow these steps to incorporate the generated or QuestaSim IP simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `sim_top.do`.

```
# # Start of template
# # If the copied and modified template file is "mentor.do", run it
# # as: vsim -c -do mentor.do
# #
# # Source the generated sim script
# source msim_setup.tcl
# # Compile eda/sim_lib contents first
# dev_com
```

```
# # Override the top-level name (so that elab is useful)
# set TOP_LEVEL_NAME top
# # Compile the standalone IP.
# com
# # Compile the top-level
# vlog -sv ../../top.sv
# # Elaborate the design.
# elab
# # Run the simulation
# run -a
# # Report success to the shell
# exit -code 0
# # End of template
```

2. Delete the first two characters of each line (comment and space):

```
# Start of template
# If the copied and modified template file is "mentor.do", run it
# as: vsim -c -do mentor.do
#
# Source the generated sim script
source msim_setup.tcl
# Compile eda/sim_lib contents first
dev_com
# Override the top-level name (so that elab is useful)
set TOP_LEVEL_NAME top
# Compile the standalone IP.
com
# Compile the top-level
vlog -sv ../../top.sv
# Elaborate the design.
elab
# Run the simulation
run -a
# Report success to the shell
exit -code 0
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the location of the simulation's top-level file. For example:

```
set TOP_LEVEL_NAME sim_top vlog -sv ../../sim_top.sv
```

4. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
5. Run the resulting top-level script from the generated simulation directory:

```
vsim –c –do <path to sim_top>.tcl
```

2.6. Unsupported Features

The Quartus Prime software does not support the following simulation features:

- Some versions of QuestaSim support SystemVerilog, PSL assertions, SystemC, and more. For more information about specific feature support, refer to Siemens EDA software documentation.

2.7. Siemens EDA QuestaSim Simulator Support Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Added new *Using the Qrun Flow in Platform Designer* section.
2023.10.02	23.3	• Updated chapter to reflect end of support for ModelSim. • Referenced new Questa Intel FPGA Edition information to new *Questa Intel FPGA Edition Simulation User Guide*. • Removed obsolete *Generating Standard Delay Output for Power Analysis* topic. Generation of SDO files is no longer supported in the Quartus Prime Pro Edition software.
2022.04.13	22.1	• Revised name of Questa Intel FPGA Edition and QuestaSim for latest guidelines throughout.
2021.10.04	21.3	• Added support for Questa Intel FPGA Edition simulator. • Removed support for ModelSim - Intel FPGA Edition simulator. • Updated directory names in *Using Questa Intel FPGA Edition Precompiled Libraries* topic. • Removed note about Maintain Hierarchy limit in *Viewing Simulation Messages* topic. • Replaced "Mentor Graphics" with "Siemens EDA" to reflect current company name.
2021.03.29	21.1	Updated *Generating Signal Activity Data for Power Analysis* topic for latest behavior.
2019.06.19	19.1.0	• Added footnote about ModelSim Remote Desktop limits to "Unsupported Features" topic.
2019.04.01	19.1.0	• Described new support for generation of SDO for use in power analysis.
2017.11.06	17.1.0	• Changed title to ModelSim - Intel FPGA Edition, ModelSim*, and QuestaSim Support* • Removed Simulating Transport Delays and Disabling Timing Violations on Registers topics. Quartus Prime Pro Edition does not support timing simulation. • Added Simulation Library Compiler details and another step to Quick Start Example

3. Synopsys VCS and VCS MX Support

You can include your supported EDA simulator in the Quartus Prime design flow. This document provides guidelines for simulation of Quartus Prime designs with the Synopsys VCS or VCS MX software.

3.1. Quick Start Example (VCS with Verilog)

You can adapt the following RTL simulation example to get started quickly with VCS:

1. To specify your EDA simulator and executable path, type the following Tcl package command in the Quartus Prime tcl shell window:

   `set_user_option -name EDA_TOOL_PATH_VCS <`*VCS executable path*`>`

   `set_global_assignment -name EDA_SIMULATION_TOOL "VCS"`

2. Compile simulation model libraries using the following method:
   - To automatically compile all required simulation model libraries for your design in your supported simulator, click **Tools ➤ Launch Simulation Library Compiler**. Specify options for your simulation tool, language, target device family, and output location, and then click **OK**.

   Use the compiled simulation model libraries during simulation of your design. Refer to your EDA simulator's documentation for information about running simulation.
3. Modify the `simlib_comp.vcs` file to specify your design and testbench files.
4. Type the following to run the VCS simulator:

```
vcs -R -file simlib_comp.vcs
```

3.2. VCS and VCS MX Guidelines

The following guidelines apply to simulation of Intel FPGA designs in the VCS or VCS MX software:

- Do not specify the -v option for `altera_lnsim.sv` because it defines a systemverilog package.
- Add `-verilog` and `+verilog2001ext+.v` options to make sure all **.v** files are compiled as verilog 2001 files, and all other files are compiled as systemverilog files.
- Add the `-lca` option for Stratix V and later families because they include IEEE-encrypted simulation files for VCS and VCS MX.
- Add `-timescale=1ps/1ps` to ensure picosecond resolution.

3.3. VCS Simulation Setup Script Example

The Quartus Prime software can generate a simulation setup script for IP cores in your design. The scripts contain shell commands that compile the required simulation models in the correct order, elaborate the top-level design, and run the simulation for 100 time units by default. You can run these scripts from a Linux command shell.

The scripts for VCS and VCS MX are `vcs_setup.sh` (for Verilog HDL or SystemVerilog) and `vcsmx_setup.sh` (combined Verilog HDL and SystemVerilog with VHDL). Read the generated `.sh` script to see the variables that are available for override when sourcing the script or redefining directly if you edit the script. To set up the simulation for a design, use the command-line to pass variable values to the shell script.

Example 2. Using Command-line to Pass Simulation Variables

```
sh vcsmx_setup.sh\
USER_DEFINED_ELAB_OPTIONS=+rad\
USER_DEFINED_SIM_OPTIONS=+vcs+lic+wait
```

Example 3. Example Top-Level Simulation Shell Script for VCS-MX

```
# Run generated script to compile libraries and IP simulation files
# Skip elaboration and simulation of the IP variation
sh ./ip_top_sim/synopsys/vcsmx/vcsmx_setup.sh SKIP_ELAB=1 SKIP_SIM=1
QSYS_SIMDIR="./ip_top_sim"
#Compile top-level testbench that instantiates IP
vlogan -sverilog ./top_testbench.sv
#Elaborate and simulate the top-level design
vcs –lca –t ps <elaboration control options> top_testbench
simv <simulation control options>
```

Example 4. Example Top-Level Simulation Shell Script for VCS

```
# Run script to compile libraries and IP simulation files
sh ./ip_top_sim/synopsys/vcs/vcs_setup.sh TOP_LEVEL_NAME=”top_testbench”\
# Pass VCS elaboration options to compile files and elaborate top-level
 passed to the script as the TOP_LEVEL_NAME
USER_DEFINED_ELAB_OPTIONS="top_testbench.sv"\
# Pass in simulation options and run the simulation for specified amount of time.
USER_DEFINED_SIM_OPTIONS=”<simulation control options>
```

3.4. Sourcing Synopsys VCS MX Simulator Setup Scripts

Follow these steps to incorporate the generated Synopsys VCS MX simulation scripts for use in top-level project simulation scripts.

1. The generated simulation script contains these template lines. Cut and paste the lines preceding the “helper file” into a new executable file. For example, `vcsmx.sh`.

```
# # Start of template
# # If the copied and modified template file is "vcsmx_sim.sh", run
# # it as: ./vcsmx_sim.sh
# #
# # Do the file copy, dev_com and com steps
# source vcsmx_setup.sh
# SKIP_ELAB=1

# SKIP_SIM=1
#
# # Compile the top level module
```

```
# vlogan +v2k
     +systemverilogext+.sv "$QSYS_SIMDIR/../top.sv"

# # Do the elaboration and sim steps
# # Override the top-level name
# # Override the sim options, so the simulation runs
# # forever (until $finish()).
# source vcsmx_setup.sh
# SKIP_FILE_COPY=1
# SKIP_DEV_COM=1
# SKIP_COM=1
# TOP_LEVEL_NAME="'-top top'"
# USER_DEFINED_SIM_OPTIONS=""
# # End of template
```

2. Delete the first two characters of each line (comment and space), as shown below:

```
# Start of template
# If the copied and modified template file is "vcsmx_sim.sh", run
# it as: ./vcsmx_sim.sh
#
# Do the file copy, dev_com and com steps
source vcsmx_setup.sh
SKIP_ELAB=1
SKIP_SIM=1

# Compile the top level module
vlogan +v2k +systemverilogext+.sv "$QSYS_SIMDIR/../top.sv"

# Do the elaboration and sim steps
# Override the top-level name
# Override the sim options, so the simulation runs
# forever (until $finish()).
source vcsmx_setup.sh
SKIP_FILE_COPY=1
SKIP_DEV_COM=1
SKIP_COM=1
TOP_LEVEL_NAME="'-top top'"
USER_DEFINED_SIM_OPTIONS=""
# End of template
```

3. Modify the TOP_LEVEL_NAME and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME="'-top sim_top'"
```

4. Make the appropriate changes to the compilation of your top-level file, for example:

```
vlogan +v2k +systemverilogext+.sv "$QSYS_SIMDIR/../sim_top.sv"
```

5. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
6. Run the resulting top-level script from the generated simulation directory by specifying the path to `vcsmx_sim.sh`.

3.5. Sourcing Synopsys VCS Simulator Setup Scripts

Follow these steps to incorporate the generated Synopsys VCS simulation scripts into a top-level project simulation script.

1. The generated simulation script contains these template lines. Cut and paste the lines preceding the "helper file" into a new executable file. For example, `synopsys_vcs.f`.

```
# # Start of template
# # If the copied and modified template file is "vcs_sim.sh", run it
# # as: ./vcs_sim.sh
# #
# # Override the top-level name
# # specify a command file containing elaboration options
# # (system verilog extension, and compile the top-level).
# # Override the sim options, so the simulation
# # runs forever (until $finish()).
# source vcs_setup.sh
# TOP_LEVEL_NAME=top
# USER_DEFINED_ELAB_OPTIONS="'-f ../../../synopsys_vcs.f'"
# USER_DEFINED_SIM_OPTIONS=""
#
# # helper file: synopsys_vcs.f
# +systemverilogext+.sv
# ../../../top.sv
# # End of template
```

2. Delete the first two characters of each line (comment and space) for the `vcs.sh` file, as shown below:

```
# Start of template
# If the copied and modified template file is "vcs_sim.sh", run it
# as: ./vcs_sim.sh
#
# Override the top-level name
# specify a command file containing elaboration options
# (system verilog extension, and compile the top-level).
# Override the sim options, so the simulation
# runs forever (until $finish()).
source vcs_setup.sh
TOP_LEVEL_NAME=top
USER_DEFINED_ELAB_OPTIONS="'-f ../../../synopsys_vcs.f'"
USER_DEFINED_SIM_OPTIONS=""
```

3. Delete the first two characters of each line (comment and space) for the `synopsys_vcs.f` file, as shown below:

```
# helper file: synopsys_vcs.f
 +systemverilogext+.sv
 ../../../top.sv
# End of template
```

4. Modify the `TOP_LEVEL_NAME` and compilation step appropriately, depending on the simulation's top-level file. For example:

```
TOP_LEVEL_NAME=sim_top
```

5. If necessary, add the `QSYS_SIMDIR` variable to point to the location of the generated IP simulation files. Specify any other changes required to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
6. Run the resulting top-level script from the generated simulation directory by specifying the path to `vcs_sim.sh`.

3.6. Synopsys VCS and VCS MX Support Revision History

Document Version	Quartus Prime Version	Changes
2022.04.13	22.1	• Revised name of Questa Intel FPGA Edition and QuestaSim for latest guidelines throughout.
2017.11.06	17.1	• Removed Simulating Transport Delays and Disabling Timing Violations on Registers topics. Quartus Prime Pro Edition does not support timing simulation. • Added Simulation Library Compiler details and another step to Quick Start Example

4. Aldec Active-HDL and Riviera-PRO Support

You can include your supported EDA simulator in the Quartus Prime design flow. This chapter provides specific guidelines for simulation of Quartus Prime designs with the Aldec Active-HDL or Riviera-PRO software.

4.1. Quick Start Example (Active-HDL VHDL)

You can adapt the following RTL simulation example to get started quickly with Active-HDL:

1. To specify your EDA simulator and executable path, type the following Tcl package command in the Quartus Prime Tcl shell window:

   `set_user_option -name EDA_TOOL_PATH_ACTIVEHDL <Active HDL executable path>`

   `set_global_assignment -name EDA_SIMULATION_TOOL "Active-HDL (VHDL)"`

2. Compile simulation model libraries using one of the following methods:
   - To automatically compile all required simulation model libraries for your design in your supported simulator, click **Tools ➤ Launch Simulation Library Compiler**. Specify options for your simulation tool, language, target device family, and output location, and then click **OK**.
   - Compile Intel FPGA simulation models manually:

     ```
     vlib <library1> <altera_library1>
     vcom -strict93 -dbg -work <library1> <lib1_component/pack.vhd> \
        <lib1.vhd>
     ```

   Use the compiled simulation model libraries during simulation of your design. Refer to your EDA simulator's documentation for information about running simulation.
3. Open the Active-HDL simulator.
4. Create and open the workspace:

   ```
   createdesign <workspace name> <workspace path>
   opendesign -a <workspace name>.adf
   ```

5. Create the work library and compile the netlist and testbench files:

   ```
   vlib work
   vcom  -strict93  -dbg -work work <output netlist> <testbench file>
   ```

6. Load the design:

   ```
   vsim +access+r -t 1ps +transport_int_delays +transport_path_delays \
   -L work -L <lib1> -L <lib2> work.<testbench module name>
   ```

7. Run the simulation in the Active-HDL simulator.

4.2. Aldec Active-HDL and Riviera-PRO Guidelines

The following guidelines apply to simulating Intel FPGA designs in the Active-HDL or Riviera-PRO software.

Compiling SystemVerilog Files

If your design includes multiple SystemVerilog files, you must compile the SystemVerilog files together with a single `alog` command.

If you have Verilog files and SystemVerilog files in your design, you must first compile the Verilog files, and then compile only the SystemVerilog files in the single `alog` command.

4.3. Using Simulation Setup Scripts

The Quartus Prime software can generate the `rivierapro_setup.tcl` simulation setup script for Intel FPGA IP cores in your design. The use and content of the script file is similar to the `msim_setup.tcl` file that the Quartus Prime software generates for use with the QuestaSim simulator.

4.4. Sourcing Aldec ActiveHDL* or Riviera Pro* Simulator Setup Scripts

Follow these steps to incorporate the generated ActiveHDL* or Riviera Pro* simulation scripts into a top-level project simulation script.

1. The generated simulation script contains the following template lines. Cut and paste these lines into a new file. For example, `sim_top.tcl`.

```
# # TOP-LEVEL TEMPLATE - BEGIN
# #
# # QSYS_SIMDIR is used in the Quartus-generated IP simulation script to
# # construct paths to the files required to simulate the IP in your Quartus
# # project. By default, the IP script assumes that you are launching the
# # simulator from the IP script location. If launching from another
# # location, set QSYS_SIMDIR to the output directory you specified when you
# # generated the IP script, relative to the directory from which you launch
# # the simulator.
# #
# set QSYS_SIMDIR <script generation output directory>
# #
# # Source the generated IP simulation script.
# source $QSYS_SIMDIR/aldec/rivierapro_setup.tcl
# #
# # Set any compilation options you require (this is unusual).
# set USER_DEFINED_COMPILE_OPTIONS <compilation options>
# set USER_DEFINED_VHDL_COMPILE_OPTIONS <compilation options for VHDL>
# set USER_DEFINED_VERILOG_COMPILE_OPTIONS <compilation options for Verilog>
# #
# # Call command to compile the Quartus EDA simulation library.
# dev_com
# #
# # Call command to compile the Quartus-generated IP simulation files.
# com
# #
# # Add commands to compile all design files and testbench files, including
# # the top level. (These are all the files required for simulation other
```

```
# # than the files compiled by the Quartus-generated IP simulation script)
# #
# vlog -sv2k5 <your compilation options> <design and testbench files>
# #
# # Set the top-level simulation or testbench module/entity name, which is
# # used by the elab command to elaborate the top level.
# #
# set TOP_LEVEL_NAME <simulation top>
# #
# # Set any elaboration options you require.
# set USER_DEFINED_ELAB_OPTIONS <elaboration options>
# #
# # Call command to elaborate your design and testbench.
# elab
# #
# # Run the simulation.
# run
# #
# # Report success to the shell.
# exit -code 0
# #
# # TOP-LEVEL TEMPLATE - END
```

2. Delete the first two characters of each line (comment and space):

```
# TOP-LEVEL TEMPLATE - BEGIN
#
# QSYS_SIMDIR is used in the Quartus-generated IP simulation script to
# construct paths to the files required to simulate the IP in your Quartus
# project. By default, the IP script assumes that you are launching the
# simulator from the IP script location. If launching from another
# location, set QSYS_SIMDIR to the output directory you specified when you
# generated the IP script, relative to the directory from which you launch
# the simulator.
#
set QSYS_SIMDIR <script generation output directory>
#
# Source the generated IP simulation script.
source $QSYS_SIMDIR/aldec/rivierapro_setup.tcl
#
# Set any compilation options you require (this is unusual).
set USER_DEFINED_COMPILE_OPTIONS <compilation options>
set USER_DEFINED_VHDL_COMPILE_OPTIONS <compilation options for VHDL>
set USER_DEFINED_VERILOG_COMPILE_OPTIONS <compilation options for Verilog>
#
# Call command to compile the Quartus EDA simulation library.
dev_com
#
# Call command to compile the Quartus-generated IP simulation files.
com
#
# Add commands to compile all design files and testbench files, including
# the top level. (These are all the files required for simulation other
# than the files compiled by the Quartus-generated IP simulation script)
#
vlog -sv2k5 <your compilation options> <design and testbench files>
#
# Set the top-level simulation or testbench module/entity name, which is
# used by the elab command to elaborate the top level.
#
set TOP_LEVEL_NAME <simulation top>
#
# Set any elaboration options you require.
set USER_DEFINED_ELAB_OPTIONS <elaboration options>
#
# Call command to elaborate your design and testbench.
elab
#
# Run the simulation.
run
```

```
 #
 # Report success to the shell.
 exit -code 0
 #
 # TOP-LEVEL TEMPLATE - END
```

3. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes that you require to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
4. Refer to the following `sim_top.tcl` example content, where this file is in the same `aldec/` sub-folder as the `rivierapro_setup.tcl` file.

```
# TOP-LEVEL TEMPLATE - BEGIN
#
# QSYS_SIMDIR is used in the Quartus-generated IP simulation script to
# construct paths to the files required to simulate the IP in your Quartus
# project. By default, the IP script assumes that you are launching the
# simulator from the IP script location. If launching from another
# location, set QSYS_SIMDIR to the output directory you specified when you
# generated the IP script, relative to the directory from which you launch
# the simulator.
#
set QSYS_SIMDIR ../
#
# Source the generated IP simulation script.
 source $QSYS_SIMDIR/aldec/rivierapro_setup.tcl
 #
 # Set any compilation options you require (this is unusual).
set USER_DEFINED_COMPILE_OPTIONS ""
set USER_DEFINED_VHDL_COMPILE_OPTIONS ""
set USER_DEFINED_VERILOG_COMPILE_OPTIONS ""
#
 # Call command to compile the Quartus EDA simulation library.
dev_com
 #
 # Call command to compile the Quartus-generated IP simulation files.
com
 #
 # Add commands to compile all design files and testbench files, including
 # the top level. (These are all the files required for simulation other
 # than the files compiled by the Quartus-generated IP simulation script)
 #
vlog -sv2k5 $QSYS_SIMDIR/PLL_RAM.v
vlog -sv2k5 $QSYS_SIMDIR/testbench_1.v
 #
 # Set the top-level simulation or testbench module/entity name, which is
 # used by the elab command to elaborate the top level.
 #
set TOP_LEVEL_NAME tb
 #
 # Set any elaboration options you require.
 set USER_DEFINED_ELAB_OPTIONS ""
 #
 # Call command to elaborate your design and testbench.
 elab
 #
 # Run the simulation.
run -all
 #
 # Report success to the shell.
 exit -code 0
 #
 # TOP-LEVEL TEMPLATE - END
```

5. To view all available options, invoke the Active-HDL or Riviera-PRO license and launch the simulator by typing `vsim` in command-line mode. After the simulator launches, type `help` in the simulator Console panel. To view options related to a specific command, for example the `vsim` simulation command, type `help vsim` in the simulator Console panel.
6. Run the new top-level script from the generated simulation directory in command-line mode. To run the simulation in GUI mode, type the following:

```
vsim -gui -l log.txt +access +r -lib dsn tb -do sim_top.tcl
```

To run the simulation in command-line mode, type the following:

```
vsim –c –do sim_top.tcl
```

4.5. Aldec Active-HDL and Riviera-PRO * Support Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Updated script content in *Sourcing Aldec ActiveHDL or Riviera Pro* Simulator Setup Scripts* topic.
2023.10.02	23.3	• Updated chapter to reflect end of support for generation of ModelSim files in favor of QuestaSim.
2022.04.13	22.1	• Revised name of Questa Intel FPGA Edition and QuestaSim for latest guidelines throughout.
2017.11.06	17.1	• Removed Simulating Transport Delays and Disabling Timing Violations on Registers topics. Quartus Prime Pro Edition does not support timing simulation. • Added Simulation Library Compiler details and another step to Quick Start Example

5. Cadence Xcelium Parallel Simulator Support

You can include your supported EDA simulator in the Quartus Prime Pro Edition design flow. This chapter provides specific guidelines for simulation of Quartus Prime Pro Edition designs with the Cadence Xcelium Parallel Simulator software.

5.1. Using the Command-Line Interface

The Quartus Prime Pro Edition software provides command-line support for the Xcelium Parallel Simulator.

The following Xcelium simulation executables are available:

Table 13. Xcelium Simulation Executables

Program	Function
`xrun`	`xrun` compiles and runs your design based on the compilation and run options you define.
`xmvlog`	`xmvlog` compiles your Verilog HDL code and performs syntax and static semantics checks.
`xmvhdl`	`xmvhdl` compiles your VHDL code and performs syntax and static semantics checks.
`xmelab`	Elaborates the design hierarchy and determines signal connectivity.
`xmsim`	Runs mixed-language simulation. This program is the simulation kernel that performs event scheduling and executes the simulation code.

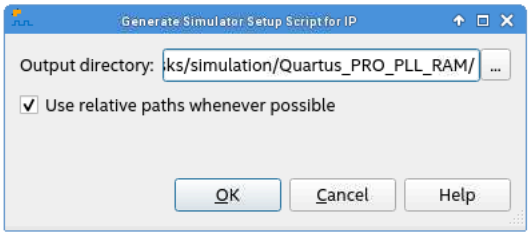
5.2. Generating Simulator Setup Script Templates

You can use simulator setup scripts to help you readily simulate IP cores in your design.

Follow these steps to generate the vendor-specific simulator setup script templates for the IP modules in your design. You can then customize these templates for your specific simulation goals.

1. To compile your design, click **Processing ➤ Start Compilation**. The Messages window indicates when compilation is complete.
2. Click **Tools ➤ Generate Simulator Setup Script for IP**.
3. Retain the default settings for the **Output directory** and also the **Use relative paths whenever possible** option.
4. To generate the setup script templates and vendor-specific sub-folders, including `xcelium/` and `common/` in the specified output directory, click **OK**.

Figure 16. Generate Simulator Setup Script for IP Dialog Box



5.3. Sourcing Cadence Xcelium Simulator Setup Scripts

1. The generated `xcelium/xmsim_setup.sh` simulation script contains the following template lines. Cut and paste these lines into a new top-level script, for example `xmsim.sh`. This new top-level script calls the generated simulation script, `xmsim_setup.sh`.

```
# # TOP-LEVEL TEMPLATE - BEGIN
# #
# # QSYS_SIMDIR is used in the Quartus-generated IP simulation script to
# # construct paths to the files required to simulate the IP in your Quartus
# # project. By default, the IP script assumes that you are launching the
# # simulator from the IP script location. If launching from another
# # location, set QSYS_SIMDIR to the output directory you specified when you
# # generated the IP script, relative to the directory from which you launch
# # the simulator. In this case, you must also copy the generated files
# # "cds.lib" and "hdl.var" - plus the directory "cds_libs" if generated -
# # into the location from which you launch the simulator, or incorporate
# # into any existing library setup.
# #
# # Run Quartus-generated IP simulation script once to compile Quartus EDA
# # simulation libraries and Quartus-generated IP simulation files, and copy
# # any ROM/RAM initialization files to the simulation directory.
# # - If necessary, specify any compilation options:
# #   USER_DEFINED_COMPILE_OPTIONS
# #   USER_DEFINED_VHDL_COMPILE_OPTIONS applied to vhdl compiler
# #   USER_DEFINED_VERILOG_COMPILE_OPTIONS applied to verilog compiler
# #
# source <script generation output directory>/xcelium/xcelium_setup.sh \
# SKIP_ELAB=1 \
# SKIP_SIM=1 \
# USER_DEFINED_COMPILE_OPTIONS=<compilation options for your design> \
# USER_DEFINED_VHDL_COMPILE_OPTIONS=<VHDL compilation options for your
design> \
# USER_DEFINED_VERILOG_COMPILE_OPTIONS=<Verilog compilation options for your
design> \
# QSYS_SIMDIR=<script generation output directory>
# #
# # Compile all design files and testbench files, including the top level.
# # (These are all the files required for simulation other than the files
# # compiled by the IP script)
# #
# xmvlog <compilation options> <design and testbench files>
# #
# # TOP_LEVEL_NAME is used in this script to set the top-level simulation or
# # testbench module/entity name.
# #
# # Run the IP script again to elaborate and simulate the top level:
# # - Specify TOP_LEVEL_NAME and USER_DEFINED_ELAB_OPTIONS.
# # - Override the default USER_DEFINED_SIM_OPTIONS. For example, to run
# #   until $finish(), set to an empty string: USER_DEFINED_SIM_OPTIONS="".
# #
```

```
# source <script generation output directory>/xcelium/xcelium_setup.sh \
# SKIP_FILE_COPY=1 \
# SKIP_DEV_COM=1 \
# SKIP_COM=1 \
# TOP_LEVEL_NAME=<simulation top> \
# USER_DEFINED_ELAB_OPTIONS=<elaboration options for your design> \
# USER_DEFINED_SIM_OPTIONS=<simulation options for your design>
# #
# # TOP-LEVEL TEMPLATE - END
```

2. Delete the first two characters of each line (comment and space):

```
 # TOP-LEVEL TEMPLATE - BEGIN
 #
 # QSYS_SIMDIR is used in the Quartus-generated IP simulation script to
 # construct paths to the files required to simulate the IP in your Quartus
 # project. By default, the IP script assumes that you are launching the
 # simulator from the IP script location. If launching from another
 # location, set QSYS_SIMDIR to the output directory you specified when you
 # generated the IP script, relative to the directory from which you launch
 # the simulator. In this case, you must also copy the generated files
 # "cds.lib" and "hdl.var" - plus the directory "cds_libs" if generated -
 # into the location from which you launch the simulator, or incorporate
 # into any existing library setup.
 #
 # Run Quartus-generated IP simulation script once to compile Quartus EDA
 # simulation libraries and Quartus-generated IP simulation files, and copy
 # any ROM/RAM initialization files to the simulation directory.
 # - If necessary, specify any compilation options:
 #   USER_DEFINED_COMPILE_OPTIONS
 #   USER_DEFINED_VHDL_COMPILE_OPTIONS applied to vhdl compiler
 #   USER_DEFINED_VERILOG_COMPILE_OPTIONS applied to verilog compiler
 #
 source <script generation output directory>/xcelium/xcelium_setup.sh \
 SKIP_ELAB=1 \
 SKIP_SIM=1 \
 USER_DEFINED_COMPILE_OPTIONS=<compilation options for your design> \
 USER_DEFINED_VHDL_COMPILE_OPTIONS=<VHDL compilation options for your
design> \
 USER_DEFINED_VERILOG_COMPILE_OPTIONS=<Verilog compilation options for your
design> \
 QSYS_SIMDIR=<script generation output directory>
 #
 # Compile all design files and testbench files, including the top level.
 # (These are all the files required for simulation other than the files
 # compiled by the IP script)
 #
 xmvlog <compilation options> <design and testbench files>
 #
 # TOP_LEVEL_NAME is used in this script to set the top-level simulation or
 # testbench module/entity name.
 #
 # Run the IP script again to elaborate and simulate the top level:
 # - Specify TOP_LEVEL_NAME and USER_DEFINED_ELAB_OPTIONS.
 # - Override the default USER_DEFINED_SIM_OPTIONS. For example, to run
 #   until $finish(), set to an empty string: USER_DEFINED_SIM_OPTIONS="".
 #
 source <script generation output directory>/xcelium/xcelium_setup.sh \
 SKIP_FILE_COPY=1 \
 SKIP_DEV_COM=1 \
 SKIP_COM=1 \
 TOP_LEVEL_NAME=<simulation top> \
 USER_DEFINED_ELAB_OPTIONS=<elaboration options for your design> \
 USER_DEFINED_SIM_OPTIONS=<simulation options for your design>
 #
 # TOP-LEVEL TEMPLATE - END
```

3. If necessary, add the QSYS_SIMDIR variable to point to the location of the generated IP simulation files. Specify any other changes that you require to match your design simulation requirements. The scripts offer variables to set compilation or simulation options. Refer to the generated script for details.
4. Refer to the following xmsim.sh example content, where this file is in the same /xcelium sub-folder as the xmsim_setup.sh file.

```
# TOP-LEVEL TEMPLATE - BEGIN
#
# QSYS_SIMDIR is used in the Quartus-generated IP simulation script to
# construct paths to the files required to simulate the IP in your Quartus
# project. By default, the IP script assumes that you are launching the
# simulator from the IP script location. If launching from another
# location, set QSYS_SIMDIR to the output directory you specified when you
# generated the IP script, relative to the directory from which you launch
# the simulator. In this case, you must also copy the generated files
# "cds.lib" and "hdl.var" - plus the directory "cds_libs" if generated -
# into the location from which you launch the simulator, or incorporate
# into any existing library setup.
#
# Run Quartus-generated IP simulation script once to compile Quartus EDA
# simulation libraries and Quartus-generated IP simulation files, and copy
# any ROM/RAM initialization files to the simulation directory.
# - If necessary, specify any compilation options:
#   USER_DEFINED_COMPILE_OPTIONS
#   USER_DEFINED_VHDL_COMPILE_OPTIONS applied to vhdl compiler
#   USER_DEFINED_VERILOG_COMPILE_OPTIONS applied to verilog compiler
#
source ./xcelium_setup.sh \
SKIP_ELAB=1 \
SKIP_SIM=1 \
USER_DEFINED_COMPILE_OPTIONS="" \
USER_DEFINED_VHDL_COMPILE_OPTIONS="" \
USER_DEFINED_VERILOG_COMPILE_OPTIONS="" \
QSYS_SIMDIR=./../
#
# Compile all design files and testbench files, including the top level.
# (These are all the files required for simulation other than the files
# compiled by the IP script)
#
xmvlog $QSYS_SIMDIR/PLL_RAM.v
xmvlog $QSYS_SIMDIR/UP_COUNTER_IP/UP_COUNTER_IP.v
xmvlog $QSYS_SIMDIR/DOWN_COUNTER_IP/DOWN_COUNTER_IP.v
xmvlog $QSYS_SIMDIR/ClockPLL/ClockPLL.v
xmvlog $QSYS_SIMDIR/RAMhub/RAMhub.v
xmvlog $QSYS_SIMDIR/testbench_1.v
#
# TOP_LEVEL_NAME is used in this script to set the top-level simulation or
# testbench module/entity name.
#
# Run the IP script again to elaborate and simulate the top level:
# - Specify TOP_LEVEL_NAME and USER_DEFINED_ELAB_OPTIONS.
# - Override the default USER_DEFINED_SIM_OPTIONS. For example, to run
#   until $finish(), set to an empty string: USER_DEFINED_SIM_OPTIONS="".
#
 source ./xcelium_setup.sh \
SKIP_FILE_COPY=1 \
SKIP_DEV_COM=1 \
SKIP_COM=1 \
TOP_LEVEL_NAME="tb" \
USER_DEFINED_ELAB_OPTIONS="-timescale\ 1ns/1ps\ -NOWARN\ CSINFI" \
USER_DEFINED_SIM_OPTIONS="-GUI"
#
# TOP-LEVEL TEMPLATE - END
```

5. Run the resulting top-level script by typing the following at the command-line:

```
sh xmsim.sh
```

Specify the path to this file if you run it from a different directory.

5.4. Cadence Xcelium Parallel Simulator Support Revision History

Document Version	Quartus Prime Version	Changes
2023.12.07	23.4	• Updated *Using the Command-Line Interface* topic to include `xrun` executable. • Added *Generating Simulator Setup Script Templates* topic. • Updated script content and steps in *Sourcing Cadence Xcelium Simulator Setup Scripts* topic. • Removed obsolete *Sourcing Cadence Incisive Simulator Setup Scripts* topic.
2022.04.13	22.1	• Revised name of Questa Intel FPGA Edition and QuestaSim for latest guidelines throughout.
2021.03.29	21.1	• Removed support for Cadence Incisive Enterprise* and removed document section.
2018.05.07	18.0	• Renamed chapter for Xcelium Parallel Simulator support. • Added Xcelium command-line support. • Updated commands in the Quick Start Example.

6. Quartus Prime Pro Edition User Guide Third-party Simulation Archive

For the latest and previous versions of this user guide, refer to Quartus Prime Pro Edition User Guide: Third-party Simulation. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Intel® Quartus® Prime Pro Edition User Guide

Third-party Synthesis

Updated for Intel® Quartus® Prime Design Suite: **23.4**

This document is part of a collection - Intel® Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

1. Siemens EDA Precision* RTL Synthesis Support

1.1. About Precision RTL Synthesis Support

This section describes Intel® Quartus® Prime software support for integration with the Siemens EDA Precision RTL Synthesis and Precision RTL Plus Synthesis software. This description includes the key design flows, methodologies, and techniques for improving your results for Intel devices by integrating Precision RTL Synthesis. This document assumes that you have appropriate installation and licensing of the Precision RTL software and the Intel Quartus Prime Pro Edition software.

To obtain and license the Precision Synthesis software and documentation, refer to the Siemens EDA website.

Related Information

Siemens EDA Precision RTL website

1.2. Precision RTL Integration Flow

The following steps describe a basic Intel Quartus Prime design flow integrating the Precision RTL synthesis software:

1. Create Verilog HDL or VHDL design files.
2. Create a project in the Precision RTL software that contains the HDL files for your design, select your target device, and set global constraints.
3. Compile the project in the Precision RTL software.
4. Add specific timing constraints, optimization attributes, and compiler directives to optimize the design during synthesis. With the design analysis and cross-probing capabilities of the Precision RTL software, you can identify and improve circuit area and performance issues using prelayout timing estimates.

   Note: For best results, Siemens EDA recommends specifying constraints that are as close as possible to actual operating requirements. Properly setting clock and I/O constraints, assigning clock domains, and indicating false and multicycle paths guide the synthesis algorithms more accurately toward a suitable solution in the shortest synthesis time.
5. Synthesize the project in the Precision RTL software.
6. Create an Intel Quartus Prime project and import the following files generated by the Precision RTL software into the Intel Quartus Prime project:

- The Verilog Quartus Mapping File (`.vqm`) netlist
- Synopsys Design Constraints File (`.sdc`) for Timing Analyzer constraints
- Tcl Script Files (`.tcl`) to set up your Intel Quartus Prime project and pass constraints

7. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

1.2.1. Timing Optimization

If your area or timing requirements are not met, you can change the constraints and resynthesize the design in the Precision RTL software, or you can change the constraints to optimize the design during place-and-route in the Intel Quartus Prime software. Repeat the process until the area and timing requirements are met.

You can use other options and techniques in the Intel Quartus Prime software to meet area and timing requirements. For example, the **WYSIWYG Primitive Resynthesis** option can perform optimizations on your EDIF netlist in the Intel Quartus Prime software.

While simulation and analysis can be performed at various points in the design process, final timing analysis should be performed after placement and routing is complete.

Related Information

- Netlist Optimizations and Physical Synthesis
- Timing Closure and Optimization

1.3. Intel Device Family Support

The Precision RTL software supports all FPGA device families available in the current version of the Intel Quartus Prime Pro Edition software. Support for newly released device families may require an overlay.

1.4. Precision RTL Generated Files

During synthesis, the Precision RTL software produces several intermediate and output files.

Table 1. Precision RTL Software Intermediate and Output Files

File Extension	File Description
`.psp`	Precision RTL Project File.
`.xdb`	Design Database File.
`.rep`(1)	Synthesis Area and Timing Report File.
	continued...

File Extension	File Description
`.vqm`[2]	Technology-specific netlist in `.vqm` file format.
`.tcl`	Forward-annotated Tcl assignments and constraints file. The *<project name>*`.tcl` file is generated for all devices. The `.tcl` file acts as the Intel Quartus Prime Project Configuration file and is used to make basic project and placement assignments, and to create and compile a Intel Quartus Prime project.
`.sdc`	Intel Quartus Prime timing constraints file in Synopsys Design Constraints format. This file is generated automatically if the device uses the Timing Analyzer by default in the Intel Quartus Prime software, and has the naming convention *<project name>*`_pnr_constraints .sdc`.

Related Information

Synthesizing the Design and Evaluating the Results on page 9

1.5. Creating and Compiling a Project in the Precision Synthesis Software

After creating your design files, create a project in the Precision RTL software that contains the basic settings for compiling the design.

1.6. Mapping the Design with Precision RTL

In the next steps, you set constraints and map the design to technology-specific cells. The Precision RTL software maps the design by default to the fastest possible implementation that meets your timing constraints. To accomplish this, you must specify timing requirements for the automatically determined clock sources. With this information, the Precision RTL software performs static timing analysis to determine the location of the critical timing paths. The Precision RTL software achieves the best results for your design when you set as many realistic constraints as possible. Be sure to set constraints for timing, mapping, false paths, multicycle paths, and other factors that control the structure of the implemented design.

Siemens EDA recommends creating an `.sdc` file and adding this file to the **Constraint Files** section of the **Project Files** list. You can create this file with a text editor, by issuing command-line constraint parameters, or by directing the Precision RTL software to generate the file automatically the first time you synthesize your design. By default, the Precision RTL software saves all timing constraints and attributes in two files: `precision_rtl.sdc` and `precision_tech.sdc`. The

(1) The timing report file includes performance estimates that are based on pre-place-and-route information. Use the f_{MAX} reported by the Intel Quartus Prime software after place-and-route for accurate post-place-and-route timing information. The area report file includes post-synthesis device resource utilization statistics that can differ from the resource usage after place-and-route due to black boxes or further optimizations performed during placement and routing. Use the device utilization reported by the Intel Quartus Prime software after place-and-route for final resource utilization results.

(2) The Precision RTL software-generated VQM file is supported by the Intel Quartus Prime software version 10.1 and later.

`precision_rtl.sdc` file contains constraints set on the RTL-level database (post-compilation) and the `precision_tech.sdc` file contains constraints set on the gate-level database (post- synthesis) located in the current implementation directory.

You can also enter constraints at the command line. After adding constraints at the command line, update the `.sdc` file with the `update constraint file` command. You can add constraints that change infrequently directly to the HDL source files with HDL attributes or pragmas.

Note: The Precision RTL `.sdc` file contains all the constraints for the Precision RTL project. For the Intel Quartus Prime software, placement constraints are written in a `.tcl` file and timing constraints for the Timing Analyzer are written in the Intel Quartus Prime `.sdc` file.

1.6.1. Setting Timing Constraints

The Precision RTL software uses timing constraints, based on the industry-standard `.sdc` file format, to deliver optimal results. Missing timing constraints can result in incomplete timing analysis and might prevent timing errors from being detected. The Precision RTL software provides constraint analysis prior to synthesis to ensure that designs are fully and accurately constrained. The *<project name>*`_pnr_constraints.sdc` file, which contains timing constraints in `.sdc` format, is generated by the Intel Quartus Prime software.

Note: Because the `.sdc` file format requires that timing constraints be set relative to defined clocks, you must specify your clock constraints before applying any other timing constraints.

You also can use multicycle path and false path assignments to relax requirements or exclude nodes from timing requirements, which can improve area utilization and allow the software optimizations to focus on the most critical parts of the design.

For details about the syntax of Synopsys Design Constraint commands, refer to the *Precision RTL Synthesis User's Manual* and the *Precision Synthesis Reference Manual*.

1.6.2. Setting Mapping Constraints

Mapping constraints affect how your design is mapped into the target Intel device. You can set mapping constraints in the user interface, in HDL code, or with the `set_attribute` command in the constraint file.

1.6.3. Assigning Pin Numbers and I/O Settings

The Precision RTL software supports assigning device pin numbers, I/O standards, drive strengths, and slew rate settings to top-level ports of the design. You can set these timing constraints with the `set_attribute` command, with the GUI, or by specifying synthesis attributes in your HDL code. These constraints are forward-annotated in the *<project name>*`.tcl` file that is read by the Intel Quartus Prime software during place-and-route and do not affect synthesis.

You can use the `set_attribute` command in the Precision RTL software `.sdc` file to specify pin number constraints, I/O standards, drive strengths, and slow slew-rate settings. The table below describes the format to use for entries in the Precision RTL software constraint file.

Table 2. Constraint File Settings

Constraint	Entry Format for Precision Constraint File
Pin number	`set_attribute -name PIN_NUMBER -value "<pin number>" -port <port name>`
I/O standard	`set_attribute -name IOSTANDARD -value "<I/O Standard>" -port <port name>`
Drive strength	`set_attribute -name DRIVE -value "<drive strength in mA>" -port <port name>`
Slew rate	`set_attribute -name SLEW -value "TRUE | FALSE" -port <port name>`

You also can use synthesis attributes or pragmas in your HDL code to make these assignments.

Example 1. Verilog HDL Pin Assignment

```
//pragma attribute clk pin_number P10;
```

Example 2. VHDL Pin Assignment

```
attribute pin_number : string
attribute pin_number of clk : signal is "P10";
```

You can use the same syntax to assign the I/O standard using the IOSTANDARD attribute, drive strength using the attribute DRIVE, and slew rate using the SLEW attribute.

For more details about attributes and how to set these attributes in your HDL code, refer to the *Precision RTL Reference Manual.*

1.6.4. Assigning I/O Registers

The Precision RTL software performs timing-driven I/O register mapping by default. You can force a register to the device IO element (IOE) using the Complex I/O constraint. This option does not apply if you turn off **I/O pad insertion.**

Note: You also can make the assignment by right-clicking on the pin in the Schematic Viewer.

1.6.5. Disabling I/O Pad Insertion

The Precision RTL software assigns I/O pad atoms (device primitives used to represent the I/O pins and I/O registers) to all ports in the top-level of a design by default. In certain situations, you might not want Precision RTL to add I/O pads to all I/O pins in the design. The Intel Quartus Prime software can compile a design without I/O pads; however, including I/O pads provides the Precision RTL software with more information about the top-level pins in the design.

1.6.5.1. Preventing the Precision RTL Software from Adding I/O Pads

If you are compiling a subdesign as a separate project, I/O pins cannot be primary inputs or outputs of the device; therefore, the I/O pins should not have an I/O pad associated with them.

To prevent the Precision RTL software from adding I/O pads:

- You can use the Precision RTL GUI or add the following command to the project file:

```
setup_design -addio=false
```

1.6.5.2. Preventing the Precision RTL Software from Adding an I/O Pad on an Individual Pin

To prevent I/O pad insertion on an individual pin when you are using a black box, such as DDR or a phase-locked loop (PLL), at the external ports of the design, perform the following steps:

1. Compile your design.
2. Use the Precision RTL GUI to select the individual pin and turn off I/O pad insertion.

Note: You also can make this assignment by attaching the `nopad` attribute to the port in the HDL source code.

1.6.6. Controlling Fan-Out on Data Nets

Fan-out is defined as the number of nodes driven by an instance or top-level port. High fan-out nets can cause significant delays that result in an unroutable net. On a critical path, high fan-out nets can cause longer delays in a single net segment that result in the timing constraints not being met. To prevent this behavior, each device family has a global fan-out value set in the Precision RTL software library. In addition, the Intel Quartus Prime software automatically routes high fan-out signals on global routing lines in the Intel device whenever possible.

To eliminate routability and timing issues associated with high fan-out nets, the Precision RTL software also allows you to override the library default value on a global or individual net basis. You can override the library value by setting a `max_fanout` attribute on the net.

1.7. Synthesizing the Design and Evaluating the Results

During synthesis, the Precision RTL software optimizes the compiled design, and then writes out netlists and reports to the implementation subdirectory of your working directory after the implementation is saved, using the following naming convention:

`<project name>_impl_<number>`

After synthesis is complete, you can evaluate the results for area and timing. The *Precision RTL Synthesis User's Manual* describes different results that can be evaluated in the software.

There are several schematic viewers available in the Precision RTL software: RTL schematic, Technology-mapped schematic, and Critical Path schematic. These analysis tools allow you to quickly and easily isolate the source of timing or area issues, and to make additional constraint or code changes to optimize the design.

1.7.1. Obtaining Accurate Logic Utilization and Timing Analysis Reports

Historically, designers have relied on post-synthesis logic utilization and timing reports to determine the amount of logic their design requires, the size of the device required, and how fast the design runs. However, today's FPGA devices provide a wide variety of advanced features in addition to basic registers and look-up tables (LUTs). The Intel Quartus Prime software has advanced algorithms to take advantage of these features, as well as optimization techniques to increase performance and reduce the amount of logic required for a given design. In addition, designs can contain black boxes and functions that take advantage of specific device features. Because of these advances, synthesis tool reports provide post-synthesis area and timing estimates, but you should use the place-and-route software to obtain final logic utilization and timing reports.

1.8. Guidelines for Intel FPGA IP Cores and Architecture-Specific Features

Intel provides parameterizable IP cores, including the LPMs, and device-specific Intel FPGA IP, and IP available through third-party partners. You can use IP cores by instantiating them in your HDL code or by inferring certain functions from generic HDL code.

If you want to instantiate an IP core such as a PLL in your HDL code, you can instantiate and parameterize the function using the port and parameter definitions, or you can customize a function with the parameter editor. Intel recommends using the IP Catalog and parameter editor, which provides a graphical interface within the Intel Quartus Prime software for customizing and parameterizing any available IP core for the design.

The Precision RTL software automatically recognizes certain types of HDL code and infers the appropriate IP core.

Related Information

- Inferring Intel FPGA IP Cores from HDL Code on page 13
- Recommended HDL Coding Styles, Design Recommendations User Guide

1.8.1. Instantiating IP Cores With IP Catalog-Generated Verilog HDL Files

The IP Catalog generates a Verilog HDL instantiation template file <*output file*>_inst.v and a hollow-body black box module declaration <*output file*>_bb.v for use in your Precision RTL design. Incorporate the instantiation template file, <*output file*>_inst.v, into your top-level design to instantiate the IP core wrapper file, <*output file*>.v.

Include the hollow-body black box module declaration <*output file*>_bb.v in your Precision RTL project to describe the port connections of the black box. Adding the IP core wrapper file <*output file*>.v in your Precision RTL project is optional, but you must add it to your Intel Quartus Prime project along with the Precision RTL generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file <*output file*>.v in your Precision RTL project and turn on the **Exclude file from Compile Phase** option in the Precision RTL software to exclude the file from compilation and to copy the file to the appropriate directory for use by the Intel Quartus Prime software during place-and-route.

1.8.2. Instantiating IP Cores With IP Catalog-Generated VHDL Files

The IP Catalog generates a VHDL component declaration file <*output file*>.cmp and a VHDL instantiation template file <*output file*>_inst.vhd for use in your Precision RTL design. Incorporate the component declaration and instantiation template into your top-level design to instantiate the IP core wrapper file, <*output file*>.vhd.

Adding the IP core wrapper file <*output file*>.vhd in your Precision RTL project is optional, but you must add the file to your Intel Quartus Prime project along with the Precision RTL-generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file <*output file*>.v in your Precision RTL project and turn on the **Exclude file from Compile Phase** option in the Precision RTL software to exclude the file from compilation and to copy the file to the appropriate directory for use by the Intel Quartus Prime software during place-and-route.

1.8.3. Instantiating Intellectual Property With the IP Catalog and Parameter Editor

Many Intel FPGA IP functions include a resource and timing estimation netlist that the Precision RTL software can use to synthesize and optimize logic around the IP efficiently. As a result, the Precision RTL software provides better timing correlation, area estimates, and Quality of Results (QoR) than a black box approach.

To create this netlist file, perform the following steps:

1. Select the IP function in the IP Catalog.
2. Click **Next** to open the Parameter Editor.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

The Intel Quartus Prime software generates a file <*output file*>_syn.v. This netlist contains the "gray box" information for resource and timing estimation, but does not contain the actual implementation. Include this netlist file into your Precision RTL project as an input file. Then include the IP core wrapper file <*output file*>.v|vhd in the Intel Quartus Prime project along with your EDIF or VQM output netlist.

The generated "gray box" netlist file, <*output file*>_syn.v , is always in Verilog HDL format, even if you select VHDL as the output file format.

1.8.4. Instantiating Black Box IP Functions With Generated Verilog HDL Files

You can use the `syn_black_box` or `black_box` compiler directives to declare a module as a black box. The top-level design files must contain the IP port mapping and a hollow-body module declaration. You can apply the directive to the module declaration in the top-level file or a separate file included in the project so that the Precision RTL software recognizes the module is a black box.

Note: The `syn_black_box` and `black_box` directives are supported only on module or entity definitions.

The example below shows a sample top-level file that instantiates `my_verilogIP.v`, which is a simplified customized variation generated by the IP Catalog and Parameter Editor.

Example 3. Top-Level Verilog HDL Code with Black Box Instantiation of IP

```
module top (clk, count);
   input clk;
   output[7:0] count;

   my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule

// Module declaration
// The following attribute is added to create a
// black box for this module.
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
   input clock;
   output[7:0] q;
endmodule
```

1.8.5. Instantiating Black Box IP Functions With Generated VHDL Files

You can use the `syn_black_box` or `black_box` compiler directives to declare a component as a black box. The top-level design files must contain the IP core variation component declaration and port mapping. Apply the directive to the component declaration in the top-level file.

Note: The `syn_black_box` and `black_box` directives are supported only on module or entity definitions.

The example below shows a sample top-level file that instantiates `my_vhdlIP.vhd`, which is a simplified customized variation generated by the IP Catalog and Parameter Editor.

Example 4. Top-Level VHDL Code with Black Box Instantiation of IP

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY top IS
  PORT (
    clk: IN STD_LOGIC ;
    count: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END top;

ARCHITECTURE rtl OF top IS
  COMPONENT my_vhdlIP
```

```
  PORT (
    clock: IN STD_LOGIC ;
    q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
  end COMPONENT;
  attribute syn_black_box : boolean;
  attribute syn_black_box of my_vhdlIP: component is true;
  BEGIN
    vhdlIP_inst : my_vhdlIP PORT MAP (
       clock => clk,
       q => count
    );
END rtl;
```

1.8.6. Inferring Intel FPGA IP Cores from HDL Code

The Precision RTL software automatically recognizes certain types of HDL code and maps arithmetical operators, relational operators, and memory (RAM and ROM), to technology-specific implementations. This functionality allows technology-specific resources to implement these structures by inferring the appropriate Intel function to provide optimal results. In some cases, the Precision RTL software has options that you can use to disable or control inference.

For coding style recommendations and examples for inferring technology-specific architecture in Intel devices, refer to the *Precision RTL Synthesis Style Guide*.

Related Information

Recommended HDL Coding Styles, Design Recommendations User Guide

1.8.6.1. Multipliers

The Precision RTL software detects multipliers in HDL code and maps them directly to device atoms to implement the multiplier in the appropriate type of logic. The Precision RTL software also allows you to control the device resources that are used to implement individual multipliers.

1.8.6.1.1. Controlling DSP Block Inference for Multipliers

By default, the Precision RTL software uses DSP blocks available in devices to implement multipliers. The default setting is **AUTO**, which allows the Precision RTL software to map to logic look-up tables (LUTs) or DSP blocks, depending on the size of the multiplier. You can use the Precision RTL GUI or HDL attributes for direct mapping to only logic elements or to only DSP blocks.

Table 3. Options for dedicated_mult Parameter to Control Multiplier Implementation in Precision RTL

Value	Description
ON	Use only DSP blocks to implement multipliers, regardless of the size of the multiplier.
OFF	Use only logic (LUTs) to implement multipliers, regardless of the size of the multiplier.
AUTO	Use logic (LUTs) or DSP blocks to implement multipliers, depending on the size of the multipliers.

1.8.6.2. Setting the Use Dedicated Multiplier Option

To set the `Use Dedicated Multiplier` option in the Precision RTL GUI, compile the design, and then in the Design Hierarchy browser, right-click the operator for the desired multiplier and click **Use Dedicated Multiplier**.

1.8.6.3. Setting the dedicated_mult Attribute

To control the implementation of a multiplier in your HDL code, use the `dedicated_mult` attribute with the appropriate value as shown in the examples below.

Example 5. Setting the dedicated_mult Attribute in Verilog HDL

```
//synthesis attribute <signal name> dedicated_mult <value>
```

Example 6. Setting the dedicated_mult Attribute in VHDL

```
ATTRIBUTE dedicated_mult: STRING;
ATTRIBUTE dedicated_mult OF <signal name>: SIGNAL IS <value>;
```

The `dedicated_mult` attribute can be applied to signals and wires; it does not work when applied to a register. This attribute can be applied only to simple multiplier code, such as `a = b * c`.

Some signals for which the `dedicated_mult` attribute is set can be removed during synthesis by the Precision RTL software for design optimization. In such cases, if you want to force the implementation, you should preserve the signal by setting the `preserve_signal` attribute to TRUE.

Example 7. Setting the preserve_signal Attribute in Verilog HDL

```
//synthesis attribute <signal name> preserve_signal TRUE
```

Example 8. Setting the preserve_signal Attribute in VHDL

```
ATTRIBUTE preserve_signal: BOOLEAN;
ATTRIBUTE preserve_signal OF <signal name>: SIGNAL IS TRUE;
```

Example 9. Verilog HDL Multiplier Implemented in Logic

```
module unsigned_mult (result, a, b);
   output [15:0] result;
   input [7:0] a;
   input [7:0} b;
   assign result = a * b;
   //synthesis attribute result dedicated_mult OFF
endmodule
```

Example 10. VHDL Multiplier Implemented in Logic

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY unsigned_mult IS
   PORT(
      a: IN std_logic_vector (7 DOWNTO 0);
      b: IN std_logic_vector (7 DOWNTO 0);
      result: OUT std_logic_vector (15 DOWNTO 0));
ATTRIBUTE dedicated_mult: STRING;
END unsigned_mult;

ARCHITECTURE rtl OF unsigned_mult IS
   SIGNAL a_int, b_int: UNSIGNED (7 downto 0);
   SIGNAL pdt_int: UNSIGNED (15 downto 0);
ATTRIBUTE dedicated_mult OF pdt_int: SIGNAL IS "OFF;
```

```
BEGIN
   a_int <= UNSIGNED (a);
   b_int <= UNSIGNED (b);
   pdt_int <= a_int * b_int;
   result <= std_logic_vector(pdt_int);
END rtl;
```

1.8.6.4. Inferring Multiplier-Accumulators and Multiplier-Adders

The Precision RTL software also allows you to control the device resources used to implement multiply-accumulators or multiply-adders in your project or in a particular module.

The Precision RTL software detects multiply-accumulators or multiply-adders in HDL code and infers an ALTMULT_ACCUM or ALTMULT_ADD IP cores so that the logic can be placed in DSP blocks, or the software maps these functions directly to device atoms to implement the multiplier in the appropriate type of logic.

Note: The Precision RTL software supports inference for these functions only if the target device family has dedicated DSP blocks.

For more information about DSP blocks in Intel devices, refer to the appropriate Intel FPGA device family documentation.

For more information about inferring multiply-accumulator and multiply-adder IP cores in HDL code, refer to the Intel *Recommended HDL Coding Styles* and the Siemens EDA *Precision RTL Synthesis Style Guide*.

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Recommendations

1.8.6.5. Controlling DSP Block Inference

By default, the Precision RTL software infers the ALTMULT_ADD or ALTMULT_ACCUM IP cores appropriately in your design. These IP cores allow the Intel Quartus Prime software to select either logic or DSP blocks, depending on the device utilization and the size of the function.

You can use the `extract_mac` attribute to prevent inference of an ALTMULT_ADD or ALTMULT_ACCUM IP cores in a certain module or entity.

Table 4. Options for extract_mac Attribute Controlling DSP Implementation

Value	Description
TRUE	The ALTMULT_ADD or ALTMULT_ACCUM IP core is inferred.
FALSE	The ALTMULT_ADD or ALTMULT_ACCUM IP core is not inferred.

To control inference, use the `extract_mac` attribute with the appropriate value from the examples below in your HDL code.

Example 11. Setting the extract_mac Attribute in Verilog HDL

```
//synthesis attribute <module name> extract_mac <value>
```

Example 12. Setting the extract_mac Attribute in VHDL

```
ATTRIBUTE extract_mac: BOOLEAN;
ATTRIBUTE extract_mac OF <entity name>: ENTITY IS <value>;
```

To control the implementation of the multiplier portion of a multiply-accumulator or multiply-adder, you must use the `dedicated_mult` attribute.

You can use the `extract_mac`, `dedicated_mult`, and `preserve_signal` attributes (in Verilog HDL and VHDL) to implement the given DSP function in logic in the Intel Quartus Prime software.

Example 13. Using extract_mac, dedicated_mult, and preserve_signal in Verilog HDL

```
module unsig_altmult_accuml (dataout, dataa, datab, clk, aclr, clken);
   input [7:0} dataa, datab;
   input clk, aclr, clken;
   output [31:0] dataout;

   reg    [31:0] dataout;
   wire   [15:0] multa;
   wire   [31:0] adder_out;

   assign multa = dataa * datab;

   //synthesis attribute multa preserve_signal TRUE
   //synthesis attribute multa dedicated_mult OFF
   assign adder_out = multa + dataout;

   always @ (posedge clk or posedge aclr)
   begin
      if (aclr)
      dataout <= 0;
      else if (clken)
      dataout <= adder_out;
   end

   //synthesis attribute unsig_altmult_accuml extract_mac FALSE
endmodule
```

Example 14. Using extract_mac, dedicated_mult, and preserve_signal in VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;
ENTITY signedmult_add IS
   PORT(
      a, b, c, d: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      result: OUT STD_LOGIC_VECTOR (15 DOWNTO 0));
   ATTRIBUTE preserve_signal: BOOLEANS;
   ATTRIBUTE dedicated_mult: STRING;
   ATTRIBUTE extract_mac: BOOLEAN;
   ATTRIBUTE extract_mac OF signedmult_add: ENTITY IS FALSE;
END signedmult_add;
ARCHITECTURE rtl OF signedmult_add IS
   SIGNAL a_int, b_int, c_int, d_int : signed (7 DOWNTO 0);
   SIGNAL pdt_int, pdt2_int : signed (15 DOWNTO 0);
   SIGNAL result_int: signed (15 DOWNTO 0);
   ATTRIBUTE preserve_signal OF pdt_int: SIGNAL IS TRUE;
   ATTRIBUTE dedicated_mult OF pdt_int: SIGNAL IS "OFF";
   ATTRIBUTE preserve_signal OF pdt2_int: SIGNAL IS TRUE;
   ATTRIBUTE dedicated_mult OF pdt2_int: SIGNAL IS "OFF";
BEGIN
   a_int <= signed (a);
   b_int <= signed (b);
```

```
   c_int <= signed (c);
   d_int <= signed (d);
   pdt_int <= a_int * b_int;
   pdt2_int <= c_int * d_int;
   result_int <= pdt_int + pdt2_int;
   result <= STD_LOGIC_VECTOR(result_int);
END rtl;
```

1.8.6.6. Inferring RAM and ROM

The Precision RTL software detects memory structures in HDL code and converts them to an operator that infers an ALTSYNCRAM or LPM_RAM_DP IP cores, depending on the device family. The Intel Quartus Prime software then places these functions in memory blocks.

The Precision RTL software supports inference for these functions only if the target Intel FPGA device family has dedicated memory blocks.

For more information about inferring RAM and ROM IP cores in HDL code, refer to the *Precision RTL Synthesis Style Guide*.

Related Information

Intel Quartus Prime Pro Edition User Guide: Design Recommendations

1.9. Siemens EDA Precision* RTL Synthesis Support Revision History

Document Version	Intel Quartus Prime Version	Changes
2023.12.12	23.4	• Added Top FAQs navigation to document cover. • Updated vendor and software product names throughout. • Removed obsolete Intel Quartus Prime Standard Edition and Altera references.
2022.03.28	18.1	Revised linking to documentation archives.
2018.09.24	18.1	Removed reference to obsolete .edf file from "Design Flow" diagram.
2018.05.07	18.0	Corrected trademark symbols on tool names.
2016.10.31	16.1	• Implemented Intel rebranding.

2. Synopsys Synplify* Support

2.1. About Synplify Support

the Intel Quartus Prime software supports use of the Synopsys Synplify software design flows, methodologies, and techniques for achieving optimal results in Intel devices. Synplify support applies to Synplify, Synplify Pro, and Synplify Premier software. This document assumes proper set up, licensing, and basic familiarity with the Synplify software.

This document covers the following information:

- General design flow with the Synplify and Intel Quartus Prime software.
- Synplify software optimization strategies, including timing-driven compilation settings, optimization options, and other attributes.
- Guidelines for use of Quartus Prime IP cores, including guidelines for HDL inference of IP cores.

Related Information

- Synplify Synthesis Techniques with the Intel Quartus Prime Software online training
- Synplify Pro Tips and Tricks online training

2.2. Synplify Software Integration Flow

The following steps describe a basic Intel Quartus Prime software flow integrating the Synplify software:

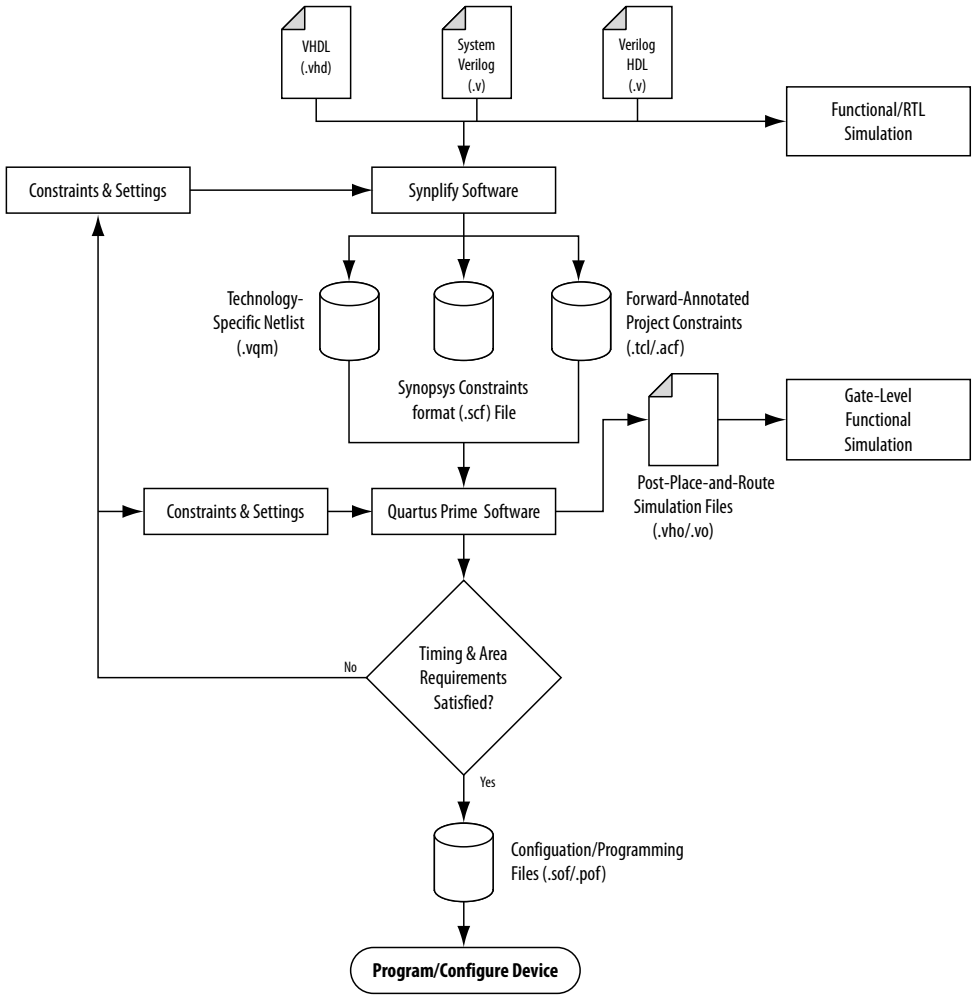
1. Create Verilog HDL (`.v`) or VHDL (`.vhd`) design files.
2. Set up a project in the Synplify software and add the HDL design files for synthesis.
3. Select a target device and add timing constraints and compiler directives in the Synplify software to help optimize the design during synthesis.
4. Synthesize the project in the Synplify software.
5. Create an Intel Quartus Prime project and import the following files generated by the Synplify software into the Intel Quartus Prime software. Use the following files for placement and routing, and for performance evaluation:

- Verilog Quartus Mapping File (`.vqm`) netlist.
- The Synopsys Constraints Format (`.scf`) file for Timing Analyzer constraints.
- The `.tcl` file to set up your Intel Quartus Prime project and pass constraints.

  Note: Alternatively, you can run the Intel Quartus Prime software from within the Synplify software.

6. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

Figure 1. Recommended Design Flow



Related Information

- Synplify Software Generated Files on page 20
- Design Constraints Support on page 21

2.3. Hardware Description Language Support

The Synplify software supports VHDL, Verilog HDL, and SystemVerilog source files. However, only the Synplify Pro and Premier software support mixed synthesis, allowing a combination of VHDL and Verilog HDL or SystemVerilog format source files.

The HDL Analyst that is included in the Synplify software is a graphical tool for generating schematic views of the technology-independent RTL view netlist (`.srs`) and technology-view netlist (`.srm`) files. You can use the Synplify HDL Analyst to analyze and debug your design visually. The HDL Analyst supports cross-probing between the RTL and Technology views, the HDL source code, the Finite State Machine (FSM) viewer, and between the technology view and the timing report file in the Intel Quartus Prime software. A separate license file is required to enable the HDL Analyst in the Synplify software. The Synplify Pro and Premier software include the HDL Analyst.

Related Information

Guidelines for Intel FPGA IP Cores and Architecture-Specific Features on page 30

2.4. Intel Device Family Support

Support for newly released device families may require an overlay. Contact Synopsys for more information.

Related Information

Synopsys Website

2.5. Tool Setup

2.5.1. Specifying the Intel Quartus Prime Software Version

You can specify your version of the Intel Quartus Prime software in **Implementation Options** in the Synplify software. This option ensures that the netlist is compatible with the software version and supports the newest features. Intel recommends using the latest version of the Intel Quartus Prime software whenever possible. If your Intel Quartus Prime software version is newer than the versions available in the **Quartus Version** list, check if there is a newer version of the Synplify software available that supports the current Intel Quartus Prime software version. Otherwise, select the latest version in the list for the best compatibility.

Note: The **Quartus Version** list is available only after selecting an Intel device.

Example 15. Specifying Intel Quartus Prime Software Version at the Command Line

```
set_option -quartus_version <version number>
```

2.6. Synplify Software Generated Files

During synthesis, the Synplify software produces several intermediate and output files.

Table 5. Synplify Intermediate and Output Files

File Extensions	File Description
.vqm	Technology-specific netlist in .vqm file format. A .vqm file is created for all Intel device families supported by the Intel Quartus Prime software.
.scf	Synopsys Constraint Format file containing timing constraints for the Timing Analyzer.
.tcl	Forward-annotated constraints file containing constraints and assignments. A .tcl file for the Intel Quartus Prime software is created for all devices. The .tcl file contains the appropriate Tcl commands to create and set up an Intel Quartus Prime project and pass placement constraints.
.srs	Technology-independent RTL netlist file that can be read only by the Synplify software.
.srm	Technology view netlist file.
.acf	Assignment and Configurations file for backward compatibility with the MAX+PLUS II software. For devices supported by the MAX+PLUS II software, the MAX+PLUS II assignments are imported from the MAX+PLUS II .acf file.
.srr(3)	Synthesis Report file.

Related Information

Synplify Software Integration Flow on page 18

2.7. Design Constraints Support

You can specify timing constraints and attributes by using the SCOPE window of the Synplify software, by editing the .sdc file, or by defining the compiler directives in the HDL source file. The Synplify software forward-annotates many of these constraints to the Intel Quartus Prime software.

After synthesis is complete, do the following steps:

1. Import the .vqm netlist to the Intel Quartus Prime software for place-and-route.
2. Use the .tcl file generated by the Synplify software to forward-annotate your project constraints including device selection. The .tcl file calls the generated .scf to forward-annotate Timing Analyzer timing constraints.

Related Information

- Synplify Software Integration Flow on page 18
- Synplify Optimization Strategies on page 23

(3) This report file includes performance estimates that are often based on pre-place-and-route information. Use the f_{MAX} reported by the Intel Quartus Prime software after place-and-route—it is the only reliable source of timing information. This report file includes post-synthesis device resource utilization statistics that might inaccurately predict resource usage after place-and-route. The Synplify software does not account for black box functions nor for logic usage reduction achieved through register packing performed by the Intel Quartus Prime software. Register packing combines a single register and look-up table (LUT) into a single logic cell, reducing logic cell utilization below the Synplify software estimate. Use the device utilization reported by the Intel Quartus Prime software after place-and-route.

2.7.1. Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script

You can run the Intel Quartus Prime software with a Synplify-generated Tcl script.

To run the Tcl script to set up your project assignments, perform the following steps:

1. Ensure the `.vqm`, `.scf`, and `.tcl` files are located in the same directory.
2. In the Intel Quartus Prime software, on the View menu, point to and click **Tcl Console**. The Intel Quartus Prime Tcl Console opens.
3. At the Tcl Console command prompt, type the following:

```
source <path>/<project name>_cons.tcl
```

2.7.2. Passing Timing Analyzer SDC Timing Constraints to the Intel Quartus Prime Software

The Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry standard constraints format, Synopsys Design Constraints (`.sdc`).

The Synplify-generated `.tcl` file contains constraints for the Intel Quartus Prime software, such as the device specification and any location constraints. Timing constraints are forward-annotated in the Synopsys Constraints Format (`.scf`) file.

Note: Synopsys recommends that you modify constraints using the SCOPE constraint editor window, rather than using the generated `.sdc`, `.scf`, or `.tcl` file.

The following list of Synplify constraints are converted to the equivalent Intel Quartus Prime SDC commands and are forward-annotated to the Intel Quartus Prime software in the `.scf` file:

- `define_clock`
- `define_input_delay`
- `define_output_delay`
- `define_multicycle_path`
- `define_false_path`

All Synplify constraints described above are mapped to SDC commands for the Timing Analyzer.

For syntax and arguments for these commands, refer to the applicable topic in this manual or refer to Synplify Help. For a list of corresponding commands in the Intel Quartus Prime software, refer to the Intel Quartus Prime Help.

Related Information

Timing-Driven Synthesis Settings on page 24

2.7.2.1. Individual Clocks and Frequencies

Specify clock frequencies for individual clocks in the Synplify software with the `define_clock` command. This command is passed to the Intel Quartus Prime software with the `create_clock` command.

2.7.2.2. Input and Output Delay

Specify input delay and output delay constraints in the Synplify software with the `define_input_delay` and `define_output_delay` commands, respectively. These commands are passed to the Intel Quartus Prime software with the `set_input_delay` and `set_output_delay` commands.

2.7.2.3. Multicycle Path

Specify a multicycle path constraint in the Synplify software with the `define_multicycle_path` command. This command is passed to the Intel Quartus Prime software with the `set_multicycle_path` command.

2.7.2.4. False Path

Specify a false path constraint in the Synplify software with the `define_false_path` command. This command is passed to the Intel Quartus Prime software with the `set_false_path` command.

2.8. Simulation and Formal Verification

You can perform simulation and formal verification at various stages in the design process. You can perform final timing analysis after placement and routing is complete.

If area and timing requirements are satisfied, use the files generated by the Intel Quartus Prime software to program or configure the Intel device. If your area or timing requirements are not met, you can change the constraints in the Synplify software or the Intel Quartus Prime software and rerun synthesis. Intel recommends that you provide timing constraints in the Synplify software and any placement constraints in the Intel Quartus Prime software. Repeat the process until area and timing requirements are met.

You can also use other options and techniques in the Intel Quartus Prime software to meet area and timing requirements, such as WYSIWYG Primitive Resynthesis, which can perform optimizations on your `.vqm` netlist within the Intel Quartus Prime software.

Note: In some cases, you might be required to modify the source code if the area and timing requirements cannot be met using options in the Synplify and Intel Quartus Prime software.

2.9. Synplify Optimization Strategies

Combining Synplify software constraints with VHDL and Verilog HDL coding techniques and Intel Quartus Prime software options can help you obtain the results that you require.

For more information about applying attributes, refer to the *Synopsys FPGA Synthesis Reference Manual*.

Related Information

Design Constraints Support on page 21

2.9.1. Using Synplify Premier to Optimize Your Design

Compared to other Synplify products, the Synplify Premier software offers additional physical synthesis optimizations. After typical logic synthesis, the Synplify Premier software places and routes the design and attempts to restructure the netlist based on the physical location of the logic in the Intel device. The Synplify Premier software forward-annotates the design netlist to the Intel Quartus Prime software to perform the final placement and routing. In the default flow, the Synplify Premier software also forward-annotates placement information for the critical path(s) in the design, which can improve the compilation time in the Intel Quartus Prime software.

The physical location annotation file is called <*design name*>`_plc.tcl`. If you open the Intel Quartus Prime software from the Synplify Premier software user interface, the Intel Quartus Prime software automatically uses this file for the placement information.

The Physical Analyst allows you to examine the placed netlist from the Synplify Premier software, which is similar to the HDL Analyst for a logical netlist. You can use this display to analyze and diagnose potential problems.

2.9.2. Using Implementations in Synplify Pro or Premier

You can create different synthesis results without overwriting the existing results, in the Synplify Pro or Premier software, by creating a new implementation from the Project menu. For each implementation, specify the target device, synthesis options, and constraint files. Each implementation generates its own subdirectory that contains all the resulting files, including `.vqm`, `.scf`, and `.tcl` files, from a compilation of the particular implementation. You can then compare the results of the different implementations to find the optimal set of synthesis options and constraints for a design.

2.9.3. Timing-Driven Synthesis Settings

The Synplify software supports timing-driven synthesis with user-assigned timing constraints to optimize the performance of the design.

The Intel Quartus Prime NativeLink feature allows timing constraints that are applied in the Synplify software to be forward-annotated for the Intel Quartus Prime software with an `.scf` file for timing-driven place and route.

The Synplify Synthesis Report File (`.srr`) contains timing reports of estimated place-and-route delays. The Intel Quartus Prime software can perform further optimizations on a post-synthesis netlist from third-party synthesis tools. In addition, designs might contain black boxes or intellectual property (IP) functions that have not been optimized by the third-party synthesis software. Actual timing results are obtained only after the design has been fully placed and routed in the Intel Quartus Prime software. For these reasons, the Intel Quartus Prime post place-and-route timing reports provide a more accurate representation of the design. Use the statistics in these reports to evaluate design performance.

Related Information

Passing Timing Analyzer SDC Timing Constraints to the Intel Quartus Prime Software on page 22

2.9.3.1. Clock Frequencies

For single-clock designs, you can specify a global frequency when using the push-button flow. While this flow is simple and provides good results, it often does not meet the performance requirements for more advanced designs. You can use timing constraints, compiler directives, and other attributes to help optimize the performance of a design. You can enter these attributes and directives directly in the HDL code. Alternatively, you can enter attributes (not directives) into an `.sdc` file with the SCOPE window in the Synplify software.

Use the SCOPE window to set global frequency requirements for the entire design and individual clock settings. Use the **Clocks** tab in the SCOPE window to specify frequency (or period), rise times, fall times, duty cycle, and other settings. Assigning individual clock settings, rather than over-constraining the global frequency, helps the Intel Quartus Prime software and the Synplify software achieve the fastest clock frequency for the overall design. The `define_clock` attribute assigns clock constraints.

2.9.3.2. Multiple Clock Domains

The Synplify software can perform timing analysis on unrelated clock domains. Each clock group is a different clock domain and is treated as unrelated to the clocks in all other clock groups. All clocks in a single clock group are assumed to be related, and the Synplify software automatically calculates the relationship between the clocks. You can assign clocks to a new clock group or put related clocks in the same clock group with the **Clocks** tab in the SCOPE window, or with the `define_clock` attribute.

2.9.3.3. Input and Output Delays

Specify the input and output delays for the ports of a design in the **Input/Output** tab of the SCOPE window, or with the `define_input_delay` and `define_output_delay` attributes. The Synplify software does not allow you to assign the t_{CO} and t_{SU} values directly to inputs and outputs. However, a t_{CO} value can be inferred by setting an external output delay; a t_{SU} value can be inferred by setting an external input delay.

Relationship Between t_{CO} and the Output Delay
t_{CO} = clock period – external output delay

Relationship Between t_{SU} and the Input Delay
t_{SU} = clock period – external input delay

When the `syn_forward_io_constraints` attribute is set to 1, the Synplify software passes the external input and output delays to the Intel Quartus Prime software using NativeLink integration. The Intel Quartus Prime software then uses the external delays to calculate the maximum system frequency.

2.9.3.4. Multicycle Paths

A multicycle path is a path that requires more than one clock cycle to propagate. Specify any multicycle paths in the design in the **Multi-Cycle Paths** tab of the SCOPE window, or with the `define_multicycle_path` attribute. You should specify which paths are multicycle to prevent the Intel Quartus Prime and the Synplify compilers from working excessively on a non-critical path. Not specifying these paths can also result in an inaccurate critical path reported during timing analysis.

2.9.3.5. False Paths

False paths are paths that should be ignored during timing analysis, or should be assigned low (or no) priority during optimization. Some examples of false paths include slow asynchronous resets, and test logic that has been added to the design. Set these paths in the **False Paths** tab of the SCOPE window, or use the `define_false_path` attribute.

2.9.4. FSM Compiler

If the FSM Compiler is turned on, the compiler automatically detects state machines in a design, which are then extracted and optimized. The FSM Compiler analyzes state machines and implements sequential, gray, or one-hot encoding, based on the number of states. The compiler also performs unused-state analysis, optimization of unreachable states, and minimization of transition logic. Implementation is based on the number of states, regardless of the coding style in the HDL code.

If the FSM Compiler is turned off, the compiler does not optimize logic as state machines. The state machines are implemented as HDL code. Thus, if the coding style for a state machine is sequential, the implementation is also sequential.

Use the `syn_state_machine` compiler directive to specify or prevent a state machine from being extracted and optimized. To override the default encoding of the FSM Compiler, use the `syn_encoding` directive.

Table 6. `syn_encoding` Directive Values

Value	Description
Sequential	Generates state machines with the fewest possible flipflops. Sequential, also called binary, state machines are useful for area-critical designs when timing is not the primary concern.
Gray	Generates state machines where only one flipflop changes during each transition. Gray-encoded state machines tend to be glitches.
One-hot	Generates state machines containing one flipflop for each state. One-hot state machines typically provide the best performance and shortest clock-to-output delays. However, one-hot implementations are usually larger than sequential implementations.
Safe	Generates extra control logic to force the state machine to the reset state if an invalid state is reached. You can use the safe value in conjunction with any of the other three values, which results in the state machine being implemented with the requested encoding scheme and the generation of the reset logic.

Example 16. Sample VHDL Code for Applying `syn_encoding` Directive

```
SIGNAL current_state : STD_LOGIC_VECTOR (7 DOWNTO 0);
ATTRIBUTE syn_encoding : STRING;
ATTRIBUTE syn_encoding OF current_state : SIGNAL IS "sequential";
```

By default, the state machine logic is optimized for speed and area, which may be potentially undesirable for critical systems. The safe value generates extra control logic to force the state machine to the reset state if an invalid state is reached.

2.9.4.1. FSM Explorer in Synplify Pro and Premier

The Synplify Pro and Premier software use the FSM Explorer to explore different encoding styles for a state machine automatically, and then implement the best encoding based on the overall design constraints. The FSM Explorer uses the FSM Compiler to identify and extract state machines from a design. However, unlike the FSM Compiler, which chooses the encoding style based on the number of states, the FSM Explorer attempts several different encoding styles before choosing a specific one. The trade-off is that the compilation requires more time to analyze the state machine, but finds an optimal encoding scheme for the state machine.

2.9.5. Optimization Attributes and Options

2.9.5.1. Retiming in Synplify Pro and Premier

The Synplify Pro and Premier software can retime a design, which can improve the timing performance of sequential circuits by moving registers (register balancing) across combinational elements. Be aware that retimed registers incur name changes. You can retime your design from **Implementation Options** or you can use the `syn_allow_retiming` attribute.

2.9.5.2. Maximum Fan-Out

When your design has critical path nets with high fan-out, use the `syn_maxfan` attribute to control the fan-out of the net. Setting this attribute for a specific net results in the replication of the driver of the net to reduce overall fan-out. The `syn_maxfan` attribute takes an integer value and applies it to inputs or registers. The `syn_maxfan` attribute cannot be used to duplicate control signals. The minimum allowed value of the attribute is 4. Using this attribute might result in increased logic resource utilization, thus straining routing resources, which can lead to long compilation times and difficult fitting.

If you must duplicate an output register or an output enable register, you can create a register for each output pin by using the `syn_useioff` attribute.

2.9.5.3. Preserving Nets

During synthesis, the compiler maintains ports, registers, and instantiated components. However, some nets cannot be maintained to create an optimized circuit. Applying the `syn_keep` directive overrides the optimization of the compiler and preserves the net during synthesis. The `syn_keep` directive is a Boolean data type value and can be applied to wires (Verilog HDL) and signals (VHDL). Setting the value to **true** preserves the net through synthesis.

2.9.5.4. Register Packing

Intel devices allow register packing into I/O cells. Intel recommends allowing the Intel Quartus Prime software to make the I/O register assignments. However, you can control register packing with the `syn_useioff` attribute. The `syn_useioff` attribute is a Boolean data type value that can be applied to ports or entire modules. Setting

the value to **1** instructs the compiler to pack the register into an I/O cell. Setting the value to **0** prevents register packing in both the Synplify and Intel Quartus Prime software.

2.9.5.5. Resource Sharing

The Synplify software uses resource sharing techniques during synthesis, by default, to reduce area. Turning off the **Resource Sharing** option on the **Options** tab of the **Implementation Options** dialog box improves performance results for some designs. You can also turn off the option for a specific module with the `syn_sharing` attribute. If you turn off this option, be sure to check the results to verify improvement in timing performance. If there is no improvement, turn on **Resource Sharing**.

2.9.5.6. Preserving Hierarchy

The Synplify software performs cross-boundary optimization by default, which causes the design to flatten to allow optimization. You can use the `syn_hier` attribute to override the default compiler settings. The `syn_hier` attribute applies a string value to modules, architectures, or both. Setting the value to **hard** maintains the boundaries of a module, architecture, or both, but allows constant propagation. Setting the value to **locked** prevents all cross-boundary optimizations. Use the **locked** setting with the partition setting to create separate design blocks and multiple output netlists.

By default, the Synplify software generates a hierarchical `.vqm` file. To flatten the file, set the `syn_netlist_hierarchy` attribute to **0**.

2.9.5.7. Register Input and Output Delays

Two advanced options, `define_reg_input_delay` and `define_reg_output_delay`, can speed up paths feeding a register, or coming from a register, by a specific number of nanoseconds. The Synplify software attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with the `define_clock` attribute). You can use these attributes to add a delay to paths feeding into or out of registers to further constrain critical paths. You can slow down a path that is too highly optimized by setting this attributes to a negative number.

The `define_reg_input_delay` and `define_reg_output_delay` options are useful to close timing if your design does not meet timing goals, because the routing delay after placement and routing exceeds the delay predicted by the Synplify software. Rerun synthesis using these options, specifying the actual routing delay (from place-and-route results) so that the tool can meet the required clock frequency. Synopsys recommends that for best results, do not make these assignments too aggressively. For example, you can increase the routing delay value, but do not also use the full routing delay from the last compilation.

In the SCOPE constraint window, the registers panel contains the following options:

- **Register**—Specifies the name of the register. If you have initialized a compiled design, select the name from the list.
- **Type**—Specifies whether the delay is an input or output delay.
- **Route**—Shrinks the effective period for the constrained registers by the specified value without affecting the clock period that is forward-annotated to the Intel Quartus Prime software.

Use the following Tcl command syntax to specify an input or output register delay in nanoseconds.

Example 17. Input and Output Register Delay

```
define_reg_input_delay {<register>} -route <delay in ns>
define_reg_output_delay {<register>} -route <delay in ns>
```

2.9.5.8. syn_direct_enable

This attribute controls the assignment of a clock-enable net to the dedicated enable pin of a register. With this attribute, you can direct the Synplify mapper to use a particular net as the only clock enable when the design has multiple clock enable candidates.

To use this attribute as a compiler directive to infer registers with clock enables, enter the `syn_direct_enable` directive in your source code, instead of the SCOPE spreadsheet.

The `syn_direct_enable` data type is Boolean. A value of **1** or **true** enables net assignment to the clock-enable pin. The following is the syntax for Verilog HDL:

```
object /* synthesis syn_direct_enable = 1 */ ;
```

2.9.5.9. I/O Standard

For certain Intel devices, specify the I/O standard type for an I/O pad in the design with the **I/O Standard** panel in the Synplify SCOPE window.

The Synplify SDC syntax for the `define_io_standard` constraint, in which the `delay_type` must be either `input_delay` or `output_delay`.

Example 18. define_io_standard Constraint

```
define_io_standard [-disable|-enable] {<objectName>} -delay_type \
[input_delay|output_delay] <columnTclName>{<value>} [<columnTclName>{<value>}...]
```

For details about supported I/O standards, refer to the *Synopsys FPGA Synthesis Reference Manual.*

2.9.6. Intel-Specific Attributes

You can use the `altera_chip_pin_lc`, `altera_io_powerup`, and `altera_io_opendrain` attributes with specific Intel device features, which are forward-annotated to the Intel Quartus Prime project, and are used during place-and-route.

2.9.6.1. altera_chip_pin_lc

Use the `altera_chip_pin_lc` attribute to make pin assignments. This attribute applies a string value to inputs and outputs. Use the attribute only on the ports of the top-level entity in the design. Do not use this attribute to assign pin locations from entities at lower levels of the design hierarchy.

Note: The `altera_chip_pin_lc` attribute is not supported for any MAX series device.

In the SCOPE window, set the value of the `altera_chip_pin_lc` attribute to a pin number or a list of pin numbers.

You can use VHDL code for making location assignments for supported Intel devices. Pin location assignments for these devices are written to the output `.tcl` file.

Note: The `data_out` signal is a 4-bit signal; `data_out[3]` is assigned to pin 14 and `data_out[0]` is assigned to pin 15.

Example 19. Making Location Assignments in VHDL

```
ENTITY sample (data_in : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    data_out: OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
ATTRIBUTE altera_chip_pin_lc : STRING;
ATTRIBUTE altera_chip_pin_lc OF data_out : SIGNAL IS "14, 5, 16, 15";
```

2.9.6.2. altera_io_powerup

Use the `altera_io_powerup` attribute to define the power-up value of an I/O register that has no set or reset. This attribute applies a string value (**high**|**low**) to ports with I/O registers. By default, the power-up value of the I/O register is set to **low**.

2.9.6.3. altera_io_opendrain

Use the `altera_io_opendrain` attribute to specify open-drain mode I/O ports. This attribute applies a boolean data type value to outputs or bidirectional ports for devices that support open-drain mode.

2.10. Guidelines for Intel FPGA IP Cores and Architecture-Specific Features

Intel provides parameterizable IP cores, including LPMs, device-specific Intel FPGA IP cores, and IP available through the Intel FPGA IP Partners Program (AMPP^{SM}). You can use IP cores by instantiating them in your HDL code, or by inferring certain IP cores from generic HDL code.

You can instantiate an IP core in your HDL code with the IP Catalog and configure the IP core with the Parameter Editor, or instantiate the IP core using the port and parameter definition. The IP Catalog and Parameter Editor provide a graphical interface within the Intel Quartus Prime software to customize any available Intel FPGA IP core for the design.

The Synplify software also automatically recognizes certain types of HDL code, and infers the appropriate Intel FPGA IP core when an IP core provides optimal results. The Synplify software provides options to control inference of certain types of IP cores.

Related Information

- Hardware Description Language Support on page 20
- Recommended HDL Coding Styles, Design Recommendations User Guide
- About the IP Catalog Online Help

2.10.1. Instantiating Intel FPGA IP Cores with the IP Catalog

When you use the IP Catalog and Parameter Editor to set up and configure an IP core, the IP Catalog creates a VHDL or Verilog HDL wrapper file <*output file*>.v|vhd that instantiates the IP core.

The Synplify software uses the Intel Quartus Prime timing and resource estimation netlist feature to report more accurate resource utilization and timing performance estimates, and uses timing-driven optimization, instead of treating the IP core as a "black box." Including the generated IP core variation wrapper file in your Synplify project, gives the Synplify software complete information about the IP core.

Note: There is an option in the Parameter Editor to generate a netlist for resource and timing estimation. This option is not recommended for the Synplify software because the software automatically generates this information in the background without a separate netlist. If you do create a separate netlist <*output file*>_syn.v and use that file in your synthesis project, you must also include the <*output file*>.v|vhd file in your Intel Quartus Prime project.

Verify that the correct Intel Quartus Prime version is specified in the Synplify software before compiling the generated file to ensure that the software uses the correct library definitions for the IP core. The **Quartus Version** setting must match the version of the Intel Quartus Prime software used to generate the customized IP core.

In addition, ensure that the *QUARTUS_ROOTDIR* environment variable specifies the installation directory location of the correct Intel Quartus Prime version. The Synplify software uses this information to launch the Intel Quartus Prime software in the background. The environment variable setting must match the version of the Intel Quartus Prime software used to generate the customized IP core.

Related Information

Specifying the Intel Quartus Prime Software Version on page 20

2.10.1.1. Instantiating Intel FPGA IP Cores with IP Catalog Generated Verilog HDL Files

If you turn on the <*output file*>_inst.v option on the Parameter Editor, the IP Catalog generates a Verilog HDL instantiation template file for use in your Synplify design. The instantiation template file, <*output file*>_inst.v, helps to instantiate the IP core variation wrapper file, <*output file*>.v, in your top-level design. Include the IP core variation wrapper file <*output file*>.v in your Synplify project. The Synplify software includes the IP core information in the output .vqm netlist file. You do not need to include the generated IP core variation wrapper file in your Intel Quartus Prime project.

2.10.1.2. Instantiating Intel FPGA IP Cores with IP Catalog Generated VHDL Files

If you turn on the <*output file*>.cmp and <*output file*>_inst.vhd options on the parameter editor, the IP Catalog generates a VHDL component declaration file and a VHDL instantiation template file for use in your Synplify design. These files can help you instantiate the IP core variation wrapper file, <*output file*>.vhd, in your top-level design. Include the <*output file*>.vhd in your Synplify project. The Synplify software includes the IP core information in the output .vqm netlist file. You do not need to include the generated IP core variation wrapper file in your Intel Quartus Prime project.

2.10.1.3. Changing Synplify's Default Behavior for Instantiated Intel FPGA IP Cores

By default, the Synplify software automatically opens the Intel Quartus Prime software in the background to generate a resource and timing estimation netlist for IP cores.

You might want to change this behavior to reduce run times in the Synplify software, because generating the netlist files can take several minutes for large designs, or if the Synplify software cannot access your Intel Quartus Prime software installation to generate the files. Changing this behavior might speed up the compilation time in the Synplify software, but the Quality of Results (QoR) might be reduced.

The Synplify software directs the Intel Quartus Prime software to generate information in two ways:

- Some IP cores provide a "clear box" model—the Synplify software fully synthesizes this model and includes the device architecture-specific primitives in the output .vqm netlist file.
- Other IP cores provide a "gray box" model—the Synplify software reads the resource information, but the netlist does not contain all the logic functionality.

  Note: You need to turn on **Generate netlist** when using the gray box model. For more information, see the Intel Quartus Prime online help.

For these IP cores, the Synplify software uses the logic information for resource and timing estimation and optimization, and then instantiates the IP core in the output .vqm netlist file so the Intel Quartus Prime software can implement the appropriate device primitives. By default, the Synplify software uses the clear box model when available, and otherwise uses the gray box model.

Note: Generation of a timing and area estimation (gray box) netlist is available only for individual Intel FPGA IP, and not for Platform Designer systems.

Related Information

- Including Files for Intel Quartus Prime Placement and Routing Only on page 35
- Synplify Synthesis Techniques with the Intel Quartus Prime Software online training

  Includes more information about design flows using clear box model and gray box model.
- Generating a Netlist for 3rd Party Synthesis Tools online help

2.10.1.4. Instantiating Intellectual Property with the IP Catalog and Parameter Editor

Many Intel FPGA IP cores include a resource and timing estimation netlist that the Synplify software uses to report more accurate resource utilization and timing performance estimates, and uses timing-driven optimization rather than a black box function.

To create this netlist file, perform the following steps:

1. Select the IP core in the IP Catalog.
2. Click **Next** to open the Parameter Editor.
3. Click **Set Up Simulation**, which sets up all the EDA options.
4. Turn on the **Generate netlist** option to generate a netlist for resource and timing estimation and click **OK**.
5. Click **Generate** to generate the netlist file.

The Intel Quartus Prime software generates a file <*output file*>_syn.v. This netlist contains the gray box information for resource and timing estimation, but does not contain the actual implementation. Include this netlist file in your Synplify project. Next, include the IP core variation wrapper file <*output file*>.v|vhd in the Intel Quartus Prime project along with your Synplify .vqm output netlist.

If your IP core does not include a resource and timing estimation netlist, the Synplify software must treat the IP core as a black box.

Related Information

Including Files for Intel Quartus Prime Placement and Routing Only on page 35

2.10.1.5. Instantiating Black Box IP Cores with Generated Verilog HDL Files

Use the syn_black_box compiler directive to declare a module as a black box. The top-level design files must contain the IP port-mapping and a hollow-body module declaration. Apply the syn_black_box directive to the module declaration in the top-level file or a separate file included in the project so that the Synplify software recognizes the module is a black box. The software compiles successfully without this directive, but reports an additional warning message. Using this directive allows you to add other directives.

The example shows a top-level file that instantiates my_verilogIP.v, which is a simple customized variation generated by the IP Catalog.

Example 20. Sample Top-Level Verilog HDL Code with Black Box Instantiation of IP

```
module top (clk, count);
    input clk;
    output [7:0] count;
    my_verilogIP verilogIP_inst (.clock (clk), .q (count));
endmodule
// Module declaration
// The following attribute is added to create a
// black box for this module.
module my_verilogIP (clock, q) /* synthesis syn_black_box */;
    input clock;
    output [7:0] q;
endmodule
```

2.10.1.6. Instantiating Black Box IP Cores with Generated VHDL Files

Use the `syn_black_box` compiler directive to declare a component as a black box. The top-level design files must contain the IP core variation component declaration and port-mapping. Apply the `syn_black_box` directive to the component declaration in the top-level file. The software compiles successfully without this directive, but reports an additional warning message. Using this directive allows you to add other directives.

The example shows a top-level file that instantiates `my_vhdlIP.vhd`, which is a simplified customized variation generated by the IP Catalog.

Example 21. Sample Top-Level VHDL Code with Black Box Instantiation of IP

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY top IS
    PORT (
        clk: IN STD_LOGIC ;
        count: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
END top;

ARCHITECTURE rtl OF top IS
COMPONENT my_vhdlIP
    PORT (
        clock: IN STD_LOGIC ;
        q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
    );
end COMPONENT;
attribute syn_black_box : boolean;
attribute syn_black_box of my_vhdlIP: component is true;
BEGIN
    vhdlIP_inst : my_vhdlIP PORT MAP (
        clock => clk,
        q => count
    );
END rtl;
```

2.10.1.7. Other Synplify Software Attributes for Creating Black Boxes

Instantiating IP as a black box does not provide visibility into the IP for the synthesis tool. Thus, it does not take full advantage of the synthesis tool's timing-driven optimization. For better timing optimization, especially if the black box does not have registered inputs and outputs, add timing models to black boxes by adding the `syn_tpd`, `syn_tsu`, and `syn_tco` attributes.

Example 22. Adding Timing Models to Black Boxes in Verilog HDL

```
module ram32x4(z,d,addr,we,clk);
    /* synthesis syn_black_box syn_tcol="clk->z[3:0]=4.0"
        syn_tpd1="addr[3:0]->[3:0]=8.0"
        syn_tsu1="addr[3:0]->clk=2.0"
        syn_tsu2="we->clk=3.0" */
    output [3:0]z;
    input[3:0]d;
    input[3:0]addr;
    input we
    input clk
endmodule
```

The following additional attributes are supported by the Synplify software to communicate details about the characteristics of the black box module within the HDL code:

- `syn_resources`—Specifies the resources used in a particular black box.
- `black_box_pad_pin`—Prevents mapping to I/O cells.
- `black_box_tri_pin`—Indicates a tri-stated signal.

For more information about applying these attributes, refer to the *Synopsys FPGA Synthesis Reference Manual*.

2.10.2. Including Files for Intel Quartus Prime Placement and Routing Only

In the Synplify software, you can add files to your project that are used only during placement and routing in the Intel Quartus Prime software. This can be useful if you have gray or black boxes for Synplify synthesis that require the full design files to be compiled in the Intel Quartus Prime software.

You can also set the option in a script using the `-job_owner par` option.

The example shows how to define files for a Synplify project that includes a top-level design file, a gray box netlist file, an IP wrapper file, and an encrypted IP file. With these files, the Synplify software writes an empty instantiation of "core" in the `.vqm` file and uses the gray box netlist for resource and timing estimation. The files `core.v` and `core_enc8b10b.v` are not compiled by the Synplify software, but are copied into the place-and-route directory. The Intel Quartus Prime software compiles these files to implement the "core" IP block.

Example 23. Commands to Define Files for a Synplify Project

```
add_file -verilog -job_owner par "core_enc8b10b.v"
add_file -verilog -job_owner par "core.v"
add_file -verilog "core_gb.v"
add_file -verilog "top.v"
```

Note: Generation of a timing and area estimation (gray box) netlist is available only for individual Intel FPGA IP, and not for Platform Designer systems.

2.10.3. Inferring Intel FPGA IP Cores from HDL Code

The Synplify software uses Behavior Extraction Synthesis Technology (BEST) algorithms to infer high-level structures such as RAMs, ROMs, operators, FSMs, and DSP multiplication operations. Then, the Synplify software keeps the structures abstract for as long as possible in the synthesis process. This allows the use of technology-specific resources to implement these structures by inferring the appropriate Intel FPGA IP core when an IP core provides optimal results.

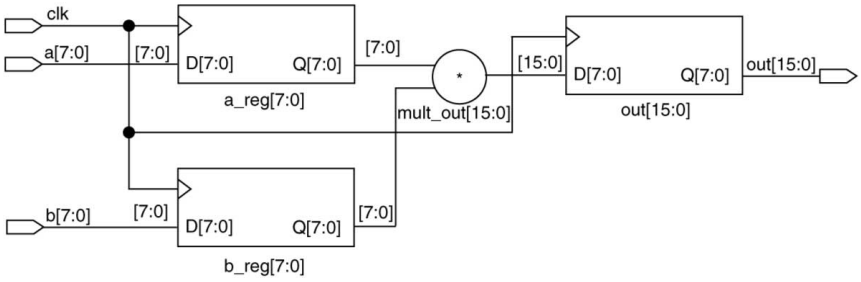
Related Information

Recommended HDL Coding Styles, Design Recommendations User Guide

2.10.3.1. Inferring Multipliers

The figure shows the HDL Analyst view of an unsigned 8 × 8 multiplier with two pipeline stages after synthesis in the Synplify software. This multiplier is converted into an ALTMULT_ADD or ALTMULT_ACCUM IP core. For devices with DSP blocks, the software might implement the function in a DSP block instead of regular logic, depending on device utilization. For some devices, the software maps directly to DSP block device primitives instead of instantiating an IP core in the **.vqm** file.

Figure 2. HDL Analyst View of LPM_MULT IP Core (Unsigned 8x8 Multiplier with Pipeline=2)



2.10.3.1.1. Resource Balancing

While mapping multipliers to DSP blocks, the Synplify software performs resource balancing for optimum performance.

Intel devices have a fixed number of DSP blocks, which includes a fixed number of embedded multipliers. If the design uses more multipliers than are available, the Synplify software automatically maps the extra multipliers to logic elements (LEs), or adaptive logic modules (ALMs).

If a design uses more multipliers than are available in the DSP blocks, the Synplify software maps the multipliers in the critical paths to DSP blocks. Next, any wide multipliers, which might or might not be in the critical paths, are mapped to DSP blocks. Smaller multipliers and multipliers that are not in the critical paths might then be implemented in the logic (LEs or ALMs). This ensures that the design fits successfully in the device.

2.10.3.1.2. Controlling the DSP Block Inference

You can implement multipliers in DSP blocks or in logic in Intel devices that contain DSP blocks. You can control this implementation through attribute settings in the Synplify software.

2.10.3.1.3. Signal Level Attribute

You can control the implementation of individual multipliers by using the `syn_multstyle` attribute as shown in the following Verilog HDL code (where *<signal_name>* is the name of the signal):

```
<signal_name> /* synthesis syn_multstyle = "logic" */;
```

The `syn_multstyle` attribute applies to wires only; it cannot be applied to registers.

Table 7. DSP Block Attribute Setting in the Synplify Software

Attribute Name	Value	Description
syn_multstyle	lpm_mult	LPM function inferred and multipliers implemented in DSP blocks.
	logic	LPM function not inferred and multipliers implemented as LEs by the Synplify software.
	block_mult	DSP IP core is inferred and multipliers are mapped directly to DSP block device primitives (for supported devices).

Example 24. Signal Attributes for Controlling DSP Block Inference in Verilog HDL Code

```
module mult(a,b,c,r,en);
    input [7:0] a,b;
    output [15:0] r;
    input [15:0] c;
    input en;
    wire [15:0] temp /* synthesis syn_multstyle="logic" */;

    assign temp = a*b;
    assign r = en ? temp : c;
endmodule
```

Example 25. Signal Attributes for Controlling DSP Block Inference in VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity onereg is port (
    r : out std_logic_vector (15 downto 0);
    en : in std_logic;
    a : in std_logic_vector (7 downto 0);
    b : in std_logic_vector (7 downto 0);
    c : in std_logic_vector (15 downto 0);
    );
end onereg;

architecture beh of onereg is
signal temp : std_logic_vector (15 downto 0);
attribute syn_multstyle : string;
attribute syn_multstyle of temp : signal is "logic";

begin
    temp <= a * b;
    r <= temp when en='1' else c;
end beh;
```

2.10.3.2. Inferring RAM

When a RAM block is inferred from an HDL design, the Synplify software uses an Intel FPGA IP core to target the device memory architecture. For some devices, the Synplify software maps directly to memory block device primitives instead of instantiating an IP core in the `.vqm` file.

Follow these guidelines for the Synplify software to successfully infer RAM in a design:

- The address line must be at least two bits wide.
- Resets on the memory are not supported. Refer to the device family documentation for information about whether read and write ports must be synchronous.
- Some Verilog HDL statements with blocking assignments might not be mapped to RAM blocks, so avoid blocking statements when modeling RAMs in Verilog HDL.

For some device families, the `syn_ramstyle` attribute specifies the implementation to use for an inferred RAM. You can apply the `syn_ramstyle` attribute globally to a module or a RAM instance, to specify `registers` or `block_ram` values. To turn off RAM inference, set the attribute value to `registers`.

When inferring RAM for some Intel device families, the Synplify software generates additional bypass logic. This logic is generated to resolve a half-cycle read/write behavior difference between the RTL and post-synthesis simulations. The RTL simulation shows the memory being updated on the positive edge of the clock; the post-synthesis simulation shows the memory being updated on the negative edge of the clock. To eliminate bypass logic, the output of the RAM must be registered. By adding this register, the output of the RAM is seen after a full clock cycle, by which time the update has occurred, thus eliminating the need for bypass logic.

For devices with TriMatrix memory blocks, disable the creation of glue logic by setting the `syn_ramstyle` value to `no_rw_check`. Set `syn_ramstyle` to `no_rw_check` to disable the creation of glue logic in dual-port mode.

Example 26. VHDL Code for Inferred Dual-Port RAM

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY dualport_ram IS
PORT ( data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
    data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0)
    wr_addr, rd_addr: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    we: IN STD_LOGIC);
    clk: IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECOR (7 DOWNTO 0);
SIGNAL mem; Mem_Type;
SIGNAL addr_reg: STD_LOGIC_VECTOR (6 DOWNTO 0);

BEGIN
    data_out <= mem (CONV_INTEGER(rd_addr));
    PROCESS (clk, we, data_in) BEGIN
        IF (clk='1' AND clk'EVENT) THEN
            IF (we='1') THEN
                mem(CONV_INTEGER(wr_addr)) <= data_in;
            END IF;
        END IF;
    END PROCESS;
END ram_infer;
```

Example 27. VHDL Code for Inferred Dual-Port RAM Preventing Bypass Logic

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;

ENTITY dualport_ram IS
PORT ( data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
```

```
    data_in : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
    wr_addr, rd_addr : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
    we : IN STD_LOGIC;
    clk : IN STD_LOGIC);
END dualport_ram;

ARCHITECTURE ram_infer OF dualport_ram IS
TYPE Mem_Type IS ARRAY (127 DOWNTO 0) OF STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL mem : Mem_Type;
SIGNAL addr_reg : STD_LOGIC_VECTOR (6 DOWNTO 0);
SIGNAL tmp_out : STD_LOGIC_VECTOR (7 DOWNTO 0); --output register

BEGIN
    tmp_out <= mem (CONV_INTEGER (rd_addr));
    PROCESS (clk, we, data_in) BEGIN
        IF (clk='1' AND clk'EVENT) THEN
            IF (we='1') THEN
                mem(CONV_INTEGER(wr_addr)) <= data_in;
            END IF;
            data_out <= tmp_out; --registers output preventing
                                 -- bypass logic generation
        END IF;
    END PROCESS;
END ram_infer;
```

2.10.3.3. RAM Initialization

Use the Verilog HDL `$readmemb` or `$readmemh` system tasks in your HDL code to initialize RAM memories. The Synplify compiler forward-annotates the initialization values in the `.srs` (technology-independent RTL netlist) file and the mapper generates the corresponding hexadecimal memory initialization (`.hex`) file. One `.hex` file is created for each of the `altsyncram` IP cores that are inferred in the design. The `.hex` file is associated with the `altsyncram` instance in the `.vqm` file using the `init_file` attribute.

The examples show how RAM can be initialized through HDL code, and how the corresponding `.hex` file is generated using Verilog HDL.

Example 28. Using $readmemb System Task to Initialize an Inferred RAM in Verilog HDL Code

```
initial
begin
    $readmemb("mem.ini", mem);
end
always @(posedge clk)
begin
    raddr_reg <= raddr;
    if(we)
        mem[waddr] <= data;
end
```

Example 29. Sample of .vqm Instance Containing Memory Initialization File

```
altsyncram mem_hex( .wren_a(we),.wren_b(GND),...);

defparam mem_hex.lpm_type = "altsyncram";
defparam mem_hex.operation_mode = "Dual_Port";
...
defparam mem_hex.init_file = "mem_hex.hex";
```

2.10.3.4. Inferring ROM

When a ROM block is inferred from an HDL design, the Synplify software uses an Intel FPGA IP core to target the device memory architecture. For some devices, the Synplify software maps directly to memory block device atoms instead of instantiating an IP core in the `.vqm` file.

Follow these guidelines for the Synplify software to successfully infer ROM in a design:

- The address line must be at least two bits wide.
- The ROM must be at least half full.
- A CASE or IF statement must make 16 or more assignments using constant values of the same width.

2.10.3.5. Inferring Shift Registers

The Synplify software infers shift registers for sequential shift components so that they can be placed in dedicated memory blocks in supported device architectures using the ALTSHIFT_TAPS IP core.

If necessary, set the implementation style with the `syn_srlstyle` attribute. If you do not want the components automatically mapped to shift registers, set the value to `registers`. You can set the value globally, or on individual modules or registers.

For some designs, turning off shift register inference improves the design performance.

2.11. Synopsys Synplify* Support Revision History

Document Version	Intel Quartus Prime Version	Changes
2022.03.28	18.1	Revised linking to documentation archives.
2018.09.24	18.1	Removed reference to obsolete .edf file from "Design Flow" diagram.
2018.05.07	18.0	Corrected trademark symbols on tool names.
2016.10.31	16.1	• Implemented Intel rebranding.

2.12. Intel Quartus Prime Pro Edition User Guide: Third-Party Synthesis Archives

For the latest and previous versions of this user guide, refer to Intel Quartus Prime Pro Edition User Guide: Third-party Synthesis. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Intel Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Intel Quartus Prime Pro Edition FPGA design flow.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Intel Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Intel Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Intel Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Intel Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Intel Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Intel® Quartus® Prime Pro Edition User Guide

Third-party Logic Equivalence Checking Tools

Updated for Intel® Quartus® Prime Design Suite: **18.0**

This document is part of a collection - Intel® Quartus® Prime Pro Edition User Guides - Combined PDF link

Online Version
Send Feedback

683881
UG-20189
2019.08.30

Contents

1. OneSpin 360 EC-FPGA* Software Support

You can optionally use the third-party OneSpin 360 EC-FPGA* sequential equivalence checking tool to verify logic equivalence between specific netlists. For more information about 360 EC-FPGA, contact OneSpin.

2. OneSpin 360 EC-FPGA Software Support Revision History

This document has the following revision history.

Document Version	Intel® Quartus® Prime Version	Changes
2019.08.30	18.0.0	Replaced all content with reference to OneSpin documentation.
2019.04.26	18.0.0	• Updated the title of this guide for latest title naming convention. • Removed `set_global_assignment -name EDA_GENERATE_FUNCTIONAL_NETLIST ON -section_id eda_simulation` assignment from *Verifying Post-Route Retiming with OneSpin 360 EC-FPGA Software* topic since the assignment is obsolete.
2018.06.28	18.0.0	Initial release of document.

A. Intel Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Intel Quartus Prime Pro Edition FPGA design flow.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Intel Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Intel Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Mentor Graphics*, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Intel Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Mentor Graphics*, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, Transceiver Toolkit, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Intel Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Mentor Graphics* and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Intel Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Intel® Quartus® Prime Pro Edition User Guide

Debug Tools

Updated for Intel® Quartus® Prime Design Suite: **23.4**

This document is part of a collection - Intel® Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

1. System Debugging Tools Overview

This chapter provides a quick overview of the tools available in the Intel® Quartus® Prime system debugging suite and discusses the criteria for selecting the best tool for your debug requirements.

1.1. System Debugging Tools Portfolio

The Intel Quartus Prime software provides a portfolio of system debugging tools for real-time verification of your design.

System debugging tools provide visibility by routing (or "tapping") signals in your design to debugging logic. The Compiler includes the debugging logic in your design and generates programming files that you download into the FPGA or CPLD for analysis.

Each tool in the system debugging portfolio uses a combination of available memory, logic, and routing resources to assist in the debugging process. Because different designs have different constraints and requirements, you can choose the tool that matches the specific requirements for your design, such as the number of spare pins available or the amount of logic or memory resources remaining in the physical device.

1.1.1. System Debugging Tools Comparison

Table 1. System Debugging Tools Portfolio

Tool	Description	Typical Usage
System Console and Debugging Toolkits	• Provides real-time in-system debugging capabilities using available debugging toolkits. • Allows you to read from and write to memory mapped components in a system without a processor or additional software. • Communicates with hardware modules in a design through a Tcl interpreter. • Allows you to take advantage of all the features of the Tcl scripting language. • Supports JTAG and TCP/IP connectivity.	• Perform system-level debugging. • Debug or optimize signal integrity of a board layout even before finishing the design. • Debug external memory interfaces. • Debug an Ethernet Intel FPGA IP interface in real time. • Debug a PCI Express* link at the Physical, Data Link, and Transaction layers. • Debug and optimize high-speed serial links in your board design.
Signal Tap logic analyzer	• Uses FPGA resources. • Captures data continuously from the signals you specify while the logic analyzer is running. To capture and store only specific signal data, you specify conditions that *trigger* the start or stop of data capture. A trigger activates when the trigger conditions are met, stopping analysis and displaying the data	You have spare on-chip memory and you want functional verification of a design running in hardware.

continued...

Tool	Description	Typical Usage
Signal Probe	Incrementally routes internal signals to I/O pins while preserving results from the last place-and-routed design.	You have spare I/O pins and you want to check the operation of a small set of control pins using either an external logic analyzer or an oscilloscope.
Logic Analyzer Interface (LAI)	• Multiplexes a larger set of signals to a smaller number of spare I/O pins. • Allows you to select which signals switch onto the I/O pins over a JTAG connection.	You have limited on-chip memory and a large set of internal data buses to verify using an external logic analyzer. Logic analyzer vendors, such as Tektronics* and Agilent*, provide integration with the tool to improve usability.
In-System Sources and Probes	Provides an easy way to drive and sample logic values to and from internal nodes using the JTAG interface. Provides real-time slow sampling capability.	You want to prototype the FPGA design using a front panel with virtual buttons.
In-System Memory Content Editor	Displays and allows you to edit on-chip memory.	You want to view and edit the contents of on-chip memory that is not connected to a Nios® II processor. You can also use the tool when you do not want to have a Nios II debug core in your system.
Virtual JTAG Interface	Allows you to communicate with the JTAG interface so that you can develop custom applications.	You want to communicate with custom signals in your design.

Refer to the following for more information about launching and using the available debugging toolkits:

- Launching a Toolkit in System Console on page 174
- Available System Debugging Toolkits on page 175

1.1.2. Suggested Tools for Common Debugging Requirements

Table 2. Tools for Common Debugging Requirements(1)

Requirement	Signal Probe	Logic Analyzer Interface (LAI)	Signal Tap Logic Analyzer	Description
More Data Storage	N/A	X	—	An external logic analyzer with the LAI tool allows you to store more captured data than the Signal Tap logic analyzer, because the external logic analyzer can provide access to a bigger buffer. The Signal Probe tool does not capture or store data.
Faster Debugging	X	X	—	You can use the LAI or the Signal Probe tool with external equipment, such as oscilloscopes and mixed signal oscilloscopes (MSOs). This ability provides access to timing mode, which allows you to debug combined streams of data.
Minimal Effect on Logic Design	X	X(2)	X(2)	The Signal Probe tool incrementally routes nodes to pins, with no effect on the design logic. The LAI adds minimal logic to a design, requiring fewer device resources. The Signal Tap logic analyzer has little effect on the design, because the Compiler considers the debug logic as a separate design partition.
Short Compile and Recompile Time	X	X(2)	X(2)	Signal Probe uses incremental routing to attach signals to previously reserved pins. This feature allows you to quickly recompile when you change the selection of source signals.

continued...

Requirement	Signal Probe	Logic Analyzer Interface (LAI)	Signal Tap Logic Analyzer	Description
				The Signal Tap logic analyzer and the LAI can refit their own design partitions to decrease recompilation time.
Sophisticated Triggering Capability	N/A	N/A	X	The triggering capabilities of the Signal Tap logic analyzer are comparable to commercial logic analyzers.
Low I/O Usage	—	—	X	The Signal Tap logic analyzer does not require additional output pins. Both the LAI and Signal Probe require I/O pin assignments.
Fast Data Acquisition	N/A	—	X	The Signal Tap logic analyzer can acquire data at speeds of over 200 MHz. Signal integrity issues limit acquisition speed for external logic analyzers that use the LAI.
No JTAG Connection Required	X	—	—	Signal Probe does not require a host for debugging purposes. The Signal Tap logic analyzer and the LAI require an active JTAG connection to a host running the Intel Quartus Prime software.
No External Equipment Required	—	—	X	The Signal Tap logic analyzer only requires a JTAG connection from a host running the Intel Quartus Prime software or the stand-alone Signal Tap logic analyzer. Signal Probe and the LAI require the use of external debugging equipment, such as multimeters, oscilloscopes, or logic analyzers.

Notes to Table:
1. • X indicates the recommended tools for the feature.
   • — indicates that while the tool is available for that feature, that tool might not give the best results.
   • N/A indicates that the feature is not applicable for the selected tool.

1.1.3. Debugging Ecosystem

The Intel Quartus Prime software allows you to use the debugging tools in tandem to exercise and analyze the logic under test and maximize closure.

A very important distinction in the system debugging tools is how they interact with the design. All debugging tools in the Intel Quartus Prime software allow you to read the information from the design node, but only a subset allow you to input data at runtime:

Table 3. Classification of System Debugging Tools

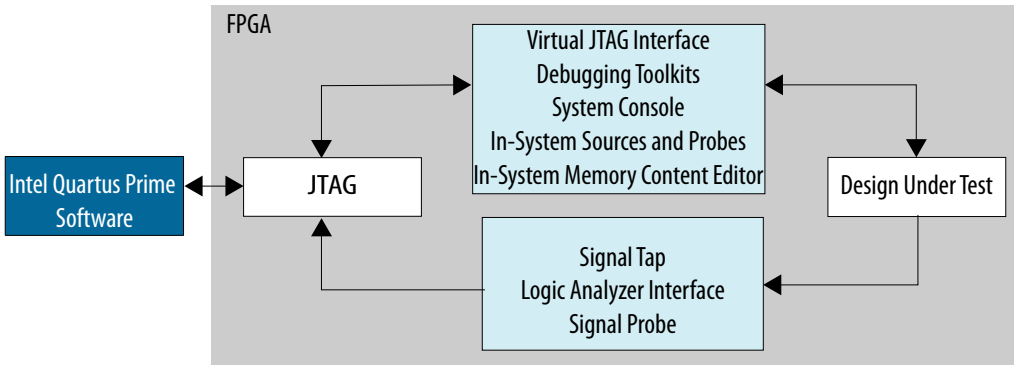
Debugging Tool	Read Data from Design	Input Values into the Design	Comments
Signal Tap logic analyzer,	Yes	No	General purpose troubleshooting tools optimized for probing signals in a register transfer level (RTL) netlist
Logic Analyzer Interface			
Signal Probe			
In-System Sources and Probes	Yes	Yes	These tools allow to:
Virtual JTAG Interface			

continued...

Debugging Tool	Read Data from Design	Input Values into the Design	Comments
System Console			• Read data from breakpoints that you define • Input values into your design during runtime
Debugging Toolkits			
In-System Memory Content Editor			

Taken together, the set of on-chip debugging tools form a debugging ecosystem. The set of tools can generate a stimulus to and solicit a response from the logic under test, providing a complete solution.

Figure 1. Debugging Ecosystem at Runtime



1.2. Tools for Monitoring RTL Nodes

The Signal Tap logic analyzer, Signal Probe, and LAI tools are useful for probing and debugging RTL signals at system speed. These general-purpose analysis tools enable you to tap and analyze any routable node from the FPGA.

- In cases when the design has spare logic and memory resources, the Signal Tap logic analyzer can provide fast functional verification of the design running on actual hardware.
- Conversely, if logic and memory resources are tight and you require the large sample depths associated with external logic analyzers, both the LAI and the Signal Probe tools simplify monitoring internal design signals using external equipment.

Related Information

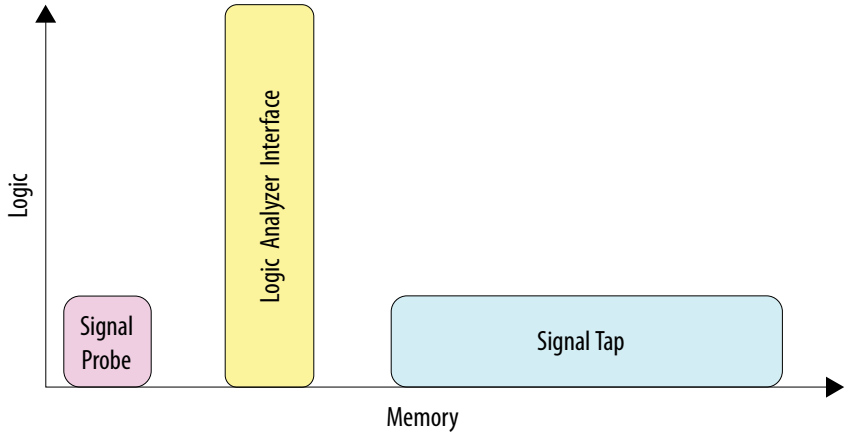
- Quick Design Verification with Signal Probe on page 124
- Design Debugging with the Signal Tap Logic Analyzer on page 29
- In-System Debugging Using External Logic Analyzers on page 129

1.2.1. Resource Usage

The most important selection criteria for these three tools are the remaining resources on the device after implementing the design and the number of spare pins.

Evaluate debugging options early on in the design planning process to ensure that you support the appropriate options in the board, Intel Quartus Prime project, and design. Planning early can reduce debugging time, and eliminates last minute changes to accommodate debug methodologies.

Figure 2. Resource Usage per Debugging Tool



1.2.1.1. Overhead Logic

Any debugging tool that requires a JTAG connection requires SLD infrastructure logic for communication with the JTAG interface and arbitration between instantiated debugging modules. This overhead logic uses around 200 logic elements (LEs), a small fraction of the resources available in any of the supported devices. All available debugging modules in your design share the overhead logic. Both the Signal Tap logic analyzer and the LAI use a JTAG connection.

1.2.1.1.1. For Signal Tap Logic Analyzer

The Signal Tap logic analyzer requires both logic and memory resources. The number of logic resources used depends on the number of signals tapped and the complexity of the trigger logic. However, the amount of logic resources that the Signal Tap logic analyzer uses is typically a small percentage of most designs.

A baseline configuration consisting of the SLD arbitration logic and a single node with basic triggering logic contains approximately 300 to 400 Logic Elements (LEs). Each additional node you add to the baseline configuration adds about 11 LEs. Compared with logic resources, memory resources are a more important factor to consider for your design. Memory usage can be significant and depends on how you configure your Signal Tap logic analyzer instance to capture data and the sample depth that your design requires for debugging. For the Signal Tap logic analyzer, there is the added benefit of requiring no external equipment, as all of the triggering logic and storage is on the chip.

1.2.1.1.2. For Signal Probe

The resource usage of Signal Probe is minimal. Because Signal Probe does not require a JTAG connection, logic and memory resources are not necessary. Signal Probe only requires resources to route internal signals to a debugging test point.

1.2.1.1.3. For Logic Analyzer Interface

The LAI requires a small amount of logic to implement the multiplexing function between the signals under test, in addition to the SLD infrastructure logic. Because no data samples are stored on the chip, the LAI uses no memory resources.

1.2.2. Pin Usage

1.2.2.1. For Signal Tap Logic Analyzer

Other than the JTAG test pins, the Signal Tap logic analyzer uses no additional pins. All data is buffered using on-chip memory and communicated to the Signal Tap logic analyzer GUI via the JTAG test port.

1.2.2.2. For Signal Probe

The ratio of the number of pins used to the number of signals tapped for the Signal Probe feature is one-to-one. Because this feature can consume free pins quickly, a typical application for this feature is routing control signals to spare pins for debugging.

1.2.2.3. For Logic Analyzer Interface

The LAI can map up to 256 signals to each debugging pin, depending on available routing resources. The JTAG port controls the active signals mapped to the spare I/O pins. With these characteristics, the LAI is ideal for routing data buses to a set of test pins for analysis.

1.2.3. Usability Enhancements

The Signal Tap logic analyzer, Signal Probe, and LAI tools can be added to your existing design with minimal effects. With the node finder, you can find signals to route to a debugging module without making any changes to your HDL files. Signal Probe inserts signals directly from your post-fit database. The Signal Tap logic analyzer and LAI support inserting signals from both pre-synthesis and post-fit netlists.

1.2.3.1. Incremental Routing

Signal Probe uses the incremental routing feature. The incremental routing feature runs only the Fitter stage of the compilation. This leaves your compiled design untouched, except for the newly routed node or nodes. With Signal Probe, you can save as much as 90% compile time of a full compilation.

1.2.3.2. Automation Via Scripting

As another productivity enhancement, all tools in the on-chip debugging tool set support scripting via the `quartus_stp` Tcl package. For the Signal Tap logic analyzer and the LAI, scripting enables user-defined automation for data collection while debugging in the lab. The System Console includes a full Tcl interpreter for scripting.

1.3. Stimulus-Capable Tools

The In-System Memory Content Editor, In-System Sources and Probes, and Virtual JTAG interface enable you to use the JTAG interface as a general-purpose communication port.

Though you can use all three tools to achieve the same results, there are some considerations that make one tool easier to use in certain applications:

- The In-System Sources and Probes is ideal for toggling control signals.
- The In-System Memory Content Editor is useful for inputting large sets of test data.
- Finally, the Virtual JTAG interface is well suited for advanced users who want to develop custom JTAG solutions.

System Console provides system-level debugging at a transaction level, such as with Avalon®-MM slave or Avalon-ST interfaces. You can communicate to a chip through JTAG and TCP/IP protocols. System Console uses a Tcl interpreter to communicate with hardware modules that you instantiate into your design.

1.3.1. In-System Sources and Probes

In-System Sources and Probes allow you to read and write to a design by accessing JTAG resources.

You instantiate an Intel FPGA IP into your HDL code. This Intel FPGA IP core contains source ports and probe ports that you connect to signals in your design, and abstracts the JTAG interface's transaction details.

In addition, In-System Sources and Probes provide a GUI that displays source and probe ports by instance, and allows you to read from probe ports and drive to source ports. These features make this tool ideal for toggling a set of control signals during the debugging process.

Related Information

Design Debugging Using In-System Sources and Probes on page 145

1.3.1.1. Push Button Functionality

During the development phase of a project, you can debug your design using the In-System Sources and Probes GUI instead of push buttons and LEDs. Furthermore, In-System Sources and Probes supports a set of scripting commands for reading and writing using the Signal Tap logic analyzer. You can also build your own Tk graphical interfaces using the Toolkit API. This feature is ideal for building a virtual front panel during the prototyping phase of the design.

Related Information

Signal Tap Scripting Support on page 118

1.3.2. In-System Memory Content Editor

The In-System Memory Content Editor allows you to quickly view and modify memory content either through a GUI interface or through Tcl scripting commands. The In-System Memory Content Editor works by turning single-port RAM blocks into dual-port RAM blocks. One port is connected to your clock domain and data signals, and the other port is connected to the JTAG clock and data signals for editing or viewing.

Related Information

In-System Modification of Memory and Constants on page 137

1.3.2.1. Generate Test Vectors

Because you can modify a large set of data easily, a useful application for the In-System Memory Content Editor is to generate test vectors for your design. For example, you can instantiate a free memory block, connect the output ports to the logic under test (using the same clock as your logic under test on the system side), and create the glue logic for the address generation and control of the memory. At runtime, you can modify the contents of the memory using either a script or the In-System Memory Content Editor GUI and perform a burst transaction of the data contents in the modified RAM block synchronous to the logic being tested.

1.3.3. System Console

System Console is a framework that you can launch from the Intel Quartus Prime software to start services for performing various debugging tasks. System Console provides you with Tcl scripts and a GUI to access the Platform Designer system integration tool to perform low-level hardware debugging of your design, as well as identify a module by its path, and open and close a connection to a Platform Designer module. You can access your design at a system level for purposes of loading, unloading, and transferring designs to multiple devices. Also, System Console supports the Tk toolkit for building graphical interfaces.

Related Information

Analyzing and Debugging Designs with System Console on page 159

1.3.3.1. Test Signal Integrity

System Console also allows you to access commands that allow you to control how you generate test patterns, as well as verify the accuracy of data generated by test patterns. You can use JTAG debug commands in System Console to verify the functionality and signal integrity of your JTAG chain. You can test clock and reset signals.

1.3.3.2. Board Bring-Up and Verification

You can use System Console to access programmable logic devices on your development board, perform board bring-up, and perform verification. You can also access software running on a Nios II or Intel FPGA SoC processor, as well as access modules that produce or consume a stream of bytes.

1.3.3.3. Debug with Available Toolkits

System Console provides the hardware debugging infrastructure to run the debugging toolkits that you can enable by the use of debug-enabled Intel FPGA IP. The debugging toolkits can help you to debug external memory interfaces, Ethernet interfaces, PCI Express links, Serial Lite IV links, and high-speed serial links by providing real-time monitoring and debugging of the design running on a board.

Refer to the following for more information about launching and using the available debugging toolkits:

- Launching a Toolkit in System Console on page 174
- Available System Debugging Toolkits on page 175

1.4. Virtual JTAG Interface Intel FPGA IP

The Virtual JTAG Interface Intel FPGA IP provides the finest level of granularity for manipulating the JTAG resource. This Intel FPGA IP allows you to build your own JTAG scan chain by exposing all of the JTAG control signals and configuring your JTAG Instruction Registers (IRs) and JTAG Data Registers (DRs). During runtime, you control the IR/DR chain through a Tcl API, or with System Console. This feature is meant for users who have a thorough understanding of the JTAG interface and want precise control over the number and type of resources used.

Related Information

- Virtual JTAG (altera_virtual_jtag) IP Core User Guide
- Virtual JTAG Interface (VJI) Intel FPGA IP
  In *Intel Quartus Prime Help*

1.5. System-Level Debug Fabric

During compilation, the Intel Quartus Prime generates the JTAG Hub to allow multiple instances of debugging tools in a design.

Most Intel FPGA on-chip debugging tools use the JTAG port to control and read-back data from debugging logic and signals under test. The JTAG Hub manages the sharing of JTAG resources.

Note: For System Console, you explicitly insert debug IP cores into the design to enable debugging.

The JTAG Hub appears in the project's design hierarchy as a partition named `auto_fab_<number>`.

1.6. SLD JTAG Bridge

The SLD JTAG Bridge extends the debug fabric across partitions, allowing a higher-level partition (static region or root partition) to access debug signals in a lower-level partition (partial reconfiguration region or core partition).

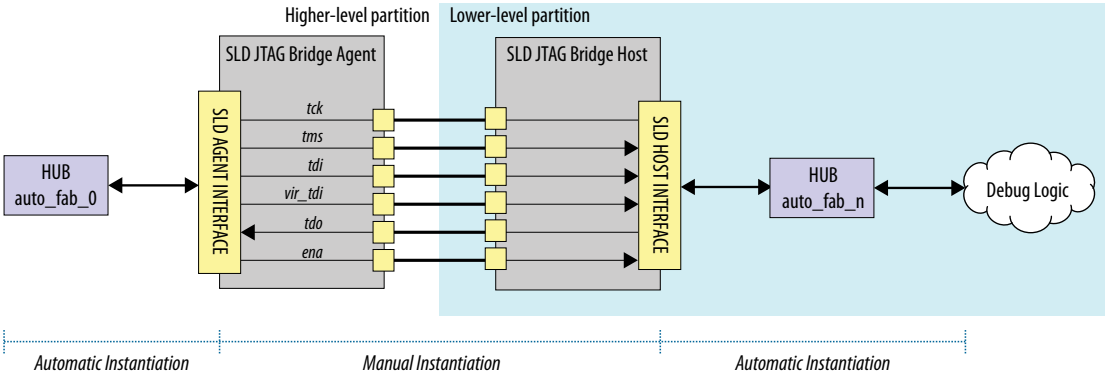
This bridge consists of two IP components:

- **SLD JTAG Bridge Agent Intel FPGA IP**—Resides in the higher-level partition.

  Extends the JTAG debug fabric from a higher-level partition to a lower-lever partition containing the SLD JTAG Bridge Host IP. You instantiate the SLD JTAG Bridge Agent IP in the higher-level partition.
- **SLD JTAG Bridge Host Intel FPGA IP**—resides in the lower-level partition. Connects to the PR JTAG hub on one end, and to the SLD JTAG Bridge Agent on the higher-level partition.

  Connects the JTAG debug fabric in a lower-level to a higher-level partition containing the SLD JTAG Bridge Agent IP. You instantiate the SLD JTAG Bridge Host IP in the lower-level partition.

Figure 3. Signals in a SLD JTAG Bridge



For each PR region or reserved core partition you debug, you must instantiate one SLD JTAG Bridge Agent in the higher-level partition and one SLD JTAG Bridge Host in the lower-level partition.

1.6.1. SLD JTAG Bridge Index

The SLD JTAG Bridge Index uniquely identifies instances of the SLD JTAG Bridge present in a design. You can find information regarding the Bridge Index in the synthesis report.

The Intel Quartus Prime software supports multiple instances of the SLD JTAG Bridge in designs. The Compiler assigns an index number to distinguish each instance. The bridge index for the root partition is always None.

When configuring the Signal Tap logic analyzer for the root partition, set the **Bridge Index** value to **None** in the JTAG Chain Configuration window.

Figure 4. JTAG Chain Configuration Bridge Index

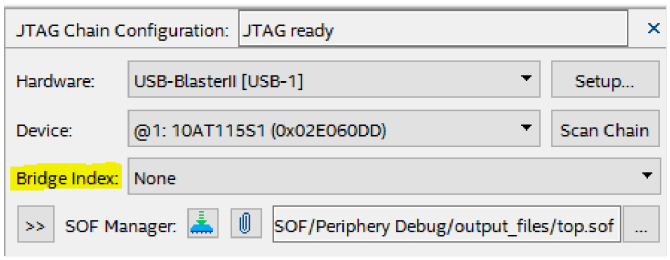
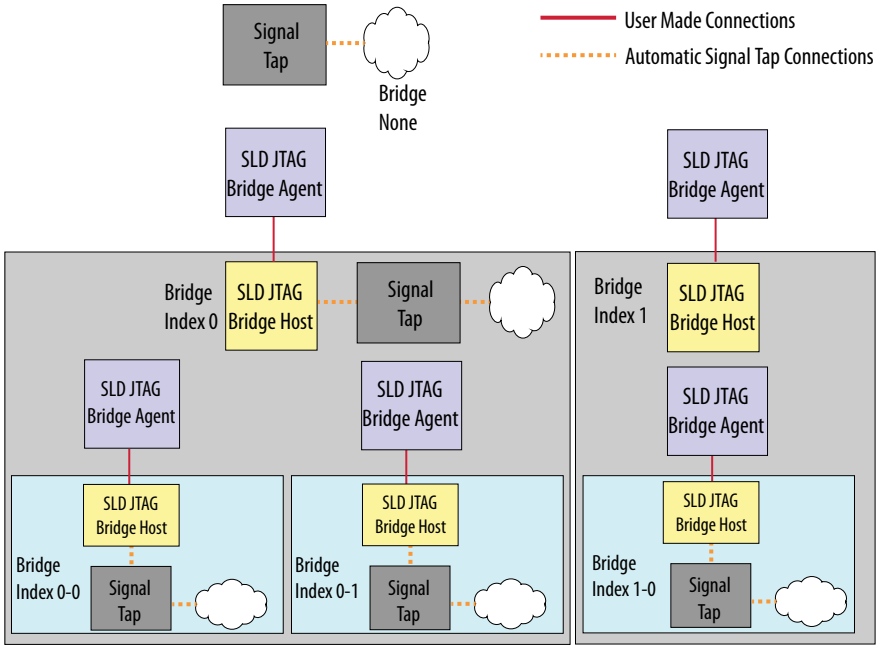


Figure 5. Design with Multiple SLD JTAG Bridges

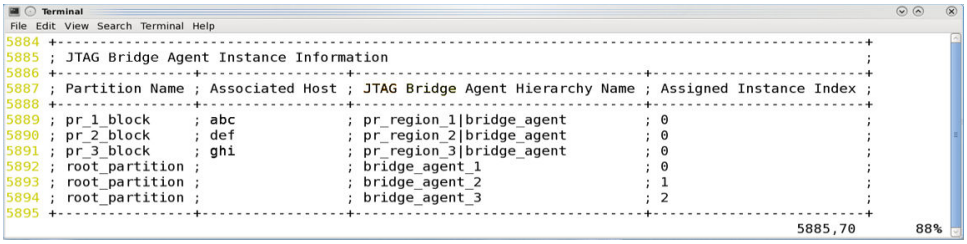


Bridge Index Information in the Compilation Report

Following design synthesis, the Compilation Report lists the index numbers for the SLD JTAG Bridge Agents in the design. Open the **Synthesis ➤ In-System Debugging ➤ JTAG Bridge Instance Agent Information** report for details about how the bridge indexes are enumerated. The reports shows the hierarchy path and the associated index.

In the synthesis report (`<base revision>.syn.rpt`), this information appears in the table **JTAG Bridge Agent Instance Information**.

Figure 6. JTAG Bridge Agent Instance Information

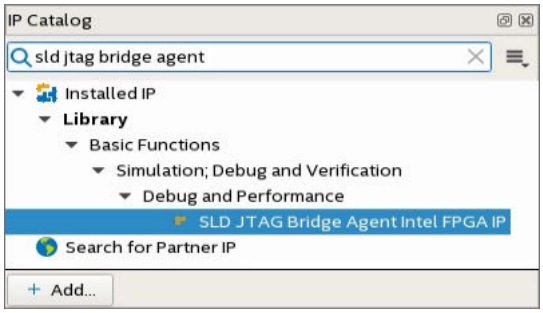


1.6.2. Instantiating the SLD JTAG Bridge Agent

To generate and instantiate the SLD JTAG Bridge Agent Intel FPGA IP:

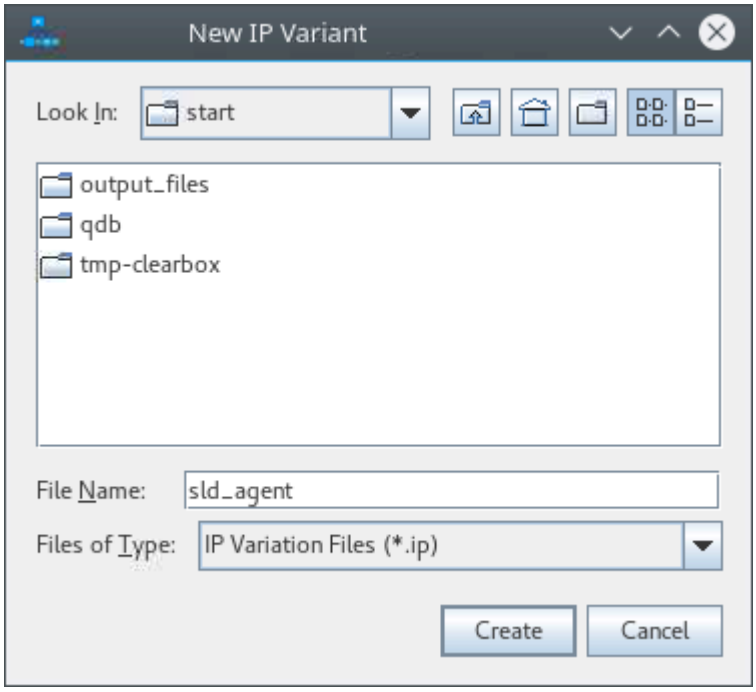
1. On the IP Catalog (**Tools ➤ IP Catalog**), type `SLD JTAG Bridge Agent`.

Figure 7. Find in IP Catalog



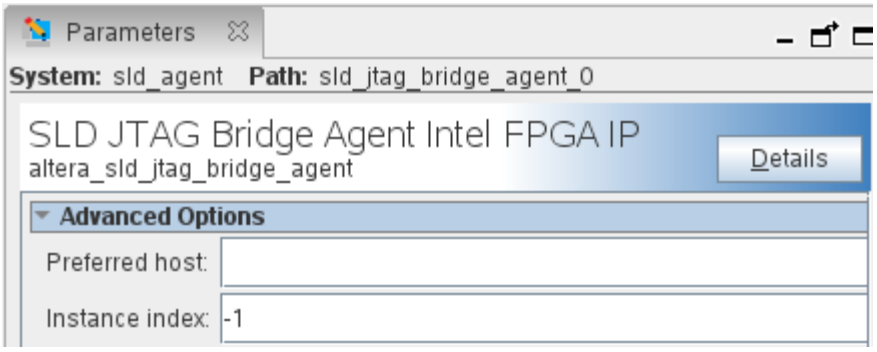
2. Double click **SLD JTAG Bridge Agent Intel FPGA IP**.
3. In the **Create IP Variant** dialog box, type a file name, and then click **Create**.

Figure 8. Create IP Variant Dialog Box



The **IP Parameter Editor Pro** window shows the IP parameters. In most cases, you do not need to change the default values.

Figure 9. SLD JTAG Bridge Agent Intel FPGA IP Parameters



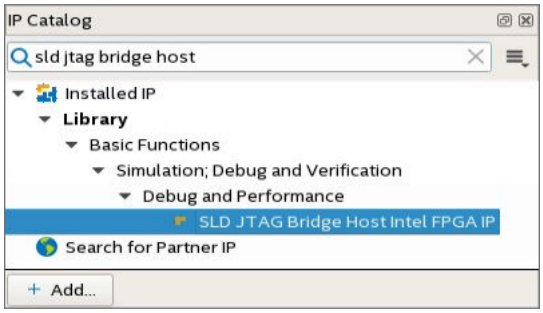
4. Click **Generate HDL**.
5. When the generation completes successfully, click **Close**.
6. If you want an instantiation template, click **Generate ➤ Show Instantiation Template** in the **IP Parameter Editor Pro**.

1.6.3. Instantiating the SLD JTAG Bridge Host

To generate and instantiate the SLD JTAG Bridge Host Intel FPGA IP:

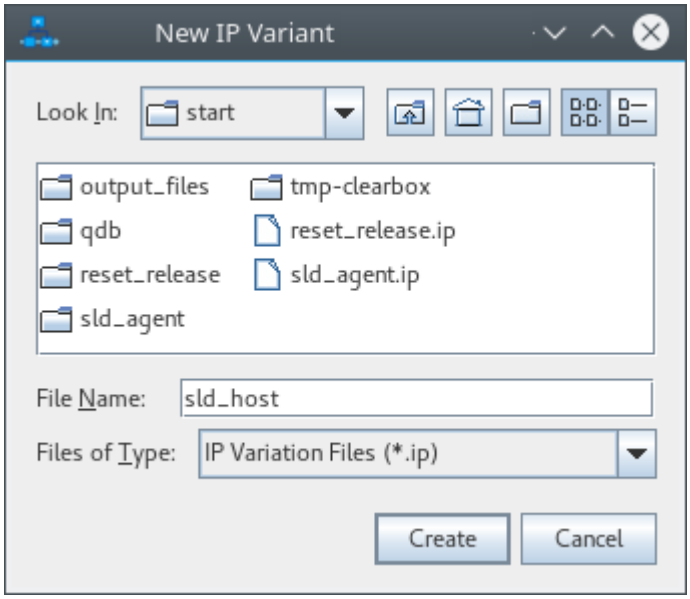
1. On the IP Catalog (**Tools ➤ IP Catalog**), type `SLD JTAG Bridge Host`.

Figure 10. Find in IP Catalog



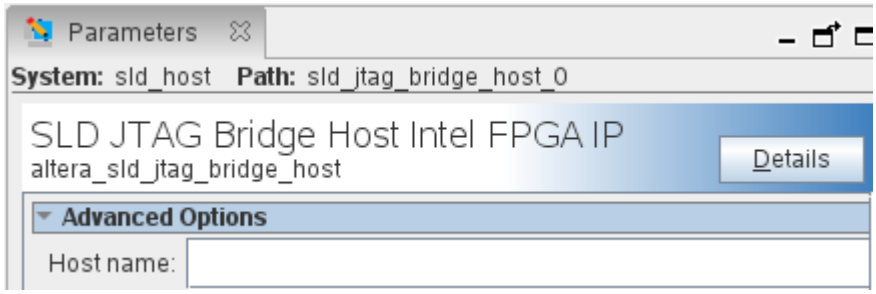
2. Double click **SLD JTAG Bridge Host Intel FPGA IP**.
3. In the **Create IP Variant** dialog box, type a file name, and then click **Create**.

Figure 11. Create IP Variant Dialog Box



The **IP Parameter Editor Pro** window shows the IP parameters. In most cases, you do not need to change the default values.

Figure 12. SLD JTAG Bridge Host Intel FPGA IP Parameters



4. Click **Generate HDL**.
5. When the generation completes successfully, click **Close**.
6. If you want an instantiation template, click **Generate ➤ Show Instantiation Template** in the **IP Parameter Editor Pro**.

1.7. Partial Reconfiguration Design Debugging

The following Intel FPGA IP cores support system-level debugging in the static region of a PR design:

- In-System Memory Content Editor
- In-System Sources and Probes Editor
- Virtual JTAG
- Nios II JTAG Debug Module
- Signal Tap Logic Analyzer

In addition, the Signal Tap logic analyzer allows you to debug the static or partial reconfiguration (PR) regions of the design. If you only want to debug the static region, you can use the In-System Sources and Probes Editor, In-System Memory Content Editor, or System Console with a JTAG Avalon bridge.

Related Information

Debugging Partial Reconfiguration Designs with Signal Tap on page 103

1.7.1. Debug Fabric for Partial Reconfiguration Designs

You must prepare the design for PR debug during the early planning stage, to ensure that you can debug the static as well as PR region.

On designs with Partial Reconfiguration, the Compiler generates centralized debug managers—or hubs—for each region (static and PR) that contains system level debug agents. Each hub handles the debug agents in its partition. In the design hierarchy, the hub corresponding to the static region is `auto_fab_0`.

To connect the hubs on parent and child partitions, you must instantiate one SLD JTAG Bridge for each PR region that you want to debug.

Related Information

- PR Design Setup for Signal Tap Debug on page 104
- Debugging Partial Reconfiguration Designs with Signal Tap on page 103

1.7.1.1. Generation of PR Debug Infrastructure

During compilation, the synthesis engine performs the following functions:

- Generates a main JTAG hub in the static region.
- If the static region contains Signal Tap instances, connects those instances to the main JTAG hub.
- Detects bridge agent and bridge host instances.
- Connects the SLD JTAG bridge agent instances to the main JTAG hub.
- For each bridge host instance in a PR region that contains a Signal Tap instance:
  - Generates a PR JTAG hub in the PR region.
  - Connects all Signal Tap instances in the PR region to the PR JTAG hub.
  - Detects instance of the SLD JTAG bridge host.
  - Connects the PR JTAG hub to the JTAG bridge host.

1.8. Preserving Signals for Debugging

The Intel Quartus Prime Pro Edition software allows you to mark and preserve specific signals through the compilation process, which enables visibility of any node within the available system debugging tools.

To ensure that specific nodes in your RTL are available for debugging after the Compiler's synthesis and place-and-route stages, you can apply the `preserve_for_debug` attribute to the signals of interest in your RTL, and also apply the **Enable preserve for debug assignments** project-level `.qsf` assignment.

This section refers to the following terms to explain use of the preserve for debug feature:

Table 4. Debug Signal Preservation Terminology

Term	Description
node	A signal name present in your design RTL and possibly in the compilation netlist for the current project. Typically, the node name refers to the output of a logical unit, such as gate, register, LUT, embedded memory, DSP, or others. The Intel Quartus Prime GUI can display this node name in various locations, such as the Node Finder, when debugging the signals in your design. You can search for this node name to apply constraints and use in debugging operations.
hpath	The Intel Quartus Prime-style hierarchical path, with instance names separated by "|", for example:`foo|boo|node`

1.8.1. Preserve for Debug Overview

The preserve for debug feature allows you to designate nodes in your design for full debugging visibility. In this context, full visibility means that you can ensure that the node name remains in the post-fit netlist generated by Place and Route, with the same name and functionality the design files define.

After you apply preserve for debug, you can easily access these nodes through the Node Finder filters available in the Intel Quartus Prime debugging tools.

Typically, you lose some visibility into the design when you debug using a post-fit netlist. This loss occurs because in the post-fit netlist, the design is already mapped to the device architecture, optimized, and retimed. The Place and Route stage often changes or removes the original signal names. Furthermore, there can be slight changes in the behavior in the post-fit netlist because of inverter push back, or because the visible signal shows only partial behavior due to logic duplication.

Preserve for Debug Use Cases

Preserve for debug is primarily for debugging purposes, and is particularly useful in the Signal Tap debugging flow, as Preserving Signals for Monitoring and Debugging on page 40 describes.

In addition, use of preserve for debug can also be helpful in any of the available system debugging tools, or within any instrumentation logic that you use in your design.

Preserve for Debug Hardware Implementation

Applying the preserve for debug feature has the following effects on hardware implementation:

- Prevents the Compiler from optimizing the specified node.
- Results in LCELL module instantiation for the specified node, impacting the overall timing on the node path.

Application of preserve for debug is the hardware equivalent of using all of the following HDL pragmas on the specified node:

Table 5. Combined Attributes

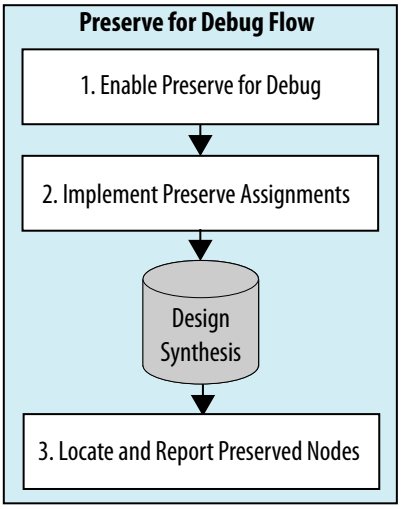
HDL Pragma	Compiler Setting	Description
`preserve`	PRESERVE_REGISTER	Prevents the Compiler from optimizing away or retiming a register.
`keep`	HDL only	Prevents the Compiler from minimizing or removing a particular signal net during combinational logic optimization.
`noprune`	HDL only	Prevents the Compiler from removing or optimizing a fan-out free register.
`dont_merge`	HDL only	Prevents the Compiler from merging a register with a duplicate register.
`dont_replicate`	HDL only	Prevents the Compiler from merging a register with a duplicate register.

1.8.2. Marking Signals for Debug

You can mark (designate) a node for preservation by use of an RTL pragma in your design file, or by specifying an assignment in the project revision `.qsf`.

You can enable or disable preserve for debug at the entity level or globally, so there is no need to individually disable marked signals when ready to compile a production stage design.

Figure 13. Preserve for Debug Flow



- Step 1: Enabling Preserve for Debug on page 23
- Step 2: Implement Preserve for Debug Assignments on page 24
- Step 3: Locate and Report Preserve for Debug Nodes on page 25

1.8.2.1. Step 1: Enabling Preserve for Debug

To ensure that the Compiler correctly processes the signals that you mark for preservation, and that the Intel Quartus Prime software Node Finders and filters correctly display these names, you must first turn on the **Enable preserve for debug assignments** setting in the GUI or project revision `.qsf`, as the following methods describe.

Note: The instance-level preserve for debug assignment takes precedence over the global preserve for debug assignment if the two assignments are in opposition to each other (that is, one assignment type is set to On, the other assignment type is set to Off).

Enabling Preserve for Debug In Project Settings on page 23

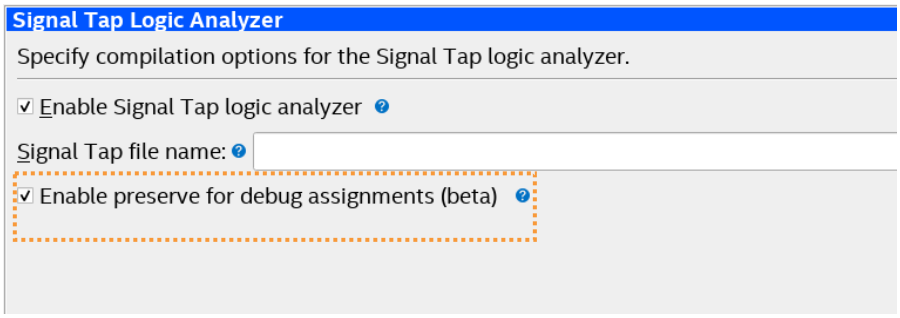
Enabling Preserve for Debug at Instance Level on page 24

1.8.2.1.1. Enabling Preserve for Debug In Project Settings

To enable preserve for debug in the project settings:

1. In the Intel Quartus Prime software, click **Assignments ➤ Settings ➤ Signal Tap Logic Analyzer**.
2. On the **Signal Tap Logic Analyzer** settings page, turn on the **Enable preserve for debug assignments** option. Preserve for debug enables project-wide.

Figure 14. Signal Tap Logic Analyzer Settings



As an alternative to the GUI setting, you can enable or disable project-wide preserve for debug by adding or modifying the following assignment in the project revision `.qsf`:

```
set_global_assignment -name PRESERVE_FOR_DEBUG_ENABLE <ON|OFF>
```

1.8.2.1.2. Enabling Preserve for Debug at Instance Level

You can enable preserve for debug in certain design blocks, and leave the feature disabled in other design blocks.

To enable the assignment at the instance level, you must specify the instance name to enable or disable for preserve for debug, as the following assignment shows:

```
set_instance_assignment -name PRESERVE_FOR_DEBUG_ENABLE ON -to \
     <instance hpath>
```

1.8.2.2. Step 2: Implement Preserve for Debug Assignments

Implement preserve for debug assignments through HDL pragmas in the design files (recommended), or by specifying assignments in the Assignment Editor or project `.qsf` file directly. The following topics provide more details:

HDL Implementation on page 24
Intel Quartus Prime Settings Implementation on page 25

1.8.2.2.1. HDL Implementation

The recommended method of preserving nodes for debug is to add HDL pragmas or attributes to the design files.

Preserve for Debug Pragma defines the preserve for debug pragma and `.qsf` assignment setting.

Table 6. Preserve for Debug Pragma

Term	Equivalent (.qsf) Setting	Description
`preserve_for_debug`	`PRESERVE_FOR_DEBUG`	Prevents the Fitter from optimizing away a register or combinational signal. The pragma also prevents any retiming, merging, and duplication optimization. This optimization prevention applies when the setting, `PRESERVE_FOR_DEBUG_ENABLE` is `ON`.

Add HDL pragmas to Verilog HDL design files in the following way:

```
(* preserve_for_debug *) reg my_reg;
```

Add HDL attributes to VHDL design files in the following way:

```
signal keep_wire : std_logic;
attribute keep: boolean;
attribute keep of keep_wire: signal is true;
```

1.8.2.2.2. Intel Quartus Prime Settings Implementation

As an alternative to HDL pragmas, you can specify the following assignment to apply the Preserve for Debug assignment through the `.qsf` settings file directly, or with Assignment Editor.

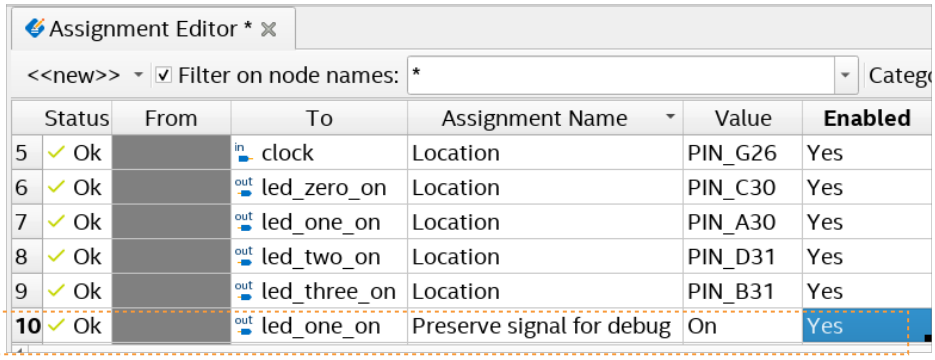
```
set_instance_assignment -name PRESERVE_FOR_DEBUG ON -to \
     <node hpath>
```

Note: This assignment supports the use of wildcards (*).

Specifying Preserve Signal for Debug in the Assignment Editor

If you prefer to specify assignments in the Intel Quartus Prime software GUI, rather than in the `.qsf` directly, you can specify the **Preserve signal for debug** assignment in Assignment Editor (Assignments menu).

Figure 15. Specifying the Preserve Signal for Debug in the Assignment Editor



1.8.2.3. Step 3: Locate and Report Preserve for Debug Nodes

After running design synthesis, you can locate preserve for debug nodes using the Node Finder in the system debugging tools. In addition, you can view data about the preserve for debug nodes in the Compilation Report. The following topics describe locating and reporting on preserve for debug nodes:

Locating Preserve for Debug Nodes on page 25

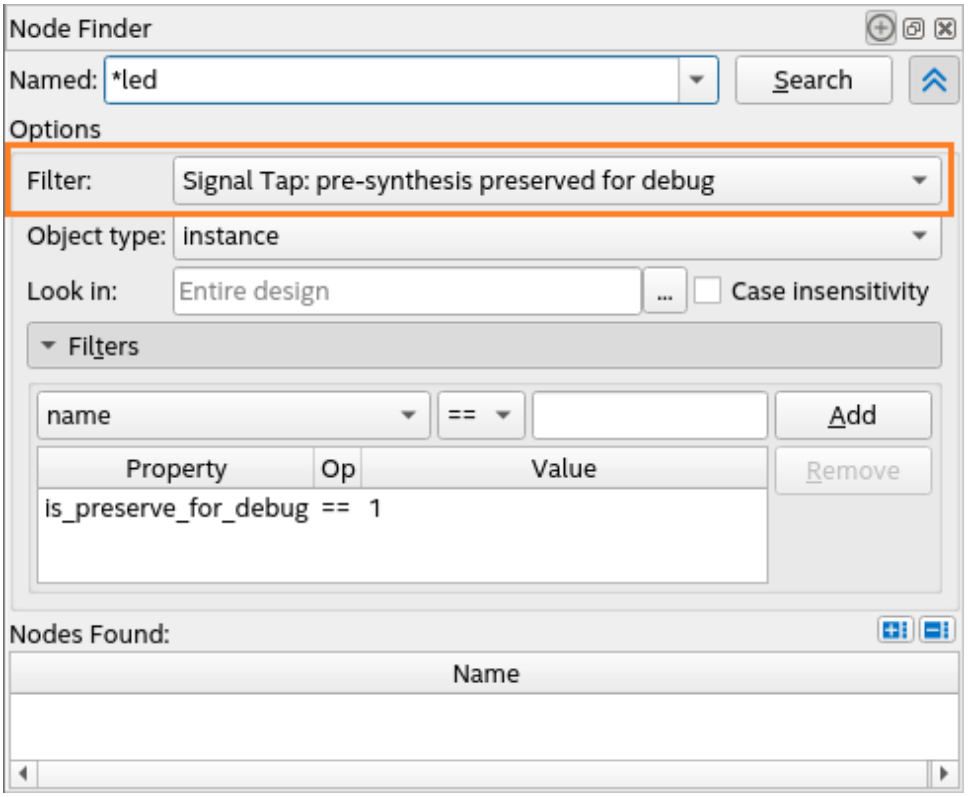
Reporting Preserve for Debug Nodes on page 26

1.8.2.3.1. Locating Preserve for Debug Nodes

The Node Finder includes the following filters that simplify the process of locating the preserve for debug nodes in your project database:

- **Signal Tap: pre-synthesis preserved for debug** filter—shows preserved nodes from the pre-synthesis netlist that generates during Analysis & Elaboration.
- **Signal Tap: post-fitting preserved for debug filter**—shows preserved nodes from the post-fit netlist.

Figure 16. Node Finder with Preserve for Debug Filter

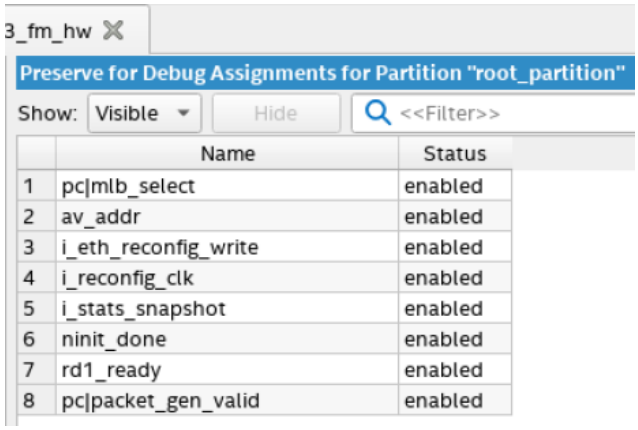


1.8.2.3.2. Reporting Preserve for Debug Nodes

You can view data about preserve for debug nodes in the Compilation Report **Preserve for Debug** folder following Analysis & Synthesis.

The Preserve for Debug Assignments for Partition report is located in **Tools ➤ Compilation Report ➤ Synthesis ➤ Partition <name> ➤ Preserve for Debug**.

Figure 18. Preserve for Debug Assignments for Partition Report



	Name	Status
1	pc|mlb_select	enabled
2	av_addr	enabled
3	i_eth_reconfig_write	enabled
4	i_reconfig_clk	enabled
5	i_stats_snapshot	enabled
6	ninit_done	enabled
7	rd1_ready	enabled
8	pc|packet_gen_valid	enabled

1.9. System Debugging Tools Overview Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2023.10.02	23.3	• Updated the Node Finder image in *Locating Preserve for Debug Nodes*.
2021.10.04	21.3	• Added new *Preserving Signals for Debugging* section. • Removed obsolete *Remote Debugging* topic. This feature is not supported in the Intel Quartus Prime Pro Edition software. • Removed obsolete *Remote Debugging* chapter 8. This feature is not supported in the Intel Quartus Prime Pro Edition software.
2020.09.28	20.3	• Revised "System Debugging Tools Comparison" to reflect replacement of Transceiver Toolkit with the available debugging toolkits. • Revised "Debugging Ecosystem" to reflect replacement of Transceiver Toolkit with the available debugging toolkits.
2019.09.30	19.3	• Clarified meaning of PR and static regions in "Partial Reconfiguration Design Debugging" topic. • Removed references to Application Notes 693.
2018.09.24	18.1	• Added figures about SLD JTAG Bridge. • Added information about block-based design.
2018.05.07	18.0	• Moved here information about debug fabric on PR designs from the *Design Debugging with the Signal Tap Logic Analyzer* chapter.
2017.05.08	17.0	• Combined Altera JTAG Interface and Required Arbitration Logic topics into a new updated topic named System-Level Debugging Infrastructure. • Added topic: Debug the Partial Reconfiguration Design with System Level Debugging Tools.
2016.10.31	16.1	• Implemented Intel rebranding.
2015.11.02	15.1	Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0	Added information that System Console supports the Tk toolkit.
November 2013	13.1	Dita conversion. Added link to Remote Debugging over TCP/IP for Altera SoC Application Note.
June 2012	12.0	Maintenance release.

continued...

Document Version	Intel Quartus Prime Version	Changes
November 2011	10.0	Maintenance release. Changed to new document template.
December 2010	10.0	Maintenance release. Changed to new document template.
July 2010	10.0	Initial release

2. Design Debugging with the Signal Tap Logic Analyzer

2.1. Signal Tap Logic Analyzer Introduction

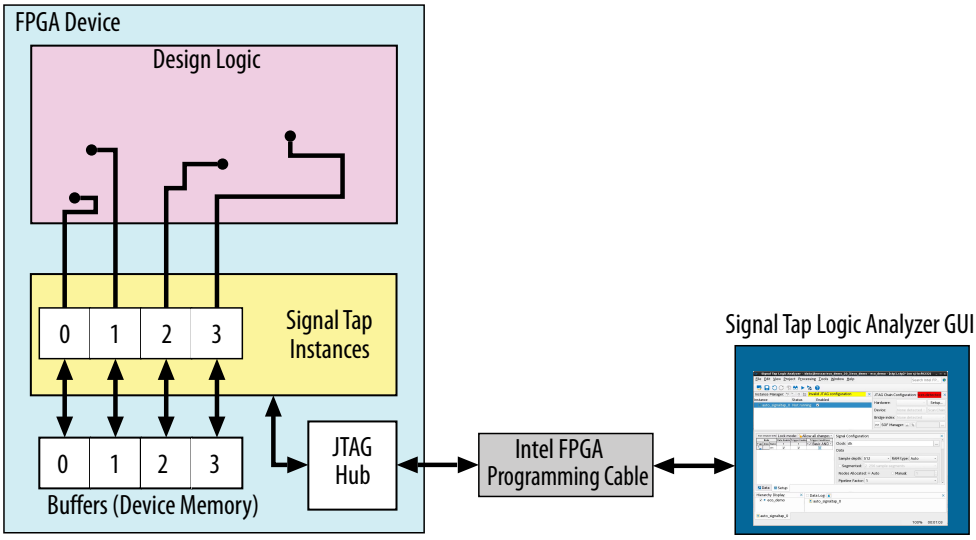
The Signal Tap logic analyzer, available in the Intel Quartus Prime software, captures and displays the real-time signal behavior in an Intel FPGA design. Use the Signal Tap logic analyzer to probe and debug the behavior of internal signals during normal device operation, without requiring extra I/O pins or external lab equipment.

By default, the Signal Tap logic analyzer captures data continuously from the signals you specify while the logic analyzer is running. To capture and store only specific signal data, you specify conditions that *trigger* the start or stop of data capture. A trigger activates when the trigger conditions are met, stopping analysis and displaying the data. You can save the captured data in device memory for later analysis, and filter data that is not relevant.

Signal Tap Logic Analyzer *Instance*

You enable the logic analyzer functionality by defining one or more instances of the Signal Tap logic analyzer in your project. You can define the properties of the Signal Tap instance in the Signal Tap logic analyzer GUI, or by HDL instantiation of the Signal Tap Logic Analyzer Intel FPGA IP. After design compilation, you configure the target device with your design (including any Signal Tap instances), which enables data capture and communication with the Signal Tap logic analyzer GUI over a JTAG connection.

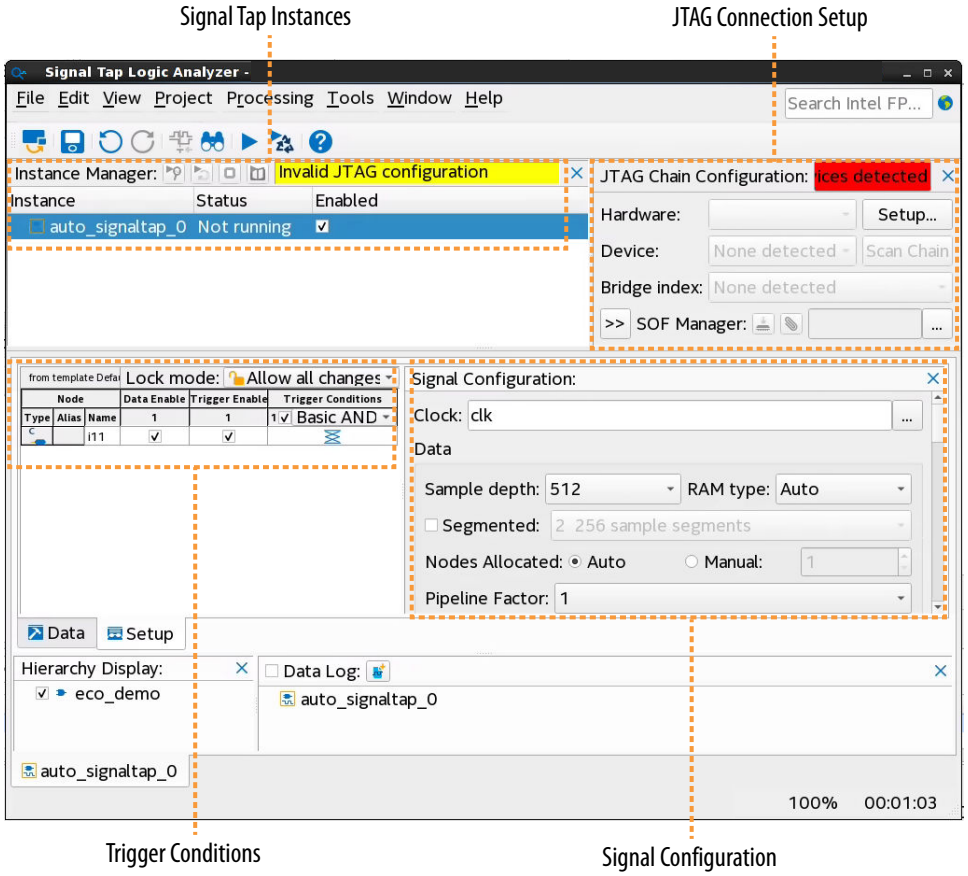
Figure 19. Signal Tap Logic Analyzer Block Diagram



Signal Tap Logic Analyzer *GUI*

The Signal Tap logic analyzer GUI helps you to rapidly define and modify Signal Tap signal configuration and JTAG connection settings, displays the captured signals during analysis, starts and stops analysis, and displays and records signal data. When you configure a Signal Tap instance in the GUI, Signal Tap preserves the instance settings in a Signal Tap Logic Analyzer file (`.stp`) for reuse.

Figure 20. Signal Tap Logic Analyzer GUI



Signal Tap Logic Analyzer and Simulator Integration

You can integrate the Signal Tap logic analyzer with your supported simulator environment. Signal Tap can readily generate a list of "simulator-aware" nodes to tap for any design hierarchy. Tapping this set of nodes then provides full visibility into the entire design hierarchy for direct observation of all internal signal states in your RTL simulator.

Signal Tap also supports automatic RTL simulation testbench creation, allowing you to export acquired Signal Tap hardware data directly into your RTL simulator and observe signals beyond those that you specify for tapping in Signal Tap. You can produce simulation events using the live data traffic to replicate in your simulator.

Signal Tap Logic Analyzer Capabilities

The Signal Tap logic analyzer supports a high number of channels, a large sample depth, fast clock speeds, and other features described in the *Key Signal Tap Logic Analyzer Capabilities* table.

Table 7. Key Signal Tap Logic Analyzer Capabilities

Capability	Benefit
Multiple logic analyzers in a single device, or in multiple devices in a single chain	Capture data from multiple clock domains and from multiple devices at the same time.
Up to 10 trigger conditions for each analyzer instance	Send complex data capture commands to the logic analyzer for greater accuracy and problem isolation.
Power-up trigger	Capture signal data for triggers that occur after device programming, but before manually starting the logic analyzer.
Custom trigger HDL object	Define a custom trigger in Verilog HDL or VHDL and tap specific instances of modules across the design hierarchy, without manual routing of all the necessary connections.
State-based triggering flow	Organize triggering conditions to precisely define data capture.
Flexible buffer acquisition modes	Precise control of data written into the acquisition buffer. Discard data samples that are not relevant to the debugging of your design.
MATLAB* integration with MEX function	Collect Signal Tap capture data into a MATLAB integer matrix.
RTL simulator integration	Easily create a set of nodes to tap for the design hierarchy, and observe all internal signal states in your RTL simulator. Automatic testbench creation allows you to inject acquired Signal Tap data directly into your RTL simulator.
Up to 4,096 channels per logic analyzer instance	Samples many signals and wide bus structures.
Up to 128K samples per instance	Captures a large sample set for each channel.
Fast clock frequencies	Synchronous sampling of data nodes using the same clock tree driving the logic under test.
Compatible with other debugging utilities	Use the Signal Tap logic analyzer in tandem with any JTAG-based on-chip debugging tool, such as an In-System Memory Content editor, to change signal values in real-time.
Floating-Point Display Format	• Single-precision floating-point format **IEEE754 Single (32-bit)**. • Double-precision floating-point format **IEEE754 Double (64-bit)**.

2.1.1. Signal Tap Hardware and Software Requirements

All editions of the Intel Quartus Prime design software include the Signal Tap logic analyzer GUI and Signal Tap Logic Analyzer Intel FPGA IP. The Signal Tap logic analyzer is also available as a stand-alone application.

During data acquisition, the memory blocks in the FPGA device store the captured data, and then transfer the data to the Signal Tap logic analyzer over a JTAG communication cable, such as Intel FPGA Ethernet Cable or Intel FPGA Download Cable.

The Signal Tap logic analyzer requires the following hardware and software to perform logic analysis:

- The Signal Tap logic analyzer included with the Intel Quartus Prime software, or the Signal Tap logic analyzer standalone software and standalone Programmer software.
- An Intel FPGA download or communications cable.
- An Intel development kit, or your own design board with a JTAG connection to the device under test.

Related Information

Running the Stand-Alone Version of Signal Tap on page 118

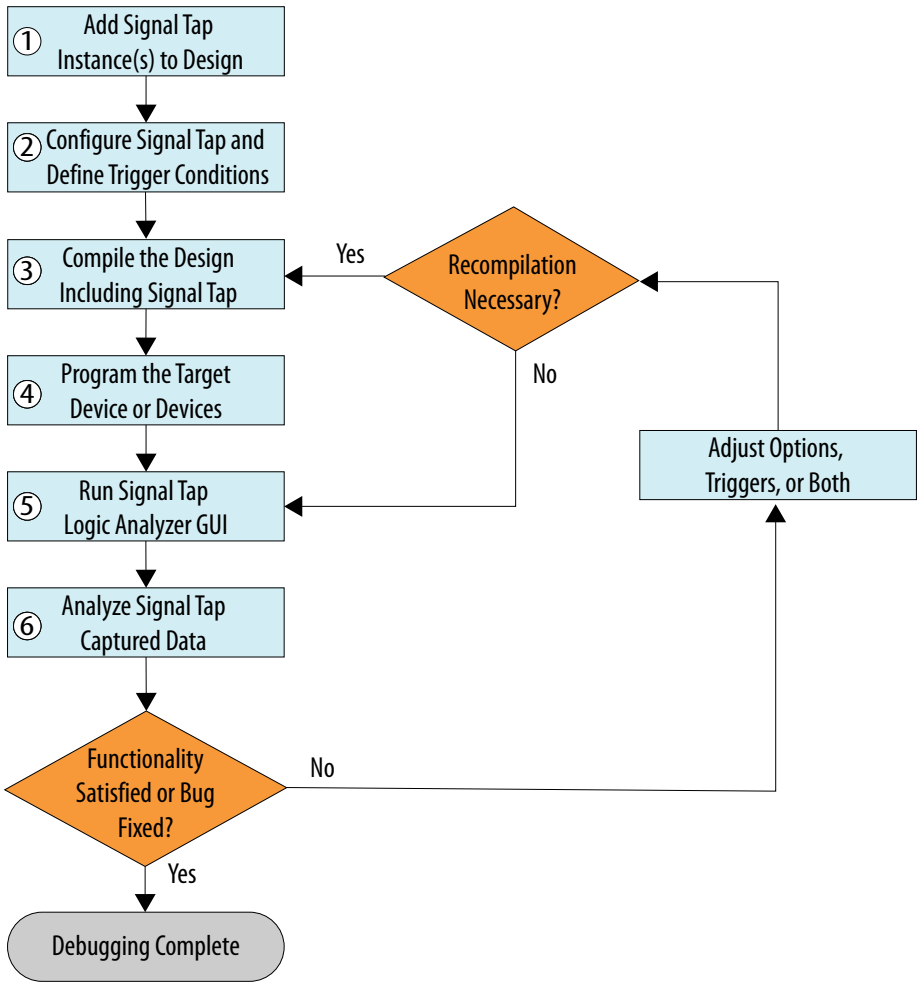
2.2. Signal Tap Debugging Flow

To use the Signal Tap logic analyzer to debug your design, you compile your design that includes one or more Signal Tap instances that you define, configure the target device, and then run the logic analyzer to capture and analyze signal data.

When Should I Add Signal Tap to the Design?

It is best to add the Signal Tap logic analyzer to your design early in the design flow to help prevent later difficulty in fitting the Signal Tap logic into the target device. If you add Signal Tap late in the design cycle, you may have difficulty with fitting if the device is already at 90-95% full. However, you can use the ECO compilation feature to add Signal Tap as soon as you initially create the design, even before adding nodes or running synthesis. This technique allows you to more easily make the ECO connections later if needed. Refer to *Using the ECO Compilation Flow* in *Intel Quartus Prime Pro Edition User Guide: Design Optimization*.

Figure 21. Signal Tap Debugging Flow



The following steps describe the Signal Tap debugging flow in detail:

- Step 1: Add the Signal Tap Logic Analyzer to the Project on page 34
- Step 2: Configure the Signal Tap Logic Analyzer on page 39
- Step 3: Compile the Design and Signal Tap Instances on page 82
- Step 4: Program the Target Hardware on page 85
- Step 5: Run the Signal Tap Logic Analyzer on page 86
- Step 6: Analyze Signal Tap Captured Data on page 92

Related Information

Using the ECO Compilation Flow in *Intel Quartus Prime Pro Edition User Guide: Design Optimization*

2.3. Step 1: Add the Signal Tap Logic Analyzer to the Project

To debug a design using the Signal Tap logic analyzer, you must first define one or more Signal Tap instances and add them to your project. You then compile the Signal Tap instances, along with your design. You can define a Signal Tap instance in the Signal Tap logic analyzer GUI or by HDL instantiation.

To help you get started quickly, the Signal Tap logic analyzer GUI includes preconfigured templates for various trigger conditions and applications. You can then modify the settings the template applies and adjust trigger conditions in the Signal Tap logic analyzer GUI.

Alternatively, you can define a Signal Tap instance by parameterizing an instance of the Signal Tap Logic Analyzer Intel FPGA IP, and then instantiating the Signal Tap entity or module in an HDL design file.

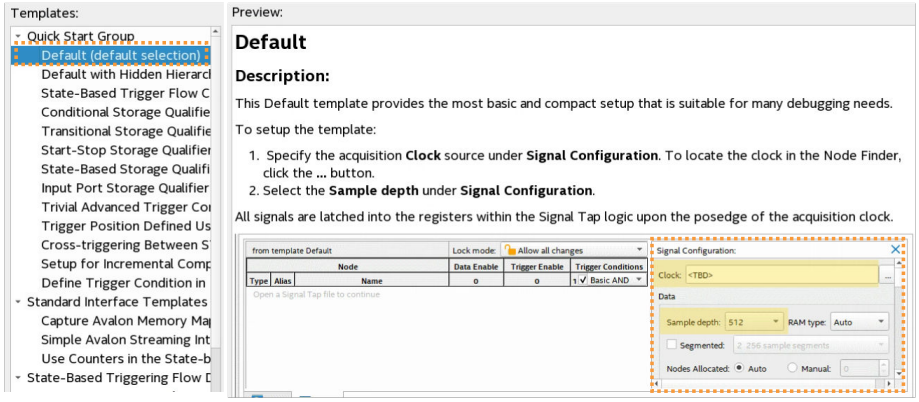
If you want to monitor multiple clock domains simultaneously, you can add additional instances of the logic analyzer to your design, limited only by the available resources in your device.

2.3.1. Creating a Signal Tap Instance with the Signal Tap GUI

When you define one or more Signal Tap instances in the GUI, Signal Tap stores the trigger and signal configuration settings in a Signal Tap Logic Analyzer File (`.stp`). You can open a `.stp` to reload that Signal Tap configuration.

1. Open a project and run **Analysis & Synthesis** on the Compilation Dashboard.
2. To create a Signal Tap instance with the Signal Tap logic analyzer GUI, perform one of the following:
   - Click **Tools ➤ Signal Tap Logic Analyzer**.
   - Click **File ➤ New ➤ Signal Tap Logic Analyzer File**.

Figure 22. Signal Tap file Templates



3. Select a Signal Tap file template. The **Preview** describes the setup and **Signal Configuration** the template applies. Refer to Signal Tap File Templates on page 116.

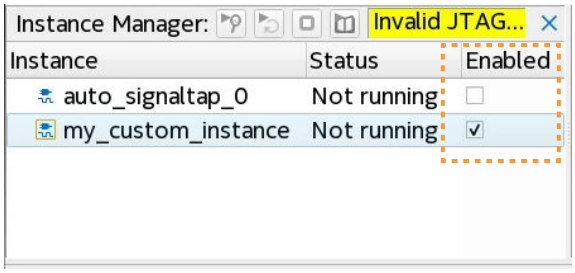
4. Click **Create**. The Signal Tap logic analyzer GUI opens with the template options preset for the Signal Tap instance.
5. Under **Signal Configuration**, specify the acquisition **Clock** and optionally modify other settings, as Step 2: Configure the Signal Tap Logic Analyzer on page 39 describes.
6. When you save or close the Signal Tap instance, click **Yes** when prompted to add the Signal Tap instance to the project.

2.3.1.1. Managing Signal Tap Instances

You can manage the properties of different Signal Tap instances in the **Instance Manager** pane. You can enable or disable one or more instances to specify whether your project includes the instance the next time you run compilation. If you enable or disable instances, you must recompile the design to implement the changes.

The **Instance Manager** toolbar allows you to **Run Analysis** and **Stop Analysis**, or start **Autorun Analysis**, which starts the Signal Tap logic analyzer in a repetitive acquisition mode, providing continuous display update.

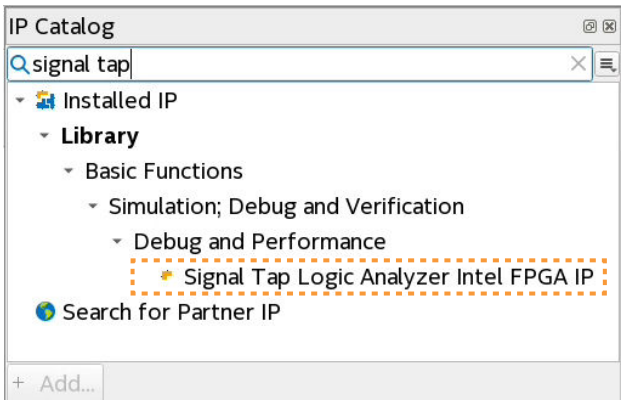
Figure 23. Enable and Disable Signal Tap Instances in Instance Manager



2.3.2. Creating a Signal Tap Instance by HDL Instantiation

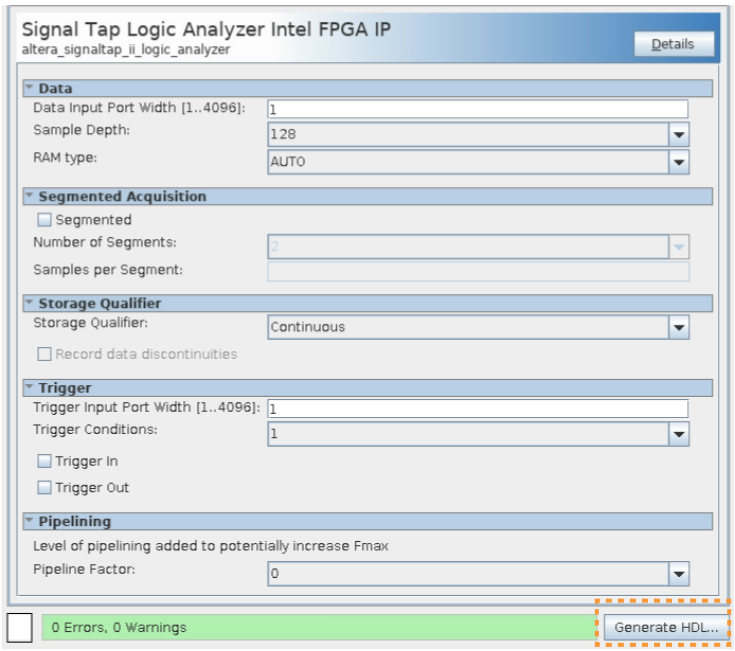
You can create a Signal Tap Instance by HDL instantiation, rather than using the Signal Tap logic analyzer GUI. When you use HDL instantiation, you first parameterize and instantiate the Signal Tap Logic Analyzer Intel FPGA IP in your RTL. Next, you compile the design and IP, and run a Signal Tap analysis using the generated `.stp` file. Follow these steps to create a Signal Tap instance by HDL instantiation:

Figure 24. Signal Tap Logic Analyzer Intel FPGA IP



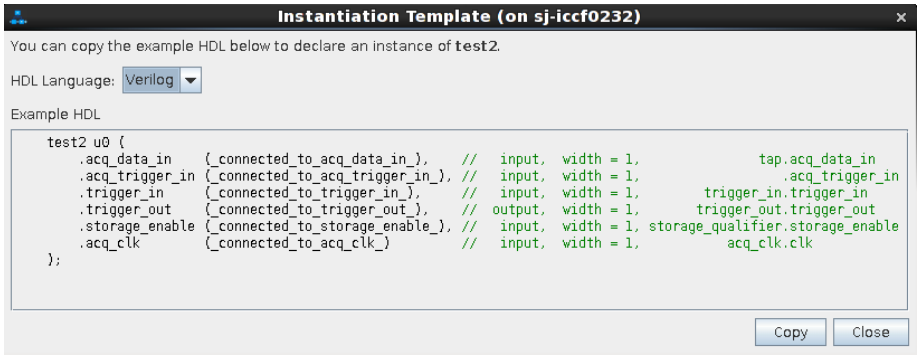
1. From the Intel Quartus Prime IP Catalog (**View ➤ IP Catalog**), locate and double-click the **Signal Tap Logic Analyzer Intel FPGA IP**.
2. In the **New IP Variant** dialog box, specify the **File Name** for your Signal Tap instance, and then click **Create**. The IP parameter editor displays the available parameter settings for the Signal Tap instance.
3. In the parameter editor, specify the **Data**, **Segmented Acquisition**, **Storage Qualifier**, **Trigger**, and **Pipelining** parameters, as Signal Tap Intel FPGA IP Parameters on page 37 describes.
4. Click Generate HDL. The parameter editor generates the HDL implementation of the Signal Tap instance according to your specifications.

Figure 25. IP Parameter Editor



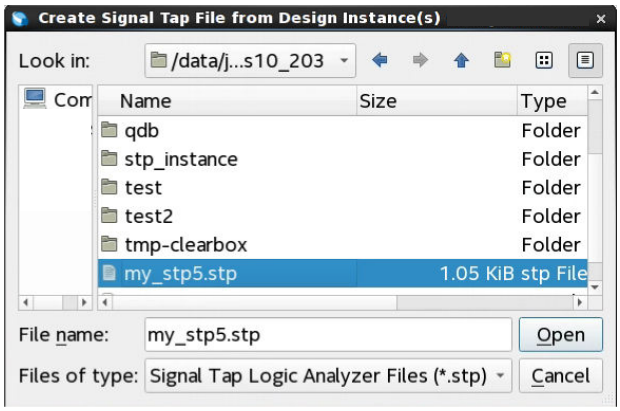
5. To instantiate the Signal Tap instance in your RTL, click **Generate ➤ Show Instantiation Template** in the parameter editor. **Copy** the **Instantiation Template** contents into your RTL.

Figure 26. Signal Tap Logic Analyzer Intel FPGA IP Instantiation Template



6. Run at least the Analysis & Synthesis stage of the Compiler to synthesize the RTL (including Signal Tap instance) by clicking **Processing ➤ Start ➤ Start Analysis & Synthesis**. Alternatively, you can run full compilation and the Assembler at this point if ready.
7. When the Compiler completes, click **Create/Update ➤ Create Signal Tap File from Design Instance** to create a `.stp` file for analysis in the Signal Tap logic analyzer GUI.

Figure 27. Create Signal Tap File from Design Instances Dialog Box



Note: If your project contains partial reconfiguration partitions, the PR partitions display in a tree view. Select a partition from the view, and click **Create Signal Tap file**. The resulting `.stp` file that generates contains all HDL instances in the corresponding PR partition. The resulting `.stp` file does not include the instances in any nested partial reconfiguration partition.

8. To analyze the Signal Tap instance, click **File ➤ Open** and select the `.stp` file. The Signal Tap instance opens in the Signal Tap logic analyzer GUI for analysis. All the fields are read-only, except runtime-configurable trigger conditions.
9. Modify any runtime-configurable trigger conditions, as Runtime Reconfigurable Options on page 89 describes.

2.3.2.1. Signal Tap Intel FPGA IP Parameters

The Signal Tap Intel FPGA IP has the following parameters:

Table 8. Signal Tap Intel FPGA IP Parameters

Parameter Groups	Parameter Descriptions
Data	• **Data Input Port Width**—from 1 to 4096. Default is 1. • **Sample Depth**—number of samples to collect from 0-128K. Default is 128. • **RAM type**—memory type for sample collection and storage. The **Auto** (default), **M20K/M10K/M9K**, **MLAB/LUTRAM**, and **M144K** options are available.
Segmented Acquisition	Specifies options for organizing the captured data buffer:
	continued...

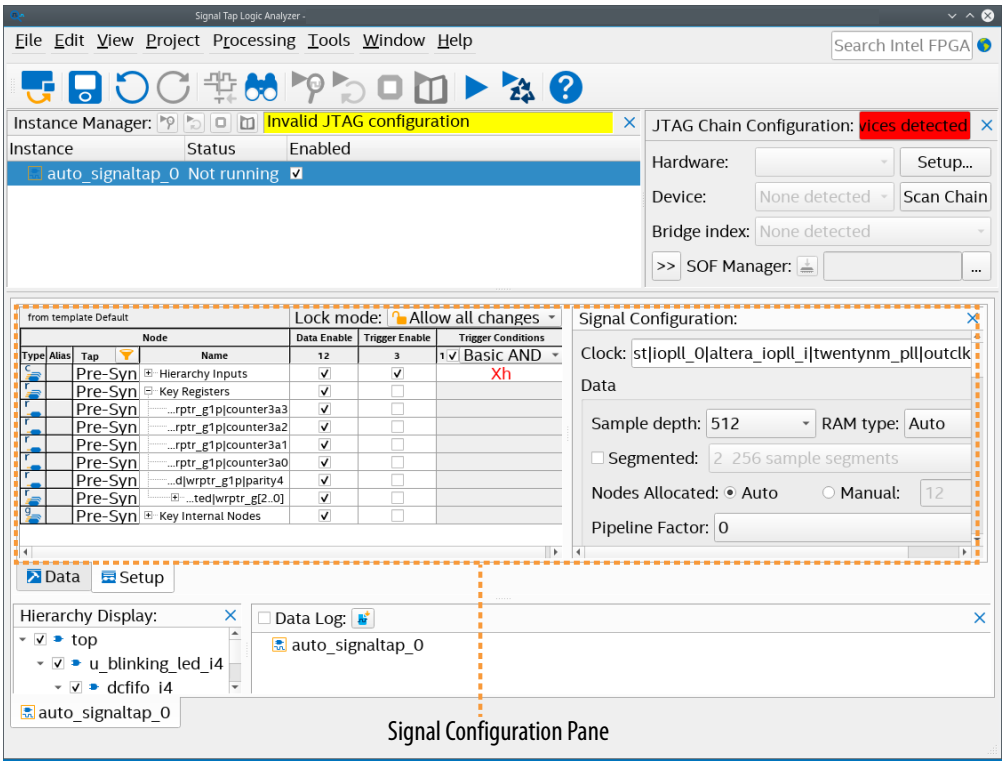
Parameter Groups	Parameter Descriptions
	• **Segmented**—the memory space is split into separate buffers. Each buffer acts as a separate FIFO with its own set of trigger conditions, and behaves as a non-segmented buffer. Only a single buffer is active during an acquisition. Default is off. • **Number of Segments**—specifies the number of segments in each memory space. Default is 2. • **Samples per Segments**—the number of samples Signal Tap captures per segment. Default is 64.
Storage Qualifier	Specifies the **Continuous** or **Input Port** method, and whether to **Record data discontinuities**.
Trigger	• **Trigger Input Port Width**—from 1 to 4096. Default is 1. • **Trigger Conditions**—number of trigger conditions or levels you are implementing 1-10. Default is 1. • **Trigger In**—enables and creates a port for the Trigger In. • **Trigger Out**—enables and creates a port for the Trigger Out.
Pipelining	The Pipeline Factor specifies the levels of pipelining added for potential f_{MAX} improvement from 0 to 5. Default is 0.

2.4. Step 2: Configure the Signal Tap Logic Analyzer

You must configure the Signal Tap logic analyzer before you can capture and analyze data. You can configure instances of the Signal Tap logic analyzer by specifying options in the Signal Tap **Signal Configuration** pane.

When you use the available Signal Tap templates to create a new Signal Tap instance, the template specifies many of the initial option values automatically.

Figure 28. Signal Tap Logic Analyzer Signal Configuration Pane



Basic configuration of the Signal Tap logic analyzer includes specifying values for the following options:

- Preserving Signals for Monitoring and Debugging on page 40
- Specifying the Clock, Sample Depth, and RAM Type on page 42
- Specifying the Buffer Acquisition Mode on page 43
- Adding Signals to the Signal Tap Logic Analyzer on page 45
- Defining Trigger Conditions on page 52
- Specifying Pipeline Settings on page 75
- Filtering Relevant Samples on page 75
- Managing Multiple Signal Tap Configurations on page 101

Related Information

Preventing Changes that Require Full Recompilation on page 42

2.4.1. Preserving Signals for Monitoring and Debugging

The Compiler optimizes the RTL signals during synthesis and place-and-route. Unless preserved, the signal names in your RTL may not exist in the post-fit netlist after signal optimizations. For example, the compilation process can merge duplicate registers, or add tildes (~) to net names that fan-out from a node.

To ensure that specific nodes in your RTL are available for Signal Tap debugging after synthesis and place-and-route, you can apply the `preserve_for_debug` attribute to the signals of interest in your RTL, and also specify the **Enable preserve for debug assignments** project `.qsf` setting. Refer to `.qsf` syntax in Debug Signal Preservation Methods.

When you preserve signals using this technique, the Compiler generates the Preserve for Debug Assignments report following synthesis that shows the status and name of all nodes with the `preserve_for_debug` attribute in your RTL.

Follow these steps to preserve signals for monitoring and debugging:

1. In your design RTL, mark signals that you want to preserve with the `preserve_for_debug` attribute:

Figure 29. preserve_for_debug Attribute

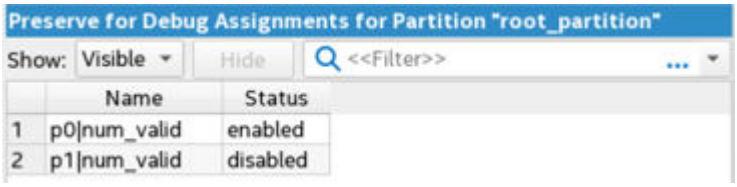
```
module blinking_led_2s(
    output value,
    input clk1,
    input clk2);

    reg [32:0] count_in;
    reg [32:0] count_out;
    reg value_in;
    reg  value_out;
    wire fifo_out;
    wire fifo_empty;
    wire fifo_rreq;
    (* preserve_for_debug *) reg fifo_wreq;
```

2. Open the project containing Signal Tap in the Intel Quartus Prime software and perform one of the following:
   - To enable preservation and reporting for specific instances, click **Assignments ➤ Assignment Editor**, and then specify the **Enable preserve for debug assignments** assignment **To** any instance of interest.

     Or
   - To enable preservation and reporting project-wide, in **Assignments ➤ Settings ➤ Signal Tap Logic Analyzer**, turn on **Enable preserve for debug assignments**.(1)
3. To synthesize the design, on the Compilation Dashboard, click **Analysis & Synthesis**. The Compilation Report appears when synthesis is complete.
4. To view the results of signal preservation, open the Preserve for Debug Assignments report located in the **Synthesis ➤ Partition <name> ➤ Preserve for Debug** report folder.

(1) The global project setting has a more limited impact and does not preserve signals that would otherwise be optimized away in their local context.

Figure 30. Preserve for Debug Assignments Report



5. Run full compilation to perform place and route of the design and Signal Tap instance, as Step 3: Compile the Design and Signal Tap Instances on page 82 describes. The debug signals that you preserve in step 2 persist through the Fitter into the finalized compilation database.
6. Optionally, make some incremental changes to the Signal Tap configuration without running full recompilation, as Changing the Post-Fit Signal Tap Target Nodes on page 86 describes.

Table 9. Debug Signal Preservation Methods

Method	Description	Example
`preserve_for_debug_enable`	Set this assignment to **On** to preserve any nodes or hierarchies marked with `preserve_for_debug`. If set to **Off** or not used, any `preserve_for_debug` assignments are ignored. Use this as a quick way to disable all debug node preservation when optimizing a completed design. The Compiler reports these nodes in the Preserve for Debug Assignments report following compilation.	`set_instance_assignment -name PRESERVE_FOR_DEBUG_ENABLE ON`
`preserve_for_debug` (**Enable preserve for debug assignments** in the Assignment Editor)	Instance-specific `.qsf` assignment that overrides the global assignment and enables preservation of all types of nodes through synthesis post-synthesis or post-fit debugging purposes. When **On**, this assignment enables preservation for the hierarchy that you specify. You can enable or disable this with the **Preserve signal for debug** assignment in the Assignment Editor. The Compiler reports these nodes in the Preserve for Debug Assignments report following compilation.	`set_instance_assignment -name PRESERVE_FOR_DEBUG ON -to <node hpath>`

Note: For more information about preserving signals, refer to *Preserving Registers During Synthesis*, in the *Intel Hyperflex™ Architecture High-Performance Design Handbook* and *Preserving a System Module, Interface, or Port for Debugging* in the *Intel Quartus Prime Pro Edition User Guide: Platform Designer*.

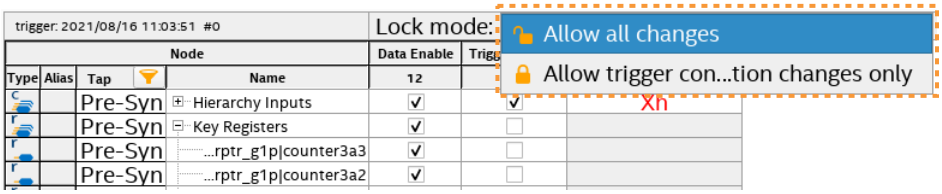
Related Information

- Intel Hyperflex Architecture High-Performance Design Handbook
- Changing the Post-Fit Signal Tap Target Nodes on page 86
- Preserving Signals for Debugging on page 21
- Preserving a System Module, Interface, or Port for Debugging, Intel Quartus Prime Pro Edition User Guide: Platform Designer

2.4.2. Preventing Changes that Require Full Recompilation

Making some types of changes to the Signal Tap configuration require full recompilation to implement. If you want to ensure that you make no changes to the Signal Tap configuration that require full recompilation, select **Allow trigger condition changes only** for the **Lock mode**. Alternatively, you can enable **Allow all changes**, including those changes that require full compilation or recompilation to implement.

Figure 31. Allow Trigger Conditions Change Only



Related Information

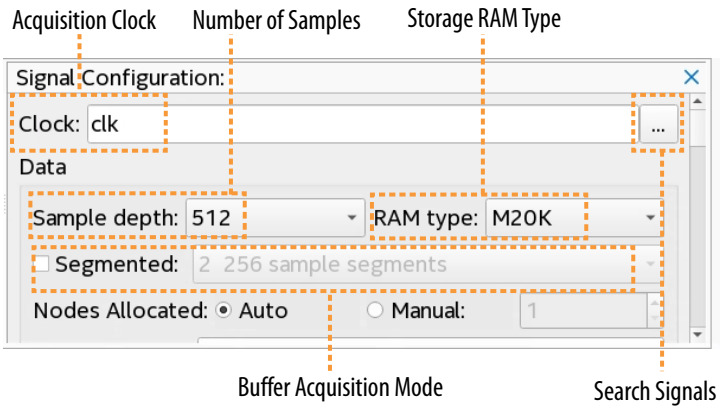
Recompiling Only Signal Tap Changes on page 82

2.4.3. Specifying the Clock, Sample Depth, and RAM Type

You must specify options for the acquisition clock, sample depth, and data storage on the **Signal Configuration** pane before using Signal Tap.

Note: The Signal Tap file templates automatically specify appropriate initial values for some of these options.

Figure 32. Clock, Sample Depth, and Data Storage Options



Specifying the Acquisition Clock

Signal Tap samples data on each positive (rising) edge of the acquisition clock. Therefore, Signal Tap requires a clock signal from your design to control the logic analyzer data acquisition. For best data acquisition, specify a global, non-gated clock that is synchronous to the signals under test. Refer to the Timing Analysis section of the Compilation Report for the maximum frequency of the logic analyzer clock.

- To specify the acquisition clock signal, enter a signal name from your design for the **Clock** setting in **Single Configuration**.

Note: Consider the following when specifying the acquisition clock:

- If you do not assign an acquisition clock, Signal Tap automatically creates clock pin `auto_stp_external_clk`. You must then make a pin assignment to this signal, and ensure that a clock signal in your design drives the acquisition clock.
- Using a transceiver recovered clock as the acquisition clock can cause incorrect or unexpected behavior, particularly when the transceiver recovered clock is the acquisition clock with the power-up trigger feature.
- Specifying a gated acquisition clock can result in unexpected data that does not accurately reflect the behavior of your design.
- Signal Tap does not support sampling on the negative (falling) clock edge.

Specifying Sample Depth

The sample depth determines the number of samples the logic analyzer captures and stores in the data buffer, for each signal. In cases with limited device memory resources, you can reduce the sample depth to reduce resource usage.

- To specify the sample depth, select the number of samples from the **Sample depth** list under **Single Configuration**. Available sample depth range is from 0 to 128K.

Specifying the RAM Type

You can specify the RAM type and buffer acquisition mode for storage of Signal Tap logic analyzer acquisition data. When you allocate the Signal Tap logic analyzer buffer to a particular RAM block, the entire RAM block becomes a dedicated resource for the logic analyzer.

- To specify the RAM type, select a **Ram type** under **Single Configuration**. Available settings are **Auto**, **MLAB**, or **M20K** RAM.

Use RAM selection to preserve a specific memory block for your design, and allocate another portion of memory for Signal Tap data acquisition. For example, if your design has an application that requires a large block of memory resources, such as a large instruction or data cache, use MLAB blocks for data acquisition and leave M20k blocks for your design.

Related Information

- Adding Nios II Processor Signals with a Plug-In on page 50
- Managing Device I/O Pins, Intel Quartus Prime Pro Edition User Guide: Design Constraints

2.4.4. Specifying the Buffer Acquisition Mode

You can specify how Signal Tap organizes the data capture buffer to potentially reduce the amount of memory that Signal Tap requires for data acquisition.

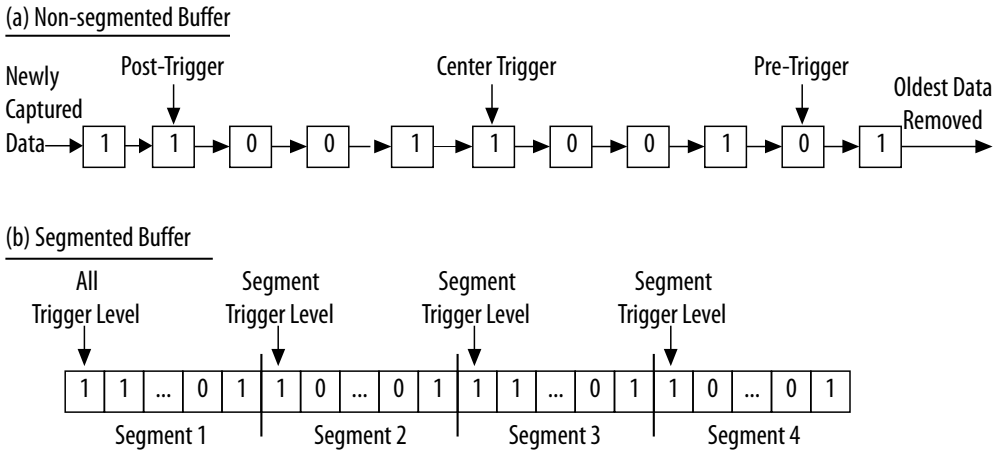
The Signal Tap logic analyzer supports either a non-segmented (or circular) buffer and a segmented buffer.

- **Non-segmented buffer**—the Signal Tap logic analyzer treats the entire memory space as a single FIFO, continuously filling the buffer until the logic analyzer reaches the trigger conditions that you specify.
- **Segmented buffer**—the memory space is split into separate buffers. Each buffer acts as a separate FIFO with its own set of trigger conditions, and behaves as a non-segmented buffer. Only a single buffer is active during an acquisition. The Signal Tap logic analyzer advances to the next segment after the trigger condition or conditions for the active segment has been reached.

When using a non-segmented buffer, you can use the storage qualification feature to determine which samples are written into the acquisition buffer. Both the segmented buffers and the non-segmented buffer with the storage qualification feature help you maximize the use of the available memory space.

Figure 33. Buffer Type Comparison in the Signal Tap Logic Analyzer

The figure illustrates the differences between the two buffer types.



Both non-segmented and segmented buffers can use a preset trigger position (Pre-Trigger, Center Trigger, Post-Trigger). Alternatively, you can define a custom trigger position using the **State-Based Triggering** tab, as Specify Trigger Position on page 61 describes.

2.4.4.1. Non-Segmented Buffer

The non-segmented buffer is the default buffer type in the Signal Tap logic analyzer.

At runtime, the logic analyzer stores data in the buffer until the buffer fills up. From that point on, new data overwrites the oldest data, until a specific trigger event occurs. The amount of data the buffer captures after the trigger event depends on the **Trigger position** setting:

- To capture more data from before the trigger occurs, select **Post trigger position** from the list.
- To capture all data from after the trigger occurs, select **Pre trigger position**.
- To center the trigger position in the data, select **Center trigger position**.

Alternatively, use the custom State-based triggering flow to define a custom trigger position within the capture buffer.

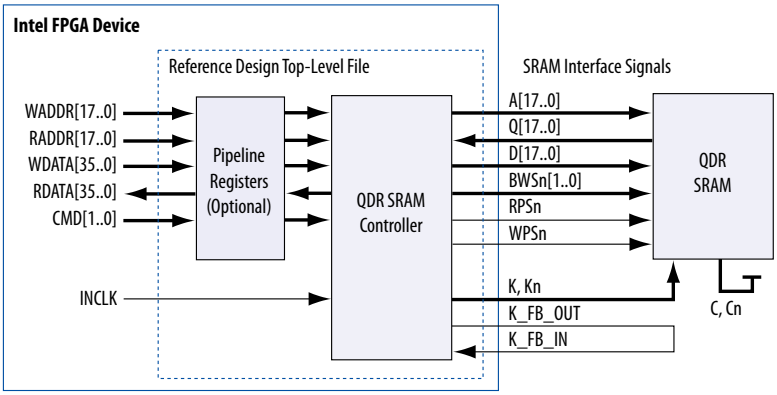
2.4.4.2. Segmented Buffer

In a segmented buffer, the acquisition memory is split into segments of even size, and you define a set of trigger conditions for all segments. Each segment acts as a non-segmented buffer. A segmented buffer allows you to debug systems that contain relatively infrequent recurring events.

If you want to have separate trigger conditions for each of the buffer segments, you must use the state-based trigger flow. The figure shows an example of a segmented buffer system.

Figure 34. System that Generates Recurring Events

In the following example, to ensure that the correct data is written to the SRAM controller, monitor the RDATA port whenever the address `H'0F0F0F0F` is sent into the RADDR port.



The buffer acquisition feature allows you to monitor multiple read transactions from the SRAM device without running the Signal Tap logic analyzer again. You can split the memory to capture the same event multiple times, without wasting allocated memory. The buffer captures as many cycles as the number of segments you define under the **Data** settings in the **Signal Configuration** pane.

To enable and configure buffer acquisition, select **Segmented** in the Signal Tap logic analyzer Editor and choose the number of segments to use. In the example in the figure, selecting 64-sample segments allows you to capture 64 read cycles.

Related Information

Viewing Capture Data Using Segmented Buffers on page 93

2.4.5. Adding Signals to the Signal Tap Logic Analyzer

You add the signals that you want to monitor to the node list in the Signal Tap logic analyzer. You can then select a signals in the node list to define the triggers for the signal.

Adding Pre-Synthesis Signals

You can add expected signals to Signal Tap for monitoring without running synthesis. Pre-synthesis signal names are those names present after Analysis & Elaboration, but before any synthesis optimizations. When you add pre-synthesis signals to Signal Tap

for monitoring, you must make all connections to the Signal Tap logic analyzer before running synthesis. The Compiler then automatically allocates the logic and routing resources to make these connections. For signals driving to and from IOEs, pre-synthesis signal names coincide with the pin's signal names.

Refer to Adding Pre-Synthesis or Post-Fit Nodes on page 46.

Adding Simulator-Aware Signals

You can easily generate a list of simulator-aware, pre-synthesis signals to tap for an entire design hierarchy, and then observe all internal signal states in your RTL simulator. This set of simulator-aware nodes can provide full visibility into other untapped nodes in the design hierarchy. You can then export captured Signal Tap signal data directly into your RTL simulator to observe signal states beyond Signal Tap observability.

Refer to Adding Simulator-Aware Signal Tap Nodes on page 48.

Adding Post-Fit Signals

You can add post-fit signals to Signal Tap for monitoring. Post-fit signal names are those names present in the netlist after physical synthesis optimizations and place-and-route. When you add post-fit signals to Signal Tap for monitoring, you are connecting to actual atoms in the post-fit netlist. You can only monitor signals that exist in the post-fit netlist, and existing routing resources must be available.

In the case of post-fit output signals, monitor the COMBOUT or REGOUT signal that drives the IOE block. For post-fit input signals, signals driving into the core logic coincide with the pin's signal name.

Note: Because NOT-gate push back applies to any register that you monitor, the signal from the atom may be inverted. You can verify the inversion by locating to the signal with the **Locate Node ➤ Locate in Resource Property Editor** or the **Locate Node ➤ Locate in Technology Map Viewer** commands. You can also view post-fit node names in the Resource Property Editor.

Related Information

Signal Tap and Simulator Integration on page 98

2.4.5.1. Adding Pre-Synthesis or Post-Fit Nodes

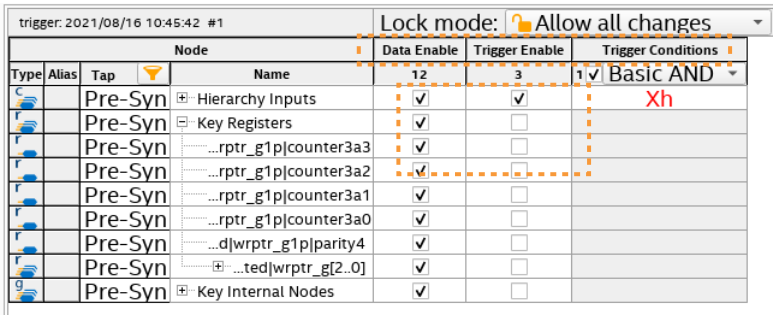
To add one or more pre-synthesis or post-fit signals to the Signal Tap **Node** list for monitoring:

1. Click either of the following commands to generate the pre-synthesis or post-fit design netlist:
   - **Processing ➤ Start ➤ Start Analysis & Elaboration** (generates pre-synthesis netlist)
   - **Processing ➤ Start ➤ Start Fitter** (generates post-fit netlist)
2. In the Signal Tap logic analyzer, Click **Edit ➤ Add Nodes**. The Node Finder appears, allowing you to find and add the signals in your design. The following Filter options are available for finding the nodes you want:

- **Signal Tap: pre-synthesis**—finds signal names present after design elaboration, but before any synthesis optimizations are done. **Signal Tap: pre-synthesis preserved for debug** finds presynthesis signals that you mark with the `preserve_for_debug` pragma, as Preserving Signals for Monitoring and Debugging on page 40 describes.
- **Signal Tap: post-fitting**—finds signal names present after physical synthesis optimizations and place-and-route. **Signal Tap: post-fitting preserved for debug** finds post-fit signals that you mark with the `preserve_for_debug` pragma.

3. In the Node Finder, select one or more nodes that you want to add, and then click the **Copy all to Selected Nodes list** button.
4. Click **Insert**. The nodes are added to the **Setup** tab signal list in the Signal Tap logic analyzer GUI.
5. Specify how the logic analyzer uses the signal by enabling or disabling the **Data Enable**, **Trigger Enable**, or **Storage Enable** option for the signal:
   - **Trigger Enable**—disabling prevents a signal from triggering the analysis, while still showing the signal's captured data.
   - **Data Enable**—disabling prevent capture of data, while still allowing the signal to trigger.

Figure 35. Signal Tap Node List Options for Data Enable and Trigger Enable



6. Define trigger conditions for the Signal Tap nodes, as Defining Trigger Conditions on page 52 describes.

The number of channels available in the Signal Tap window waveform display is directly proportional to the number of logic elements (LEs) or adaptive logic modules (ALMs) in the device. Therefore, there is a physical restriction on the number of channels that are available for monitoring. Signals shown in blue text are post-fit node names. Signals shown in black text are pre-synthesis node names.

After successful Analysis and Elaboration, invalid signals appear in red. Unless you are certain that these signals are valid, remove them from the `.stp` file for correct operation. The Signal Tap Status Indicator also indicates if an invalid node name exists in the `.stp` file.

You can monitor signals only if a routing resource (row or column interconnects) exists to route the connection to the Signal Tap instance. For example, you cannot monitor signals that exist in the I/O element (IOE), because there are no direct routing resources from the signal in an IOE to a core logic element. For input pins, you can monitor the signal that is driving a logic array block (LAB) from an IOE, or, for output pins, you can monitor the signal from the LAB that is driving an IOE.

Note: The Intel Quartus Prime Pro Edition software uses only the instance name, and not the entity name, in the form of:

```
a|b|c
```

not `a_entity:a|b_entity:b|c_entity:c`

2.4.5.2. Adding Simulator-Aware Signal Tap Nodes

Note: This version of the Signal Tap simulator integration feature is a beta release. The following known limitations apply to this beta release:

- Supports only Verilog HDL simulation.
- Supports testbench generation only within the current project directory.

To automatically generate and add a list of simulator-aware signals to the Signal Tap **Node** list for Signal Tap and simulator monitoring, follow these steps:

1. To generate the pre-synthesis design netlist, click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
2. In the Signal Tap logic analyzer, click **Edit ➤ Add Simulator Aware Nodes**. The **Simulator Aware Node Finder** opens, allowing you to specify the following options to find and add the minimum set of nodes to tap to for full visibility into the selected hierarchy's cone of logic:
   a. Click the **Select Hierarchies** button, select one or more design hierarchies that you want to tap, and then click **OK**. The clock domains in the hierarchy appear in the **Clock Domains** list.
   b. Under **Clock Domains**, enable only the domains of interest. If you select multiple clock domains, Signal Tap creates an instance for each domain.
   c. Click the **Search** button. All nodes required to provide full visibility into the selected hierarchy automatically appear enabled in the **Total nodes to tap** list. Disabling any of the simulator-aware nodes may reduce simulation visibility.
   d. Click the **Insert** button. The enabled signals in the **Total nodes to tap** list are copied to the Signal Tap **Node** list, and the acquisition clock updates according to the simulator-aware signal data. Refer to Add Simulator-Aware Node Finder Settings on page 50.

Figure 36. Simulator Aware Node Finder

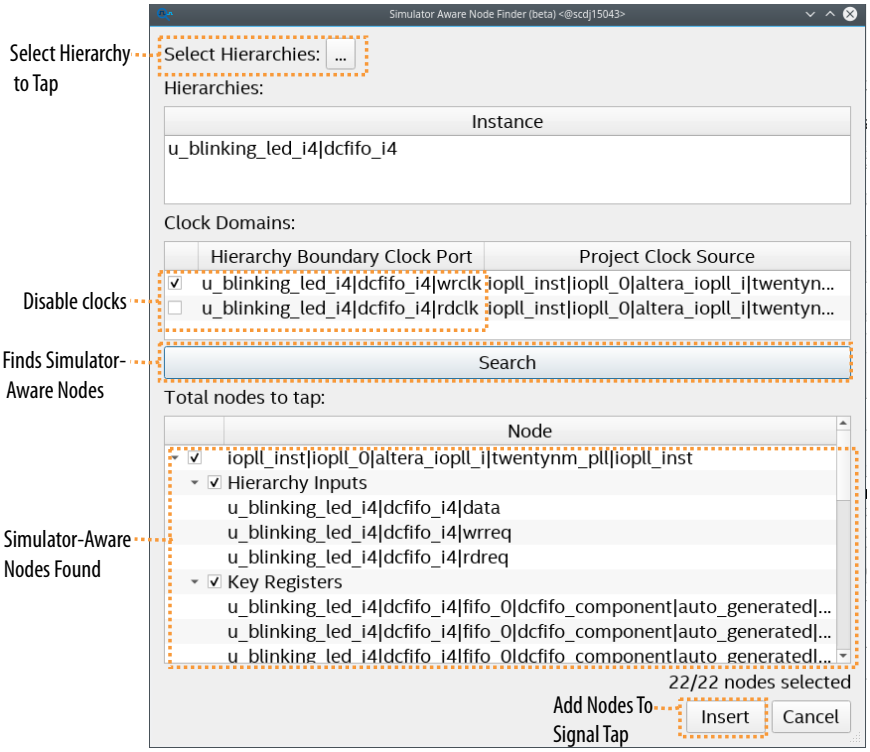
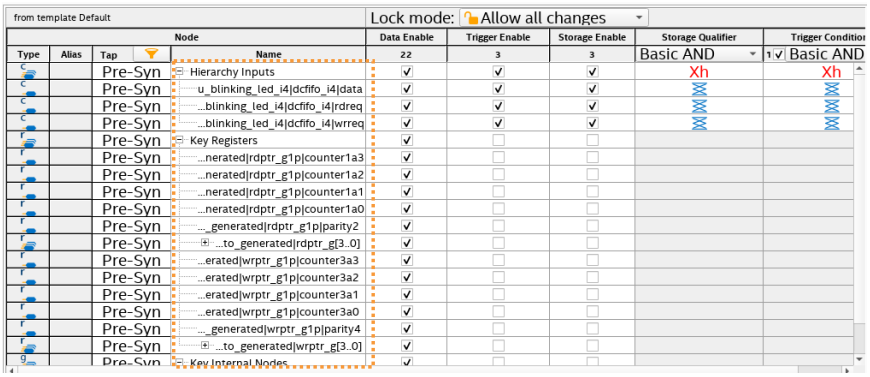


Figure 37. Simulator-Aware Nodes Copied to Signal Tap Window



3. Modify trigger conditions for the Signal Tap nodes, as Defining Trigger Conditions on page 52 describes.
4. Compile the design and Signal Tap instance, Step 3: Compile the Design and Signal Tap Instances on page 82 describes.
5. Program the target hardware, as Step 4: Program the Target Hardware on page 85 describes.
6. Run the Signal Tap logic analyzer, as Step 5: Run the Signal Tap Logic Analyzer on page 86 describes.
7. Generate a simulation testbench from Signal Tap capture data, as Generating a Simulation Testbench from Signal Tap Data on page 98 describes.

2.4.5.2.1. Add Simulator-Aware Node Finder Settings

The following options are available for searching and adding simulator aware nodes to Signal Tap for the purpose of generating an RTL simulation testbench from Signal Tap data. The default values derive from Signal Tap signal data and are set correctly for most scenarios.

Table 10. Add Simulator Aware Node Finder Settings (Signal Tap Logic Analyzer)

Name	Description
Select Hierarchies	Specifies the design hierarchy from which to extract simulator-aware nodes. Select one or more design hierarchies that you want to tap. The clock domains of the hierarchy appear in the **Clock Domains** list. Only nodes from the hierarchy you specify are added.
Clock Domains	Specifies the clock domains to include in the simulator-aware node finder. Turn on only the domains that you want to include.
Search button	Starts the search for simulator-aware nodes according to the specifications in this dialog box. Search results appear in the **Total nodes to tap** list.
Total nodes to tap	Displays the results of the simulator-aware node name search, showing all of the names in the hierarchy enabled by default. Turn the node names on to include or off to exclude from the list of nodes added to Signal Tap. Disabling any of the simulator-aware nodes may reduce simulation visibility.
Insert Button	Copies the enabled signals in the **Total nodes to tap** list to the Signal Tap **Node** list, and the acquisition clock updates according to the simulator-aware signal data.

2.4.5.3. Adding Nios II Processor Signals with a Plug-In

You can use a plug-in to automatically add relevant signals for the Nios II processor for monitoring, rather than adding the signals manually with the Node Finder. The plug-in provides preset mnemonic tables for trigger creation and viewing, as well as the ability to disassemble code in captured data.

Note: This feature does not yet support the Nios V embedded processor.

The Nios II plug-in creates one mnemonic table in the **Setup** tab and two tables in the **Data** tab:

- **Nios II Instruction** (**Setup** tab)—capture all the required signals for triggering on a selected instruction address.
- **Nios II Instance Address** (**Data** tab)—display address of executed instructions in hexadecimal format or as a programming symbol name if defined in an optional Executable and Linking Format (`.elf`) file.
- **Nios II Disassembly** (**Data** tab)—display disassembled code from the corresponding address.

To add Nios II IP signals to the logic analyzer using a plug-in, perform the following steps after running Analysis and Elaboration on your design:

1. In the Signal Tap logic analyzer, right-click the node list, and then click **Add Nodes with Plug-In ➤ Nios II**.
2. Select the IP that contains the signals you want to monitor with the plug-in, and click **OK**.

— If all the signals in the plug-in are available, a dialog box might appear, depending on the plug-in, where you can specify options for the plug-in.

3. With the Nios II plug-in, you can optionally select an `.elf` containing program symbols from your Nios II Integrated Development Environment (IDE) software design. Specify options for the selected plug-in, and click **OK**.

2.4.5.4. Signals Unavailable for Signal Tap Debugging

Some post-fit signals in your design are unavailable for Signal Tap debugging. The Node Finder's **Signal Tap: post-fitting** filter does not return nodes that are unavailable for Signal Tap debugging.

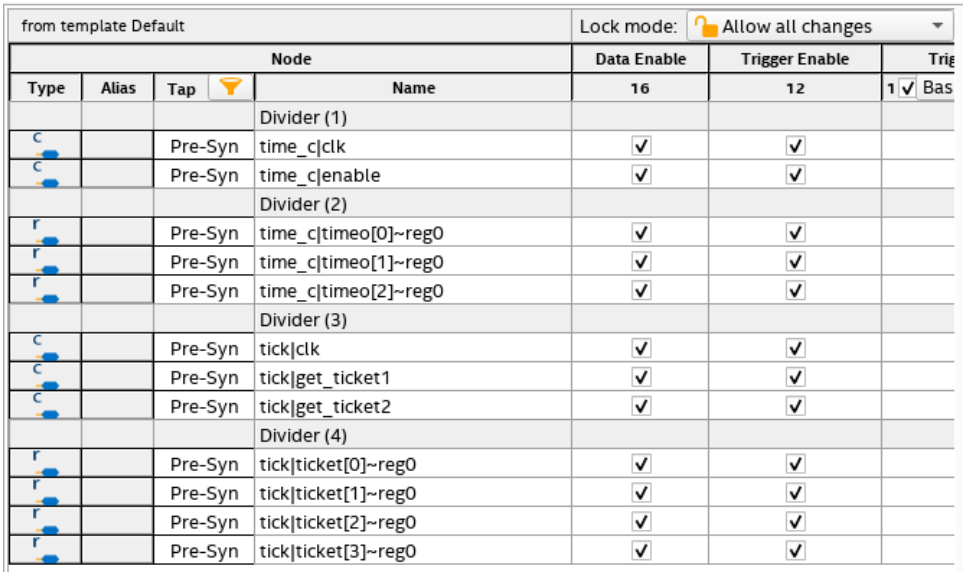
The following signal types are unavailable for Signal Tap debugging:

- **Post-fit output pins**—You cannot monitor a post-fit output or bidirectional pin directly. To make an output signal visible, monitor the register or buffer that drives the output pin.
- **Carry chain signals**—You cannot monitor the carry out (`cout0` or `cout1`) signal of a logic element. Due to architectural restrictions, the carry out signal can only feed the carry in of another LE.
- **JTAG signals**—You cannot monitor the JTAG control (`TCK`, `TDI`, `TDO`, or `TMS`) signals.
- **LVDS**—You cannot monitor the data output from a serializer/deserializer (SERDES) block.
- **DQ**, **DQS signals**—You cannot directly monitor the `DQ` or `DQS` signals in a DDR or DDRII design.

2.4.5.5. Organizing Signals in the Signal Tap Logic Analyzer

Use dividers in the node list to separate and categorize your signals into groups to make reviewing the node table easier. You can move and rename dividers in the node table at any time.

Figure 38. Example Signal Tap Logic Analyzer node table that shows a lists of nodes and dividers



- To add a divider, right-click a node in the node table and select **Add Divider**.

  The new divider is added immediately above the selected node. The divider is assigned a name that you can change at any time.
- To rename a divider, right-click the divider that you want to rename and select **Rename**.
- To move a divider, drag-and-drop the divider to its new location.
- To delete a divider, right-click the divider and select **Delete**.

2.4.6. Defining Trigger Conditions

By default, the Signal Tap logic analyzer captures data continuously from the signals you specify while the logic analyzer is running. To capture and store only specific signal data, you can specify conditions that *trigger* the start or stop of data capture. A trigger activates—that is, the logic analyzer stops and displays the data—when the signals you specify reach the trigger conditions that you define.

The Signal Tap logic analyzer allows you to define trigger conditions that range from very simple, such as the rising edge of a single signal, to very complex, involving groups of signals, extra logic, and multiple conditions. Additionally, you can specify Power-Up Triggers to capture data from trigger events occurring immediately after the device enters user-mode after configuration.

2.4.6.1. Basic Trigger Conditions

If you select the **Basic AND** or **Basic OR** trigger type, you must specify the trigger pattern for each signal that you add.

To specify the trigger pattern, right-click the **Trigger Conditions** column and click **Don't Care**, **Low**, **High**, **Falling Edge** , **Rising Edge**, or **Either Edge**.

For buses, type a pattern in binary, or right-click and select **Insert Value** to enter the pattern in other number formats. Note that you can enter X to specify a set of "don't care" values in either your hexadecimal or your binary string. For signals in the `.stp` file that have an associated mnemonic table, you can right-click and select an entry from the table to specify pre-defined conditions for the trigger.

When you add signals through plug-ins, you can create basic triggers using predefined mnemonic table entries. For example, with the Nios II plug-in, if you specify an `.elf` file from your Nios II IDE design, you can type the name of a function from your Nios II code. The logic analyzer triggers when the Nios II instruction address matches the address of the code function name that you specify.

Data capture stops and the logic analyzer stores the data in the buffer when the logical AND of all the signals for a given trigger condition evaluates to TRUE.

2.4.6.2. Nested Trigger Conditions

When you specify a set of signals as a nested group (group of groups) with the **Basic OR** trigger type, the logic analyzer generates an advanced trigger condition. This condition sorts the signals within groups to minimize the need to recompile your design. If you always retain the parent-child relationship of nodes, the advanced trigger condition does not change. You can modify the sibling relationships of nodes, without requiring recompilation.

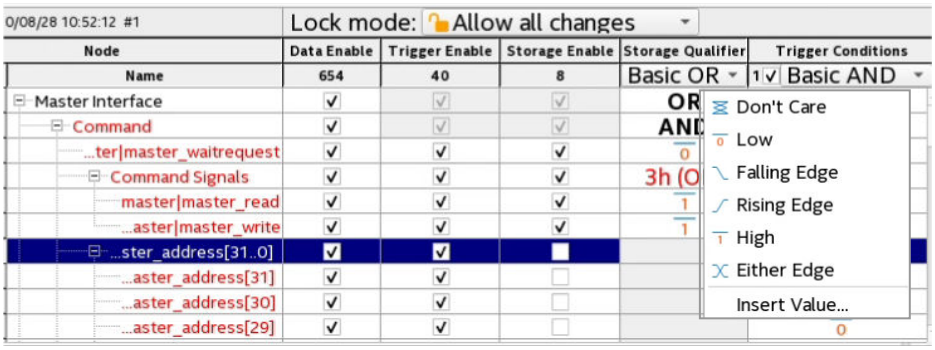
The evaluation precedence of a nested trigger condition starts at the bottom-level with the leaf-groups. The logic analyzer uses the resulting logic value to compute the parent group's logic value. If you manually set the value of a group, the logic value of the group's members doesn't influence the result of the group trigger.

To create a nested trigger condition:

1. Select **Basic OR** under **Trigger Conditions**.
2. In the **Setup** tab, select several nodes. Include groups in your selection.
3. Right-click the **Setup** tab and select **Group**.
4. Select the nested group and right-click to set a group trigger condition that applies the reduction **AND**, **OR**, **NAND**, **NOR**, **XOR**, **XNOR**, or logical **TRUE** or **FALSE**.

   Note: You can only select OR and AND group trigger conditions for bottom-level groups (groups with no groups as children).

Figure 39. Applying Trigger Condition to Nested Group



2.4.6.3. Comparison Trigger Conditions

The **Comparison** trigger allows you to compare multiple grouped bits of a bus to an expected integer value by specifying simple comparison conditions on the bus node. The **Comparison** trigger preserves all the trigger conditions that the **Basic OR** trigger includes. You can use the **Comparison** trigger in combination with other triggers. You can also switch between **Basic OR** trigger and **Comparison** trigger at run-time, without the need for recompilation.

Signal Tap logic analyzer supports the following types of **Comparison** trigger conditions:

- **Single-value comparison**—compares a bus node's value to a numeric value that you specify. Use one of these operands for comparison: >, >=, ==, <=, <, !=. Returns 1 when the bus node matches the specified numeric value.
- **Interval check**—verifies whether a bus node's value confines to an interval that you define. Returns 1 when the bus node's value lies within the specified bounded interval.

Follow these rules when using the **Comparison** trigger condition:

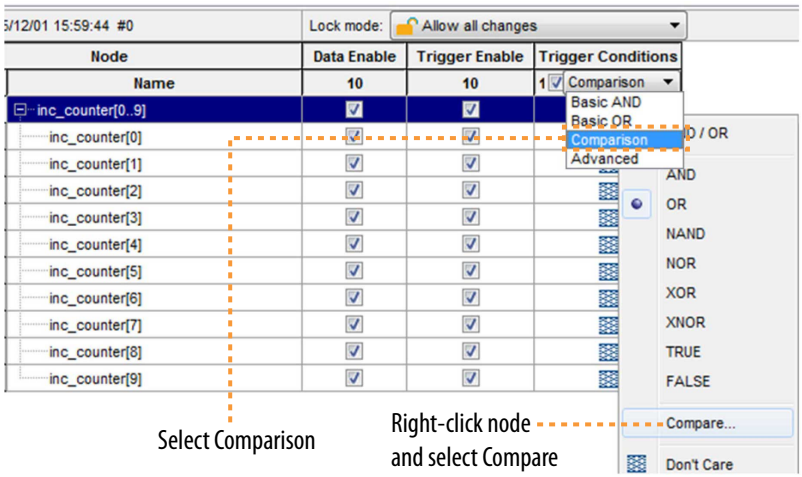
- Apply the **Comparison** trigger only to bus nodes consisting of leaf nodes.
- Do not form sub-groups within a bus node.
- Do not enable or disable individual trigger nodes within a bus node.
- Do not specify comparison values (in case of single-value comparison) or boundary values (in case of interval check) exceeding the selected node's bus-width.

2.4.6.3.1. Specifying the Comparison Trigger Conditions

Follow these steps to specify the **Comparison** trigger conditions:

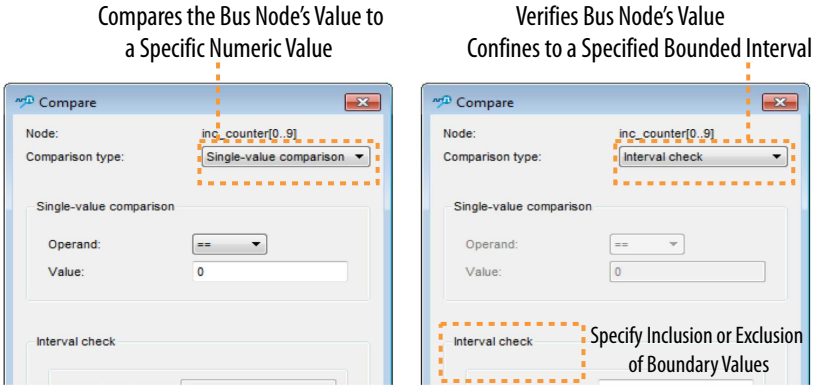
1. From the **Setup** tab, select **Comparison** under **Trigger Conditions**.
2. Right-click the node in the trigger editor, and select **Compare**.
3. Select the **Comparison type** from the Compare window.
   - If you choose **Single-value comparison** as your comparison type, specify the operand and value.
   - If you choose **Interval check** as your comparison type, provide the lower and upper bound values for the interval.

Figure 40. Selecting the Comparison Trigger Condition



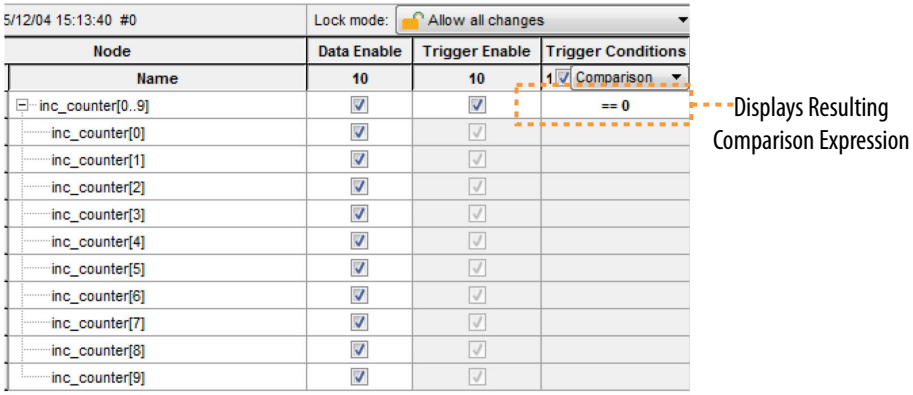
You can also specify if you want to include or exclude the boundary values.

Figure 41. Specifying the Comparison Values



4. Click **OK**. The trigger editor displays the resulting comparison expression in the group node condition text box.

Figure 42. Resulting Comparison Condition in Text Box



2.4.6.4. Advanced Trigger Conditions

To capture data for a given combination of conditions, build an advanced trigger. The Signal Tap logic analyzer provides the **Advanced Trigger** tab, which helps you build a complex trigger expression using a GUI. Open the **Advanced Trigger** tab by selecting **Advanced** in the **Trigger Conditions** list.

Figure 43. Accessing the Advanced Trigger Condition Tab

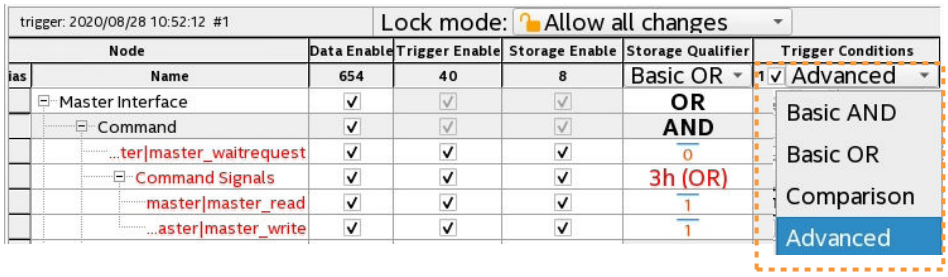
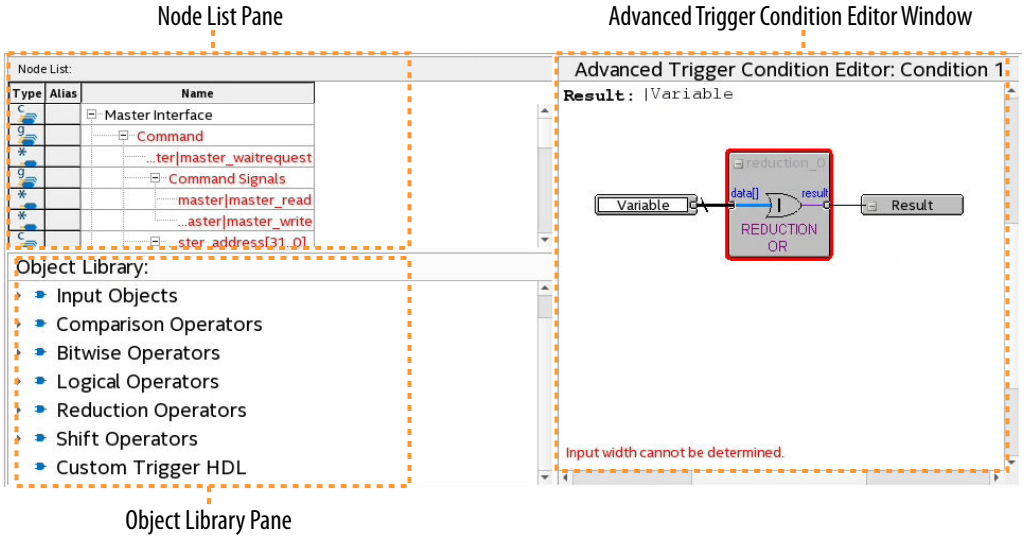


Figure 44. Advanced Trigger Condition Tab



To build a complex trigger condition in an expression tree, drag-and-drop operators from the **Object Library** pane and the **Node List** pane into the **Advanced Trigger Configuration Editor** window.

To configure the operators' settings, double-click or right-click the operators that you placed and click **Properties**.

Table 11. Advanced Triggering Operators

Category	Name
Signal Detection	Edge and Level Detector
Input Objects	Bit Bit Value Bus

continued...

Category	Name
	Bus Value
Comparison	Less Than Less Than or Equal To Equality Inequality Greater Than or Equal To Greater Than
Bitwise	Bitwise Complement Bitwise AND Bitwise OR Bitwise XOR
Logical	Logical NOT Logical AND Logical OR Logical XOR
Reduction	Reduction AND Reduction OR Reduction XOR
Shift	Left Shift Right Shift
Custom Trigger HDL	

Adding many objects to the Advanced Trigger Condition Editor can make the work space cluttered and difficult to read. To keep objects organized while you build your advanced trigger condition, use the shortcut menu and select **Arrange All Objects**. Alternatively, use the **Zoom-Out** command to fit more objects into the **Advanced Trigger Condition Editor** window.

2.4.6.4.1. Examples of Advanced Triggering Expressions

The following examples show how to use advanced triggering:

Figure 45. Bus outa Is Greater Than or Equal to Bus outb

Trigger when bus `outa` is greater than or equal to `outb`.

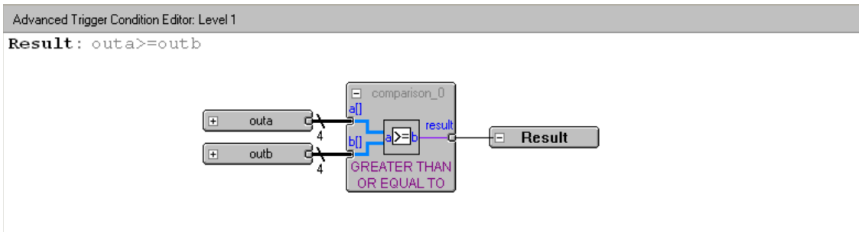


Figure 46. Enable Signal Has a Rising Edge

Trigger when bus `outa` is greater than or equal to bus `outb`, and when the enable signal has a rising edge.

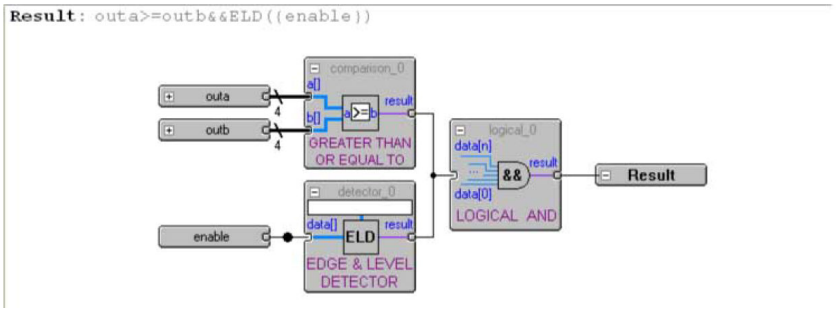
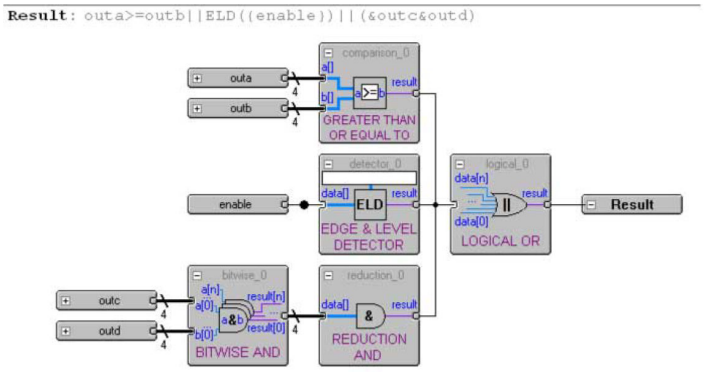


Figure 47. Bitwise AND Operation

Trigger when bus `outa` is greater than or equal to bus `outb`, or when the enable signal has a rising edge. Or, when a bitwise AND operation has been performed between bus `outc` and bus `outd`, and all bits of the result of that operation are equal to 1.

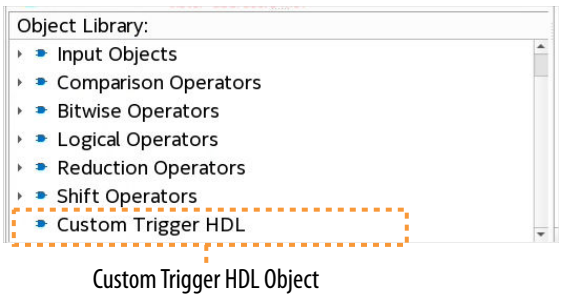


2.4.6.5. Custom Trigger HDL Object

The Signal Tap logic analyzer supports use of your own HDL module to define a custom trigger condition. You can use the Custom Trigger HDL object to simulate your triggering logic and ensure that the logic itself is not faulty. Additionally, you can monitor instances of modules anywhere in the hierarchy of your design, without having to manually route all the necessary connections.

The **Custom Trigger HDL** object appears in the **Object Library** pane of the **Advanced Trigger** editor.

Figure 48. Object Library

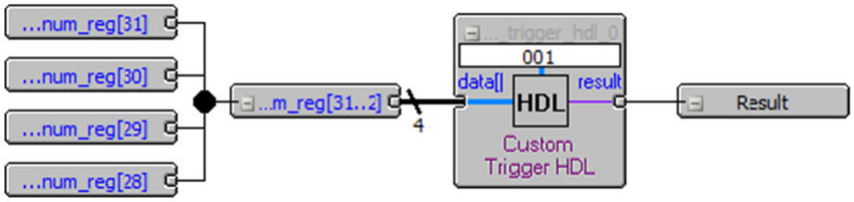


2.4.6.5.1. Using the Custom Trigger HDL Object

To define a custom trigger flow:

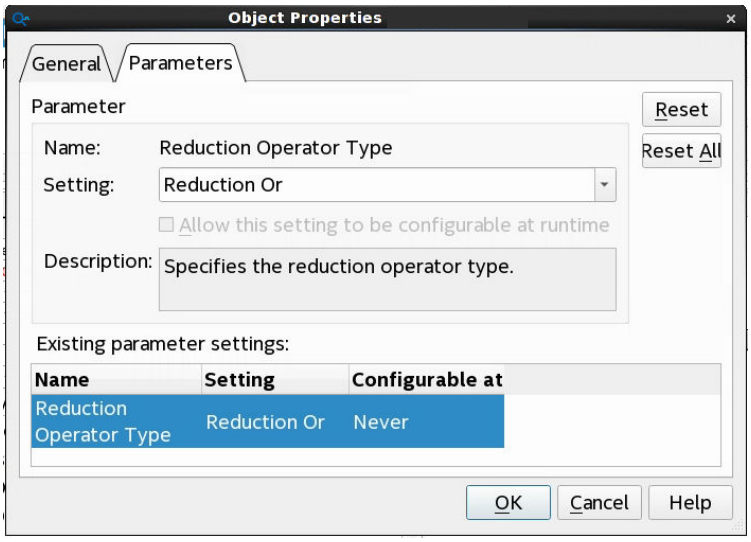
1. Select the trigger you want to edit.
2. Open the **Advanced Trigger** tab by selecting **Advanced** in the **Trigger Conditions** list.
3. Add to your project the Verilog HDL or VHDL source file that contains the trigger module using the **Project Navigator**.
4. Implement the inputs and outputs that your Custom Trigger HDL module requires.
5. Drag in your **Custom Trigger HDL** object and connect the object's data input bus and result output bit to the final trigger result.

Figure 49. Custom Trigger HDL Object



6. Right-click your **Custom Trigger HDL** object and configure the object's properties.

Figure 50. Configure Object Properties



7. Compile your design.
8. Acquire data with Signal Tap using your custom Trigger HDL object.

Example 1. Verilog HDL Triggers

The following trigger uses configuration bitstream:

```
module test_trigger
    (
        input acq_clk, reset,
        input[3:0] data_in,
        input[1:0] pattern_in,
        output reg trigger_out
    );
    always @(pattern_in) begin
        case (pattern_in)
            2'b00:
                trigger_out = &data_in;
            2'b01:
                trigger_out = |data_in;
            2'b10:
                trigger_out = 1'b0;
            2'b11:
                trigger_out = 1'b1;
        endcase
    end
endmodule
```

This trigger does not have configuration bitstream:

```
module test_trigger_no_bs
    (
        input acq_clk, reset,
        input[3:0] data_in,
        output reg trigger_out
    );
    assign trigger_out = &data_in;
endmodule
```

2.4.6.5.2. Required Inputs and Outputs of Custom Trigger HDL Module

Table 12. Custom Trigger HDL Module Required Inputs and Outputs

Name	Description	Input/Output	Required/ Optional
acq_clk	Acquisition clock that Signal Tap uses	Input	Required
reset	Reset that Signal Tap uses when restarting a capture.	Input	Required
data_in	• Data input you connect in the Advanced Trigger editor. • Data your module uses to trigger.	Input	Required
pattern_in	• Module's input for the configuration bitstream property. • Runtime configurable property that you can set from Signal Tap GUI to change the behavior of your trigger logic.	Input	Optional
trigger_out	Output signal of your module that asserts when trigger conditions met.	Output	Required

2.4.6.5.3. Custom Trigger HDL Module Properties

Table 13. Custom Trigger HDL Module Properties

Property	Description
Custom HDL Module Name	Module name of the triggering logic.
Configuration Bitstream	• Allows to create trigger logic that you can configure at runtime, based upon the value of the configuration bitstream. • The Signal Tap logic analyzer reads the configuration bitstream property as binary, therefore the bitstream must contain only the characters `1` and `0`. • The bit-width (number of `1`s and `0`s) must match the `pattern_in` bit width. • A blank configuration bitstream implies that the module does not have a `pattern_in` input.
Pipeline	Specifies the number of pipeline stages in the triggering logic. For example, if after receiving a triggering input the LA needs three clock cycles to assert the trigger output, you can denote a pipeline value of three.

2.4.6.6. Specify Trigger Position

You can specify the amount of data the logic analyzer acquires before and after a trigger event. Positions for Runtime and Power-Up triggers are separate.

The Signal Tap logic analyzer offers three pre-defined ratios of pre-trigger data to post-trigger data:

- **Pre**—saves signal activity that occurred after the trigger (12% pre-trigger, 88% post-trigger).
- **Center**—saves 50% pre-trigger and 50% post-trigger data.
- **Post**—saves signal activity that occurred before the trigger (88% pre-trigger, 12% post-trigger).

These pre-defined ratios apply to both non-segmented buffers and each segment of a buffer.

2.4.6.6.1. Post-fill Count

In a custom state-based triggering flow with the `segment_trigger` and `trigger` buffer control actions, you can use the `post-fill_count` argument to specify a custom trigger position.

- If you do not use the `post-fill_count` argument, the trigger position for the affected buffer defaults to the trigger position you specified in the **Setup** tab.
- In the `trigger` buffer control action (for non-segmented buffers), `post-fill_count` specifies the number of samples to capture before stopping data acquisition.
- In the `segment_trigger` buffer control action (for segmented buffer), `post-fill_count` specifies a data segment.

Note: In the case of `segment_trigger`, acquisition of the current buffer stops immediately if a subsequent triggering action is issued in the next state, regardless of the current buffer's post-fill count. The logic analyzer discards the remaining unfilled post-count acquisitions in the current buffer, and displays them as grayed-out samples in the data window.

When the Signal Tap data window displays the captured data, the trigger position appears as the number of post-count samples from the end of the acquisition segment or buffer.

Sample Number of Trigger Position = (*N* – *Post-Fill Count*)

In this case, *N* is the sample depth of either the acquisition segment or non-segmented buffer.

Related Information

Buffer Control Actions on page 72

2.4.6.7. Power-Up Triggers

Power-Up Triggers capture events that occur during device initialization, immediately after you power or reset the FPGA.

The typical use of Signal Tap logic analyzer is triggering events that occur during normal device operation. You start an analysis manually once the target device fully powers on and the JTAG connection for the device is available. With Signal Tap Power-Up Trigger feature, the Signal Tap logic analyzer captures data immediately after device initialization.

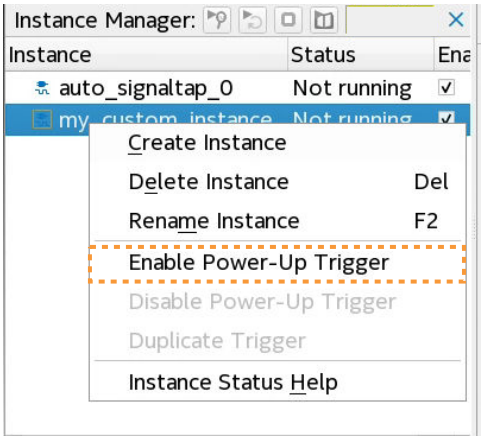
You can add a different Power-Up Trigger to each logic analyzer instance in the **Signal Tap Instance Manager** pane.

2.4.6.7.1. Enabling a Power-Up Trigger

To enable the Power-Up Trigger for Signal Tap instance:

- In the Instance Manager, right-click the Signal Tap instance and click **Enable Power-Up Trigger**.

Figure 51. Enabling Power-Up Trigger in Signal Tap Instance Manager



Power-Up Trigger appears as a child instance of the selected Signal Tap instance. The first time you enable Power-Up Trigger, the **Trigger conditions** column in the **Setup** tab is populated with the default values. Subsequently, when you disable Power-Up Trigger, the current values in the **Setup** tab is retrieved the next time you re-enable Power-Up Trigger.

To disable a Power-Up Trigger, right-click the instance and click **Disable Power-Up Trigger**.

2.4.6.7.2. Configuring Power-Up Trigger Conditions

- Any change that you make to a Power-Up Trigger conditions requires that you recompile the Signal Tap logic analyzer instance, even if a similar change to the Runtime Trigger conditions does not require a recompilation.
- You can also force trigger conditions with the In-System Sources and Probes in conjunction with the Signal Tap logic analyzer. The In-System Sources and Probes feature allows you to drive and sample values on to selected nets over the JTAG chain.

Related Information

Design Debugging Using In-System Sources and Probes on page 145

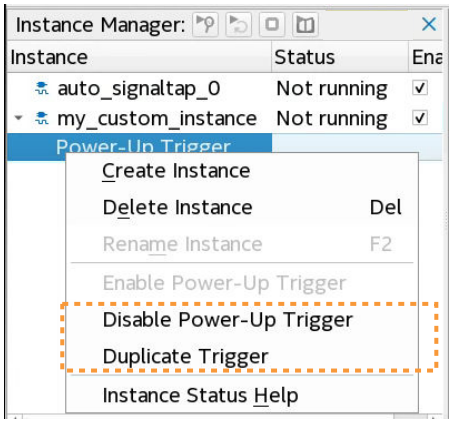
2.4.6.7.3. Managing Signal Tap Instances with Run-Time and Power-Up Trigger Conditions

Inside Instance Manager, on instances that have two types of trigger conditions, Power-Up Trigger conditions are color coded light blue, while Run-Time Trigger conditions remain white.

To switch between the trigger conditions of the Power-Up Trigger and the Run-Time Trigger, double-click the instance name or the Power-Up Trigger name in the **Instance Manager**.

To copy trigger conditions from a Run-Time Trigger to a Power-Up Trigger or vice versa, right-click the trigger name in the **Instance Manager** and click **Duplicate Trigger**. Alternatively, select the trigger name and click **Edit ➤ Duplicate Trigger**.

Figure 52. Instance Manager Commands



Note: Run-time trigger conditions allow fewer adjustments than power-up trigger conditions.

2.4.6.8. External Triggers

External trigger inputs allow you to trigger the Signal Tap logic analyzer from an external source.

The external trigger input behaves like trigger condition 0, in that the condition must evaluate to TRUE before the logic analyzer evaluates any other trigger conditions.

The Signal Tap logic analyzer supplies a signal to trigger external devices or other logic analyzer instances. These features allow you to synchronize external logic analysis equipment with the internal logic analyzer. Power-Up Triggers can use the external triggers feature, but they must use the same source or target signal as their associated Run-Time Trigger.

You can use external triggers to perform cross-triggering on a hard processor system (HPS):

- The processor debugger allows you to configure the HPS to obey or disregard cross-trigger request from the FPGA, and to issue or not issue cross-trigger requests to the FPGA.
- The processor debugger in combination with the Signal Tap external trigger feature allow you to develop a dynamic combination of cross-trigger behaviors.
- You can implement a system-level debugging solution for an Intel FPGA SoC by using the cross-triggering feature with the ARM Development Studio 5 (DS-5) software.

2.4.6.9. Trigger Condition Flow Control

The Trigger Condition Flow Control allows you to define the relationship between a set of triggering conditions. Signal Tap logic analyzer **Signal Configuration** pane offers two flow control mechanisms for organizing trigger conditions:

- **Sequential Triggering**—default triggering flow. Sequential triggering allows you to define up to 10 triggering levels that must be satisfied before the acquisition buffer finishes capturing.
- **State-Based Triggering**—gives the greatest control over your acquisition buffer. Custom-based triggering allows you to organize trigger conditions into states based on a conditional flow that you define.

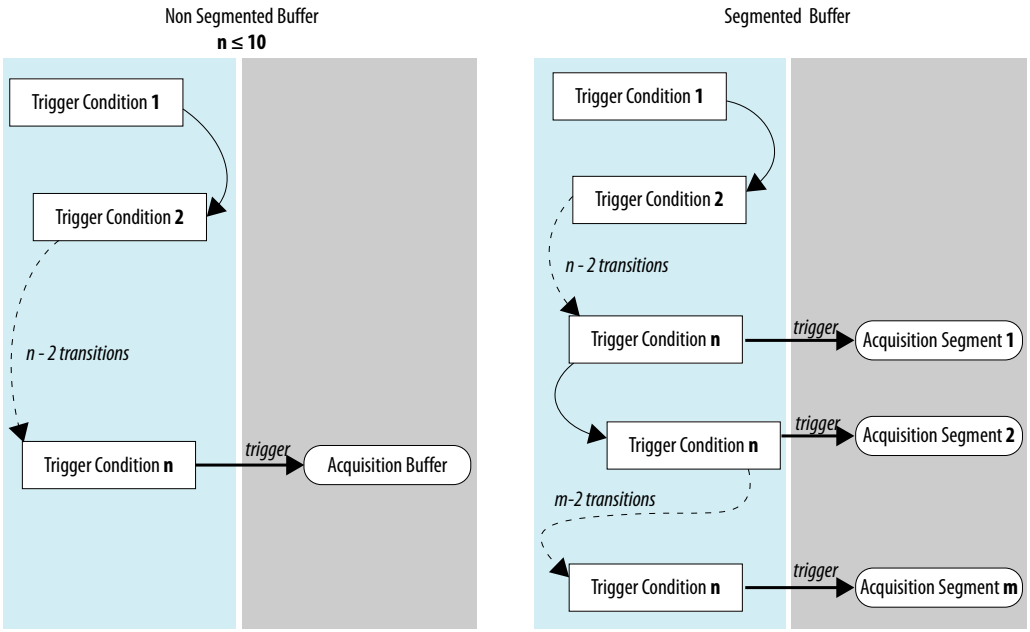
You can use sequential or state based triggering with either a segmented or a non-segmented buffer.

2.4.6.10. Sequential Triggering

When you specify a sequential trigger the Signal Tap logic analyzer sequentially evaluates each the conditions. The sequential triggering flow allows you to cascade up to 10 levels of triggering conditions.

When the last triggering condition evaluates to TRUE, the Signal Tap logic analyzer starts the data acquisition. For segmented buffers, every acquisition segment after the first starts on the last condition that you specified. The Signal Tap **Node** annotates this final condition column with Seg if a segmented buffer is enabled. The Simple Sequential Triggering feature allows you to specify basic triggers, comparison triggers, advanced triggers, or a mix of all three. The following figure illustrates the simple sequential triggering flow for non-segmented and segmented buffers. The acquisition buffer starts capture when all n triggering levels are satisfied, where $n \leq 10$.

Figure 53. Sequential Triggering Flow



The Signal Tap logic analyzer considers external triggers as level 0, evaluating external triggers before any other trigger condition.

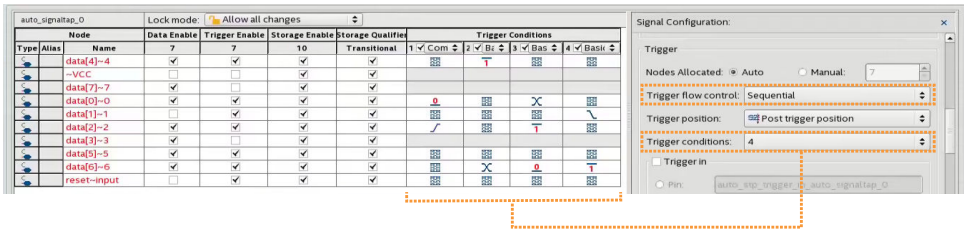
2.4.6.10.1. Configuring the Sequential Triggering Flow

To configure Signal Tap logic analyzer for sequential triggering:

1. On **Trigger Flow Control**, select **Sequential**
2. On **Trigger Conditions**, select the number of trigger conditions from the drop-down list.
   The **Node List** pane now displays the same number of trigger condition columns.
3. Configure each trigger condition in the **Node List** pane.

   You can enable/disable any trigger condition from the column header.

Figure 54. Sequential Triggering Flow Configuration



2.4.6.11. State-Based Triggering

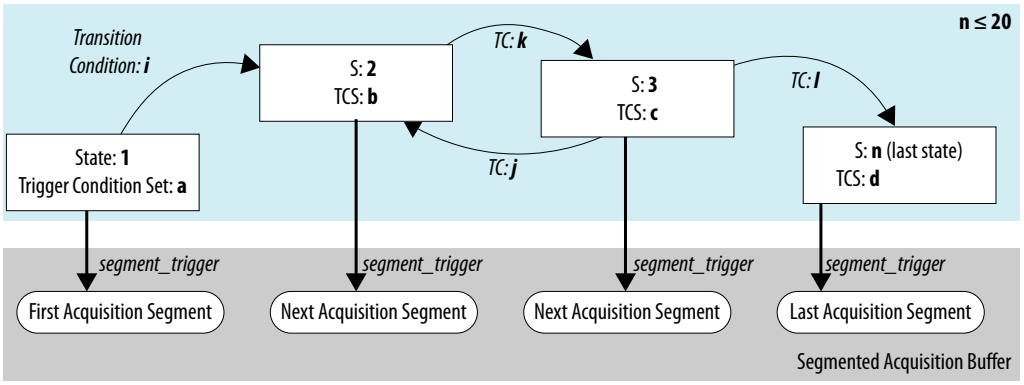
With state-based triggering, a state diagram organizes the events that trigger the acquisition buffer. The states capture all actions that the acquisition buffer performs, and each state contains conditional expressions that define transition conditions.

Custom state-based triggering grants control over triggering condition arrangement. Because the logic analyzer only captures samples of interest, custom state-based triggering allows for more efficient use of the space available in the acquisition buffer.

To help you describe the relationship between triggering conditions, the state-based triggering flow provides tooltips in the GUI. Additionally, you can use the Signal Tap Trigger Flow Description Language, which is based upon conditional expressions.

Each state allows you to define a set of conditional expressions. Conditional expressions are Boolean expressions that depend on a combination of triggering conditions, counters, and status flags. You configure the triggering conditions within the **Setup** tab. The Signal Tap logic analyzer custom-based triggering flow provides counters and status flags.

Figure 55. State-Based Triggering Flow



Within each conditional expression you define a set of actions. Actions include triggering the acquisition buffer to stop capture, a modification to either a counter or status flag, or a state transition.

Trigger actions can apply to either a single segment of a segmented acquisition buffer or to the entire non-segmented acquisition buffer. Each trigger action provides an optional count that specifies the number of samples the buffer captures before the logic analyzer stops acquisition of the current segment. The count argument allows you to control the amount of data the buffer captures before and after a triggering event occurs.

Resource manipulation actions allow you to increment and decrement counters or set and clear status flags. The logic analyzer uses counter and status flag resources as optional inputs in conditional expressions. Counters and status flags are useful for counting the number of occurrences of certain events and for aiding in triggering flow control.

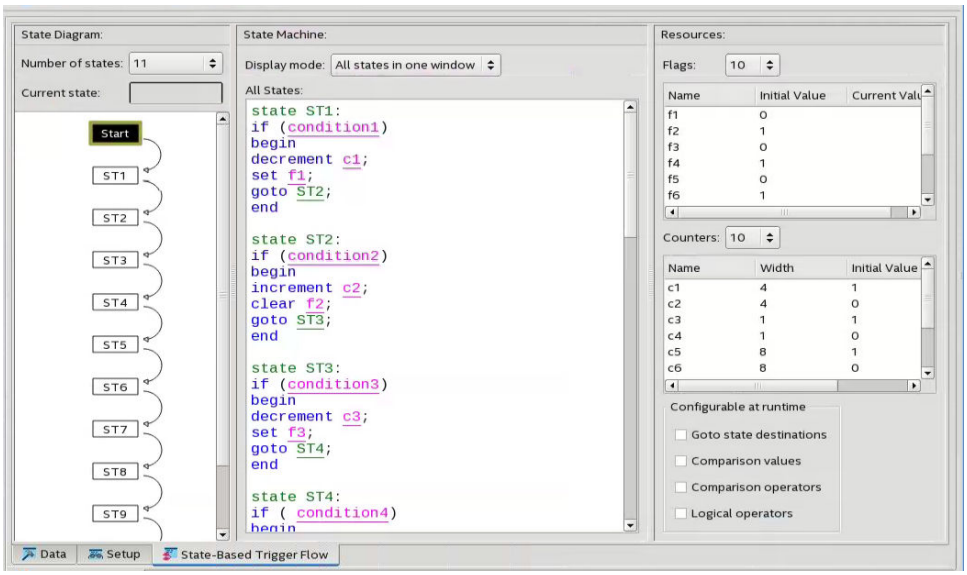
The state-based triggering flow allows you to capture a sequence of events that may not necessarily be contiguous in time. For example, a communication transaction between two devices that includes a hand shaking protocol containing a sequence of acknowledgments.

2.4.6.11.1. State-Based Triggering Flow Tab

The **State-Based Trigger Flow** tab is the control interface for the custom state-based triggering flow.

This tab is only available when you select **State-Based** on the **Trigger Flow Control** list. If you specify **Trigger Flow Control** as **Sequential**, the **State-Based Trigger Flow** tab is not visible.

Figure 56. State-Based Triggering Flow Tab



The **State-Based Trigger Flow** tab contains three panes:

2.4.6.11.2. State Machine Pane

The **State Machine** pane contains the text entry boxes where you define the triggering flow and actions associated with each state.

- You can define the triggering flow using the Signal Tap Trigger Flow Description Language, a simple language based on "if-else" conditional statements.
- Tooltips appear when you move the mouse over the cursor, to guide command entry into the state boxes.
- The GUI provides a syntax check on your flow description in real-time and highlights any errors in the text flow.

The State Machine description text boxes default to show one text box per state. You can also have the entire flow description shown in a single text field. This option can be useful when copying and pasting a flow description from a template or an external text editor. To toggle between one window per state, or all states in one window, select the appropriate option under **State Display mode.**

Related Information

Signal Tap Trigger Flow Description Language on page 68

2.4.6.11.3. Resources Pane

The **Resources** pane allows you to declare status flags and counters for your Custom Triggering Flow's conditional expressions.

- You can increment/decrement counters or set/clear status flags within your triggering flow.
- You can specify up to 20 counters and 20 status flags.
- To initialize counter and status flags, right-click the row in the table and select **Set Initial Value.**
- To specify a counter width, right-click the counter in the table and select **Set Width**.
- To assist in debugging your trigger flow specification, the logic analyzer dynamically updates counters and flag values after acquisition starts.

The **Configurable at runtime** settings allow you to control which options can change at runtime without requiring a recompilation.

Table 14. Runtime Reconfigurable Settings, State-Based Triggering Flow

Setting	Description
Destination of `goto` action	Allows you to modify the destination of the state transition at runtime.
Comparison values	Allows you to modify comparison values in Boolean expressions at runtime. In addition, you can modify the `segment_trigger` and trigger action post-fill count argument at runtime.
Comparison operators	Allows you to modify the operators in Boolean expressions at runtime.
Logical operators	Allows you to modify the logical operators in Boolean expressions at runtime.

Related Information

- Performance and Resource Considerations on page 84
- Runtime Reconfigurable Options on page 89

2.4.6.11.4. State Diagram Pane

The **State Diagram** pane provides a graphical overview of your triggering flow. this pane displays the number of available states and the state transitions. To adjust the number of available states, use the menu above the graphical overview.

2.4.6.11.5. Signal Tap Trigger Flow Description Language

The Trigger Flow Description Language is based on a list of conditional expressions per state to define a set of actions.

To describe the actions that the logic analyzer evaluates when a state is reached, follow this syntax:

Syntax of Trigger Flow Description Language

```
state <state_label>:
    <action_list>
    if (<boolean_expression>)
        <action_list>
    [else if (<boolean_expression>)
```

```
        <action_list>]
    [else
        <action_list>]
```

- Non-terminals are delimited by "<>".
- Optional arguments are delimited by "[]".
- The priority for evaluation of conditional statements is from top to bottom.
- The Trigger Flow Description Language allows multiple `else if` conditions.

<state_label> on page 69
<boolean_expression> on page 69
<action_list> on page 70
Trigger that Skips Clock Cycles after Hitting Condition on page 71
Storage Qualification with Post-Fill Count Value Less than m on page 72
Resource Manipulation Action on page 72
Buffer Control Actions on page 72
State Transition Action on page 73

Related Information

Custom State-Based Triggering Flow Examples on page 114

<state_label>

Identifies a given state. You use the state label to start describing the actions the logic analyzer evaluates once said state is reached. You can also use the state label with the `goto` command.

The state description header syntax is:
`state` *<state_label>*

The description of a state ends with the beginning of another state or the end of the whole trigger flow description.

<boolean_expression>

Collection of operators and operands that evaluate into a Boolean result. The operators can be logical or relational. Depending on the operator, the operand can reference a trigger condition, a counter and a register, or a numeric value. To group a set of operands within an expression, you use parentheses.

Table 15. Logical Operators

Logical operators accept any boolean expression as an operand.

Operator	Description	Syntax
!	NOT operator	`! expr1`
&&	AND operator	`expr1 && expr2`
||	OR operator	`expr1 || expr2`

Table 16. Relational Operators

You use relational operators on counters or status flags.

Operator	Description	Syntax
>	Greater than	`<identifier> > <numerical_value>`
>=	Greater than or Equal to	`<identifier> >= <numerical_value>`
==	Equals	`<identifier> == <numerical_value>`
!=	Does not equal	`<identifier> != <numerical_value>`
<=	Less than or equal to	`<identifier> <= <numerical_value>`
<	Less than	`<identifier> < <numerical_value>`

Notes to table:
1. *<identifier>* indicates a counter or status flag.
2. *<numerical_value>* indicates an integer.

Note:

- The *<boolean_expression>* in an `if` statement can contain a single event or multiple event conditions.
- When the boolean expression evaluates TRUE, the logic analyzer evaluates all the commands in the *<action_list>* concurrently.

<action_list>

List of actions that the logic analyzer performs within a state once a condition is satisfied.

- Each action must end with a semicolon (`;`).
- If you specify more than one action within an `if` or an `else if` clause, you must delimit the `action_list` with `begin` and `end` tokens.

Possible actions include:

Buffer Control Actions

Actions that control the acquisition buffer.

Table 17. Buffer Control Actions

Action	Description	Syntax
`trigger`	Stops the acquisition for the current buffer and ends analysis. This command is required in every flow definition.	`trigger <post-fill_count>;`
`segment_trigger`	Available only in segmented acquisition mode. Ends acquisition of the current segment. After evaluating this command, the Signal Tap logic analyzer starts acquiring from the next segment. If all segments are written, the logic analyzer overwrites the oldest segment with the latest sample. When a trigger action is evaluated the acquisition stops.	`segment_trigger <post-fill_count>;`

continued...

Action	Description	Syntax
`start_store`	Active only in state-based storage qualifier mode. Asserts the `write_enable` to the Signal Tap acquisition buffer.	`start_store`
`stop_store`	Active only in state-based storage qualifier mode. De-asserts the `write_enable` signal to the Signal Tap acquisition buffer.	`stop_store`

Both `trigger` and `segment_trigger` actions accept an optional `post-fill_count` argument.

State Transition Action

Specifies the next state in the custom state control flow. The syntax is: `goto` *<state_label>*;

Trigger that Skips Clock Cycles after Hitting Condition

Trigger flow description that skips three clock cycles of samples after hitting condition 1

Code:

```
State 1: ST1
    start_store
    if ( condition1 )
    begin
       stop_store;
       goto ST2;
    end
State 2: ST2
    if (c1 < 3)
       increment c1; //skip three clock cycles; c1 initialized to 0
    else if (c1 == 3)
    begin
       start_store;//start_store necessary to enable writing to finish
                   //acquisition
       trigger;
    end
```

The figures show the data transaction on a continuous capture and the data capture when you apply the Trigger flow description.

Figure 57. Continuous Capture of Data Transaction

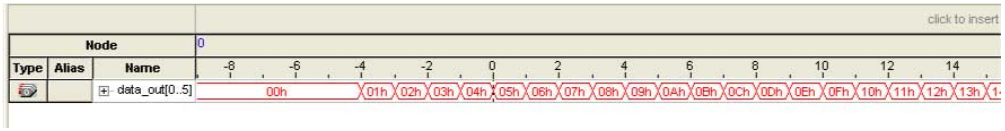
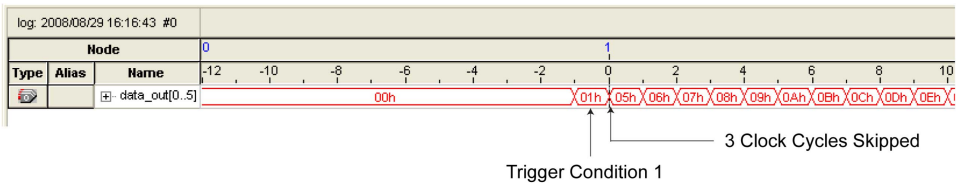


Figure 58. Capture of Data Transaction with Trigger Flow Description Applied

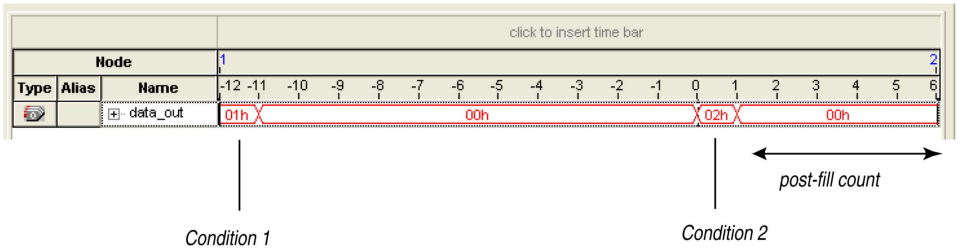


Storage Qualification with Post-Fill Count Value Less than m

The data capture finishes successfully. It uses a buffer with a sample depth of 64, *m* = *n* = *10*, and `post-fill count = 5`.

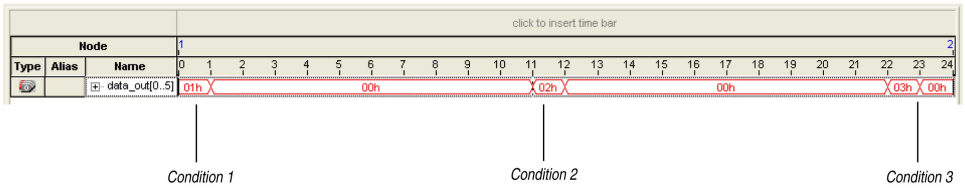
Real data acquisition of the previous scenario

Figure 59. Storage Qualification with Post-Fill Count Value Less than m (Acquisition Successfully Completes)



The combination of using counters, Boolean and relational operators in conjunction with the `start_store` and `stop_store` commands can give a clock-cycle level of resolution to controlling the samples that are written into the acquisition buffer.

Figure 60. Waveform After Forcing the Analysis to Stop



Resource Manipulation Action

The resources the trigger flow description uses can be either counters or status flags.

Table 18. Resource Manipulation Actions

Action	Description	Syntax
`increment`	Increments a counter resource by 1	`increment <counter_identifier>;`
`decrement`	Decrements a counter resource by 1	`decrement <counter_identifier>;`
`reset`	Resets counter resource to initial value	`reset <counter_identifier>;`
`set`	Sets a status flag to 1	`set <register_flag_identifier>;`
`clear`	Sets a status flag to 0	`clear <register_flag_identifier>;`

Buffer Control Actions

Actions that control the acquisition buffer.

Table 19. Buffer Control Actions

Action	Description	Syntax
`trigger`	Stops the acquisition for the current buffer and ends analysis. This command is required in every flow definition.	`trigger <post-fill_count>;`
`segment_trigger`	Available only in segmented acquisition mode. Ends acquisition of the current segment. After evaluating this command, the Signal Tap logic analyzer starts acquiring from the next segment. If all segments are written, the logic analyzer overwrites the oldest segment with the latest sample. When a trigger action is evaluated the acquisition stops.	`segment_trigger <post-fill_count>;`
`start_store`	Active only in state-based storage qualifier mode. Asserts the `write_enable` to the Signal Tap acquisition buffer.	`start_store`
`stop_store`	Active only in state-based storage qualifier mode. De-asserts the `write_enable` signal to the Signal Tap acquisition buffer.	`stop_store`

Both `trigger` and `segment_trigger` actions accept an optional `post-fill_count` argument.

Related Information

Post-fill Count on page 61

State Transition Action

Specifies the next state in the custom state control flow. The syntax is:
`goto` *<state_label>*;

2.4.6.11.6. State-Based Storage Qualifier Feature

Selecting a state-based storage qualifier type enables the `start_store` and `stop_store` actions. When you use these actions in conjunction with the expressions of the State-based trigger flow, you get maximum flexibility to control data written into the acquisition buffer.

Note: You can only apply the `start_store` and `stop_store` commands to a non-segmented buffer.

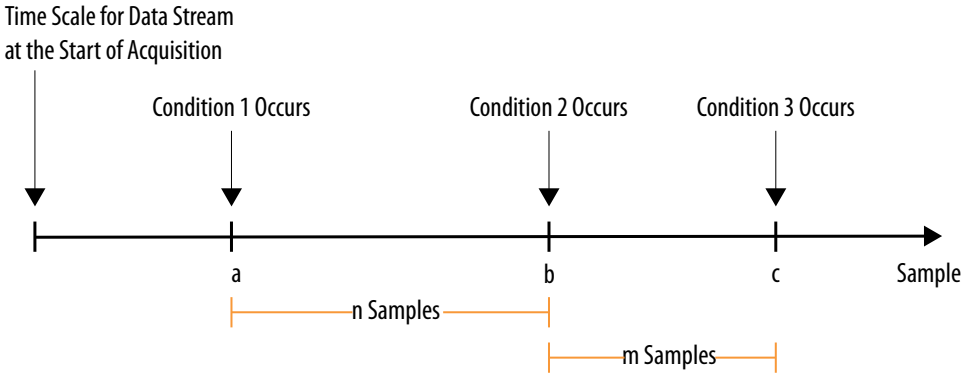
The `start_store` and `stop_store` commands are similar to the start and stop conditions of the **start/stop** storage qualifier mode. If you enable storage qualification, the Signal Tap logic analyzer doesn't write data into the acquisition buffer until the `start_store` command occurs. However, in the state-based storage qualifier type you must include a `trigger` command as part of the trigger flow description. This `trigger` command is necessary to complete the acquisition and display the results on the waveform display.

Storage Qualification Feature for the State-Based Trigger Flow

This trigger flow description contains three trigger conditions that occur at different times after you click **Start Analysis**:

```
State 1: ST1:
    if ( condition1 )
       start_store;
    else if ( condition2 )
       trigger value;
    else if ( condition3 )
       stop_store;
```

Figure 61. Capture Scenario for Storage Qualification with the State-Based Trigger Flow



When you apply the trigger flow to the scenario in the figure:

1. The Signal Tap logic analyzer does not write into the acquisition buffer until **Condition 1** occurs (sample **a**).
2. When **Condition 2** occurs (sample **b**), the logic analyzer evaluates the `trigger value` command, and continues to write into the buffer to finish the acquisition.
3. The trigger flow specifies a `stop_store` command at sample **c**, which occurs `m` samples after the trigger point.
4. If the data acquisition finishes the post-fill acquisition samples before **Condition 3** occurs, the logic analyzer finishes the acquisition and displays the contents of the waveform. In this case, the capture ends if the post-fill count value is < `m`.
5. If the post-fill count value in the Trigger Flow description 1 is > `m` samples, the buffer pauses acquisition indefinitely, provided there is no recurrence of Condition 1 to trigger the logic analyzer to start capturing data again.

The Signal Tap logic analyzer continues to evaluate the `stop_store` and `start_store` commands even after evaluating the trigger. If the acquisition paused, click **Stop Analysis** to manually stop and force the acquisition to trigger. You can use counter values, flags, and the State diagram to help you perform the trigger flow. The counter values, flags, and the current state update in real-time during a data acquisition.

2.4.6.12. Trigger Lock Mode

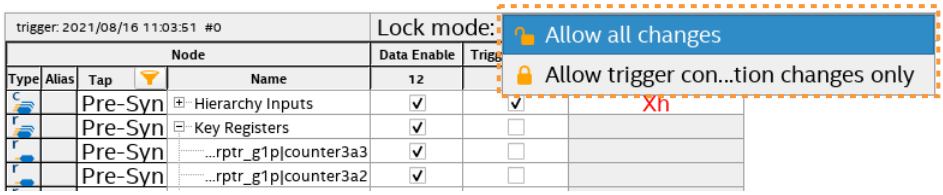
Trigger lock mode restricts changes to only the configuration settings that you specify as **Configurable at runtime**. The runtime configurable settings for the **Custom Trigger Flow** tab are on by default.

Note: You may get some performance advantages by disabling some of the runtime configurable options.

You can restrict changes to your Signal Tap configuration to include only the options that do not require a recompilation. Trigger lock-mode allows you to make changes that reflect immediately in the device.

1. On the **Setup** tab, point to **Lock mode** and select **Allow trigger condition changes only**.

Figure 62. Allow Trigger Conditions Change Only



2. Modify the Trigger Flow conditions.

2.4.7. Specifying Pipeline Settings

The **Pipeline factor** setting indicates the number of pipeline registers that the Intel Quartus Prime software can add to boost the f_{MAX} of the Signal Tap logic analyzer.

To specify the pipeline factor from the Signal Tap GUI:

- In the **Signal Configuration** pane, specify a **pipeline factor** ranging from 0 to 5. The default value is 0.

Note: Setting the pipeline factor does not guarantee an increase in f_{MAX}, as the pipeline registers may not be in the critical paths.

Alternatively, you can specify pipeline parameters as part of HDL instantiation, as Creating a Signal Tap Instance by HDL Instantiation on page 35 describes.

Note: The Signal Tap Intel FPGA IP is not optimized for the Intel Hyperflex architecture.

2.4.8. Filtering Relevant Samples

The Storage Qualifier feature allows you to filter out individual samples not relevant to debugging your design.

The Signal Tap logic analyzer offers a snapshot in time of the data that the acquisition buffers store. By default, the Signal Tap logic analyzer writes into acquisition memory with data samples on every clock cycle. With a non-segmented buffer, there is one data window that represents a comprehensive snapshot of the data stream. Conversely, segmented buffers use several smaller sampling windows spread out over more time, with each sampling window representing a contiguous data set.

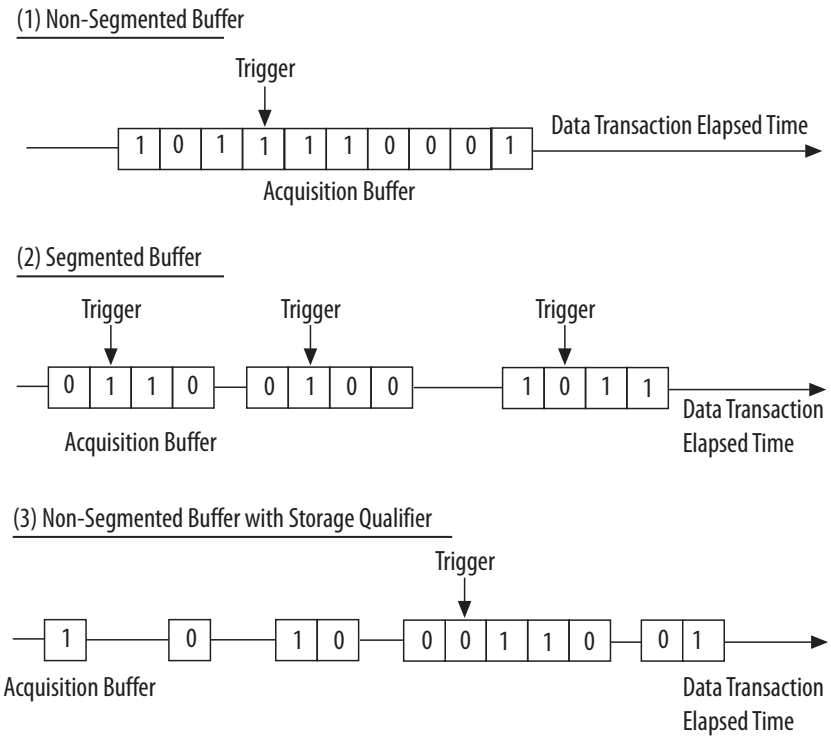
With analysis using acquisition buffers you can capture most functional errors in a chosen signal set, provided adequate trigger conditions and a generous sample depth for the acquisition. However, each data window can have a considerable amount of unnecessary data; for example, long periods of idle signals between data bursts. The default behavior in the Signal Tap logic analyzer doesn't discard the redundant sample bits.

The Storage Qualifier feature allows you to establish a condition that acts as a write enable to the buffer during each clock cycle of data acquisition, thus allowing a more efficient use of acquisition memory over a longer period of analysis.

Because you can create a discontinuity between any two samples in the buffer, the Storage Qualifier feature is equivalent to creating a custom segmented buffer in which the number and size of segment boundaries are adjustable.

Note: You can only use the Storage Qualifier feature with a non-segmented buffer. The IP Catalog flow only supports the Input Port mode for the Storage Qualifier feature.

Figure 63. Data Acquisition Using Different Modes of Controlling the Acquisition Buffer



Notes to figure:

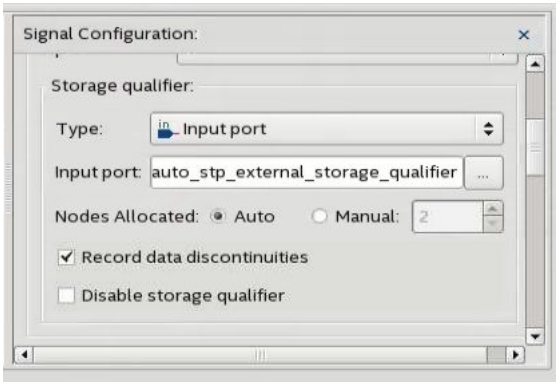
1. Non-segmented buffers capture a fixed sample window of contiguous data.
2. Segmented buffers divide the buffer into fixed sized segments, with each segment having an equal sample depth.
3. Storage Qualifier allows you to define a custom sampling window for each segment you create with a qualifying condition, thus potentially allowing a larger time scale of coverage.

There are six storage qualifier types available under the Storage Qualifier feature:

- **Continuous** (default) Turns the Storage Qualifier off.
- **Input port**
- **Transitional**

- **Conditional**
- **Start/Stop**
- **State-based**

Figure 64. Storage Qualifier Settings



Upon the start of an acquisition, the Signal Tap logic analyzer examines each clock cycle and writes the data into the buffer based upon the storage qualifier type and condition. Acquisition stops when a defined set of trigger conditions occur.

The Signal Tap logic analyzer evaluates trigger conditions independently of storage qualifier conditions.

2.4.8.1. Input Port Mode

When using the Input port mode, the Signal Tap logic analyzer takes any signal from your design as an input. During acquisition, if the signal is high on the clock edge, the Signal Tap logic analyzer stores the data in the buffer. If the signal is low on the clock edge, the logic analyzer ignores the data sample. If you don't specify an internal node, the logic analyzer creates and connects a pin to this input port.

When creating a Signal Tap logic analyzer instance with the Signal Tap logic analyzer GUI, specify the **Storage Qualifier** signal for the **Input port** field located on the **Setup** tab. You must specify this port for your project to compile.

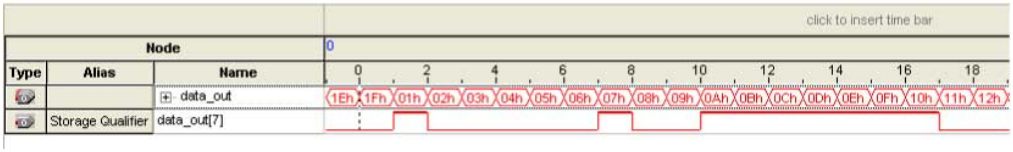
When creating a Signal Tap logic analyzer instance through HDL instantiation, specify the **Storage Qualifier** parameter to include in the instantiation template. You can then connect this port to a signal in your RTL. If you enable the input port storage qualifier, the port accepts a signal and predicates when signals are recorded into the acquisition buffer before or after the specified trigger condition occurs. That is, the trigger you specify is responsible for triggering and moving the logic analyzer into the post-fill state. The input port storage qualifier signal you select controls the recording of samples.

The following example compares and contrasts two waveforms of the same data, one without storage qualifier enabled (**Continuous** means always record samples, effectively no storage qualifier), and the other with **Input Port** mode. The bottom signal in the waveform, `data_out[7]`,is the input port storage qualifier signal. The continuous mode waveform shows 01h, 07h, 0Ah, 0Bh, 0Ch, 0Dh, 0Eh, 0Fh, 10h as the sequence of `data_out[7]` bus values where the storage qualifier signal is asserted. The lower waveform for input port storage qualifier shows how this same

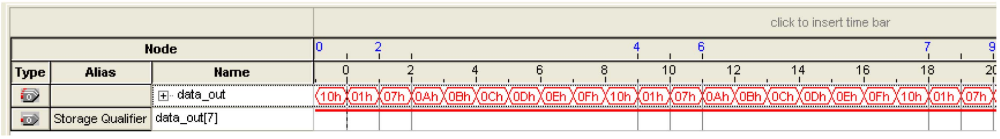
traffic pattern of the `data_out` bus is recorded when you enable the input port storage qualifier. Values recorded are a repeating sequence of the 01h, 07h, 0Ah, 0Bh, 0Ch, 0Dh, 0Eh, 0Fh, 10h (same as **Continuous** mode).

Figure 65. Comparing Continuous and Input Port Capture Mode in Data Acquisition of a Recurring Data Pattern

- **Continuous** Mode:



- **Input Port** Storage Qualifier:



2.4.8.2. Transitional Mode

In **Transitional** mode, the logic analyzer monitors changes in a set of signals, and writes new data in the acquisition buffer only after detecting a change. You select the signals for monitoring using the check boxes in the **Storage Qualifier** column.

Figure 66. Transitional Storage Qualifier Setup

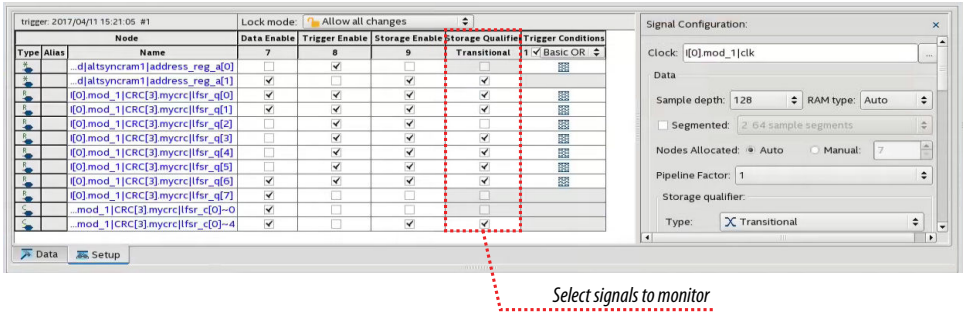
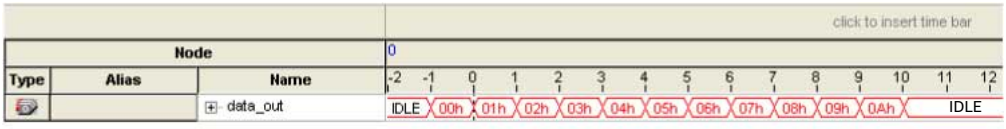
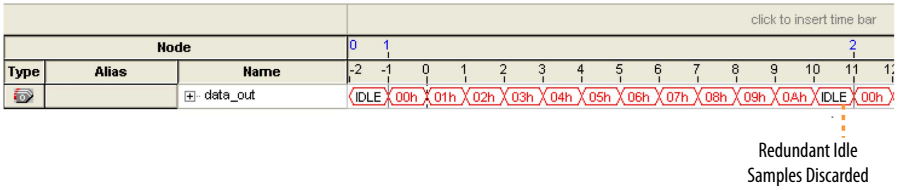


Figure 67. Comparing Continuous and Transitional Capture Mode in Data Acquisition of a Recurring Data Pattern

- **Continuous** mode:



- **Transitional** mode:



2.4.8.3. Conditional Mode

In **Conditional** mode, the Signal Tap logic analyzer determines whether to store a sample by evaluating a combinational function of predefined signals within the node list. The Signal Tap logic analyzer writes into the buffer during the clock cycles in which the condition you specify evaluates TRUE.

You can select either **Basic AND**, **Basic OR**, **Comparison**, or **Advanced** storage qualifier conditions. A **Basic AND** or **Basic OR** condition matches each signal to one of the following:

- **Don't Care**
- **Low**
- **High**
- **Falling Edge**
- **Rising Edge**
- **Either Edge**

If you specify a **Basic AND** storage qualifier condition for more than one signal, the Signal Tap logic analyzer evaluates the logical AND of the conditions.

You can specify any other combinational or relational operators with the enabled signal set for storage qualification through advanced storage conditions.

You can define storage qualification conditions similar to the manner in which you define trigger conditions.

Figure 68. Conditional Storage Qualifier Setup

The figure details the conditional storage qualifier setup in the `.stp` file.

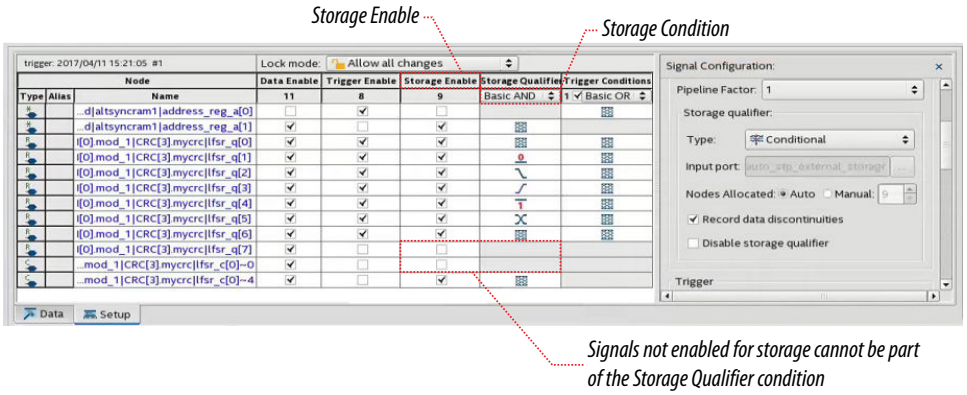
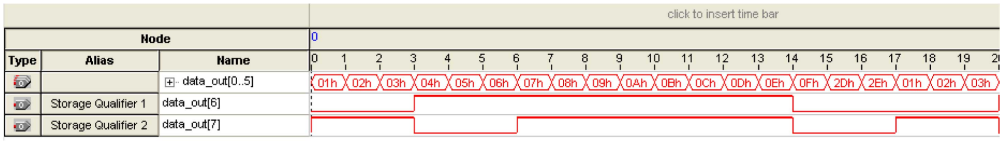


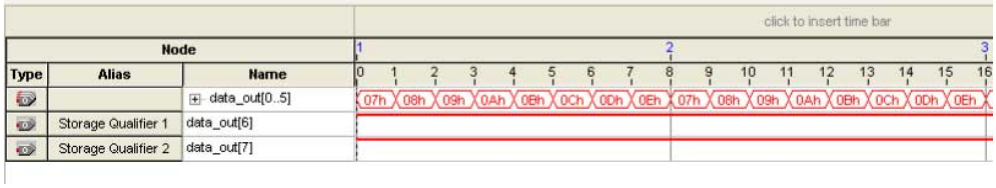
Figure 69. Comparing Continuous and Conditional Capture Mode in Data Acquisition of a Recurring Data Pattern

The data pattern is the same in both cases.

- **Continuous** sampling capture mode:



- **Conditional** sampling capture mode:



Related Information

- Basic Trigger Conditions on page 52
- Comparison Trigger Conditions on page 54
- Advanced Trigger Conditions on page 56

2.4.8.4. Start/Stop Mode

The **Start/Stop** mode uses two sets of conditions, one to start data capture and one to stop data capture. If the start condition evaluates to TRUE, the Signal Tap logic analyzer stores the buffer data every clock cycle until the stop condition evaluates to TRUE, which then pauses the data capture. The logic analyzer ignores additional start signals received after the data capture starts. If both start and stop evaluate to TRUE at the same time, the logic analyzer captures a single cycle.

Note: You can force a trigger by pressing the **Stop** button if the buffer fails to fill to completion due to a stop condition or if the start condition never occurs.

Figure 70. Start/Stop Mode Storage Qualifier Setup

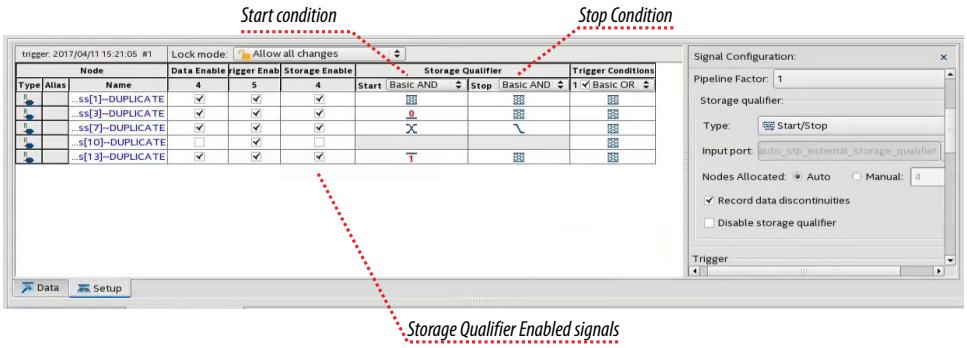
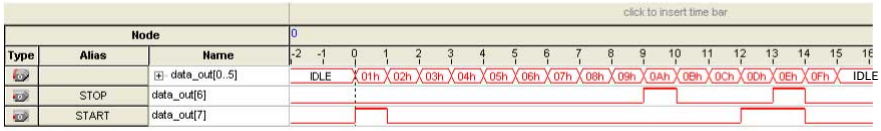
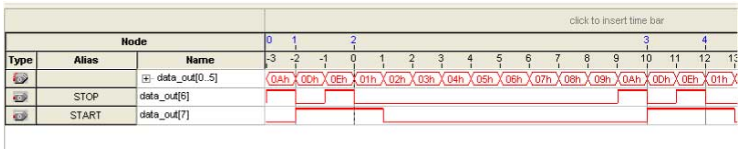


Figure 71. Comparing Continuous and Start/Stop Acquisition Modes for a Recurring Data Pattern

- **Continuous** Mode:



- **Start/Stop** Storage Qualifier:



2.4.8.5. State-Based Mode

The State-based storage qualification mode is part of the State-based triggering flow. The state based triggering flow evaluates a conditional language to define how the Signal Tap logic analyzer writes data into the buffer. With the State-based trigger flow, you have command over boolean and relational operators to guide the execution flow for the target acquisition buffer.

When you enable the storage qualifier feature for the State-based flow, two additional commands become available: `start_store` and `stop_store`. These commands are similar to the Start/Stop capture conditions. Upon the start of acquisition, the Signal Tap logic analyzer doesn't write data into the buffer until a `start_store` action is performed. The `stop_store` command pauses the acquisition. If both `start_store` and `stop_store` actions occur within the same clock cycle, the logic analyzer stores a single sample into the acquisition buffer.

Related Information

State-Based Triggering on page 65

2.4.8.6. Showing Data Discontinuities

When you turn on **Record data discontinuities**, the Signal Tap logic analyzer marks the samples during which the acquisition paused from a storage qualifier. This marker is displayed in the waveform viewer after acquisition completes.

2.4.8.7. Disable the Storage Qualifier

You can disable the storage qualifier with the **Disable Storage Qualifier** option, and then perform a continuous capture. The **Disable Storage Qualifier** option is run-time reconfigurable. Changing the storage qualifier mode from the **Type** field requires recompilation of the project.

2.5. Step 3: Compile the Design and Signal Tap Instances

After you configure one or more Signal Tap instances and define trigger conditions, you must compile your project that includes the Signal Tap logic analyzer, prior to device configuration.

When you define a Signal Tap instance in the logic analyzer GUI or with HDL instantiation, the Signal Tap logic analyzer instance becomes part of your design for compilation.

To run full compilation of the design that includes the Signal Tap logic analyzer instance:

- Click **Processing ➤ Start Compilation**

You can employ various techniques to preserve specific signals for debugging during compilation, and to reduce overall compilation time and iterations. Refer to the following sections for more details.

2.5.1. Recompiling Only Signal Tap Changes

Certain Signal Tap configuration changes require a full recompilation of the design to implement. However, you can use the **Start Recompile** command to implement the following types of configuration changes without running a full design compilation.

Table 20. Signal Tap Configuration Changes Not Requiring Full Compilation

Change the post-fit tap target	Increase the number of post-fit targets
Change the post-fit tap inputs to a Basic AND trigger	Change the post-fit tap inputs to a Basic OR trigger
Change an Advanced trigger (post-fit inputs or logic)	Convert a pre-synthesis tap into a post-fit tap

Start Recompile appends Signal Tap node changes to the existing finalized snapshot, without changing placement and routing outside of the Signal Tap partition.

To recompile Signal Tap configuration changes only, follow these steps:

1. Make supported changes to the Signal Tap configuration in the **Signal Configuration** pane, according to Signal Tap Configuration Changes Not Requiring Full Compilation.
2. In the Signal Tap window, click **Processing ➤ Start Recompile**, or click the **Start Recompile** button. A dialog box displays whether each change is Supported or Unsupported by **Start Recompile**.

Figure 72. Signal Tap Toolbar Start Recompile Button and Command

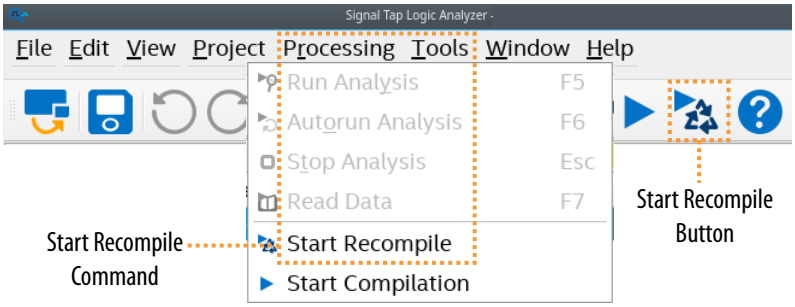
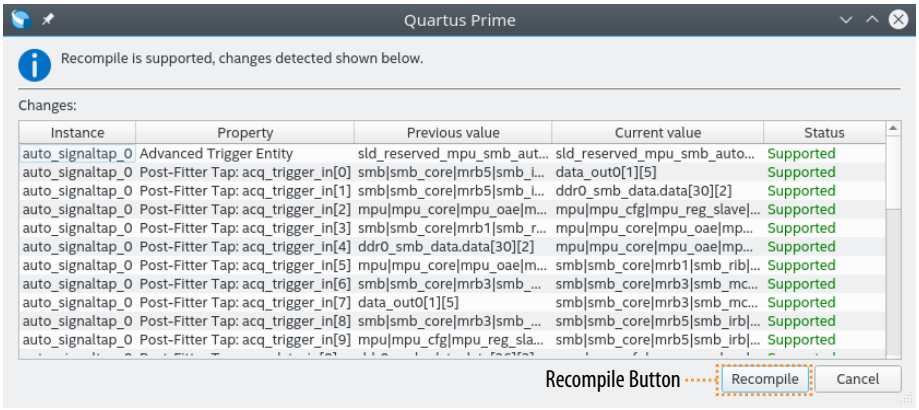
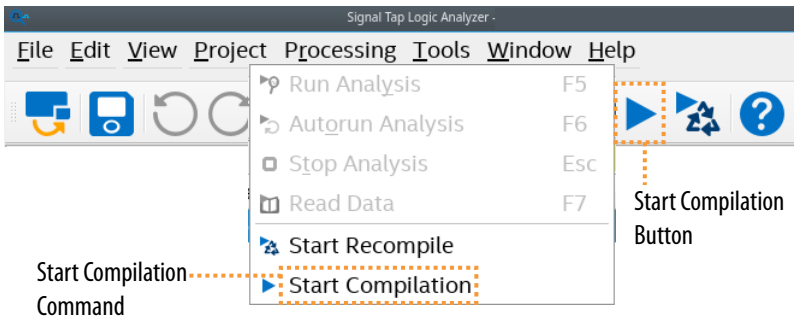


Figure 73. Recompilation Changes List



3. If the Signal Tap configuration changes have a **Status** of Supported, click the **Recompile** button to recompile and implement only the Signal Tap configuration changes, as Recompilation Changes List shows.
4. For any change with **Status** of Unsupported, you must either revert the change to **Previous value**, or click **Processing ➤ Start Compilation** in Signal Tap to perform a full compilation to implement the change.

Figure 74. Signal Tap Toolbar Start Compilation Button and Command



Related Information

Changing the Post-Fit Target Nodes on page 86

2.5.2. Timing Preservation

The following techniques can help you preserve timing in designs that include the Signal Tap logic analyzer:

- Avoid adding critical path signals to the `.stp` file.
- Minimize the number of combinational signals you add to the `.stp` file, and add registers whenever possible.
- Specify an f_{MAX} constraint for each clock in the design.

Related Information

Timing Closure and Optimization
In *Intel Quartus Prime Pro Edition User Guide: Design Optimization*

2.5.3. Performance and Resource Considerations

When you perform logic analysis of your design, you can see the necessary trade-off between runtime flexibility, timing performance, and resource usage. The Signal Tap logic analyzer allows you to select runtime configurable parameters to balance the need for runtime flexibility, speed, and area.

The default values of the runtime configurable parameters provide maximum flexibility, so you can complete debugging as quickly as possible; however, you can adjust these settings to determine whether there is a more appropriate configuration for your design. Because performance results are design-dependent, try these options in different combinations until you achieve the desired balance between functionality, performance, and utilization.

2.5.3.1. Increasing Signal Tap Logic Performance

If Signal Tap logic is part of your critical path, follow these tips to speed up the performance of the Signal Tap logic:

- **Disable runtime configurable options**—runtime flexibility features expend some device resources. If you use Advanced Triggers or State-based triggering flow, disable runtime configurable parameters to a boost in f_{MAX} of the Signal Tap logic. If you use the State-based triggering flow, disable the **Goto state destination** option and perform a recompilation before disabling the other runtime configurable options. The **Goto state destination** option has the greatest impact on f_{MAX}, compared to the other runtime configurable options.
- **Minimize the number of signals that have Trigger Enable selected**—By default, the Signal Tap logic analyzer enables the **Trigger Enable** option for all signals that you add to the `.stp` file. For signals that you do not plan to use as triggers, turn this option off.
- **Turn on Physical Synthesis for register retiming**—If many (more than the number of inputs that fit in a LAB) enabled triggering signals fan-in logic to a gate-based triggering condition (basic trigger condition or a logical reduction operator in the advanced trigger tab), turn on **Perform register retiming**. This can help balance combinational logic across LABs.

2.5.3.2. Reducing Signal Tap Device Resources

If your design has resource constraints, follow these tips to reduce the logic or memory the Signal Tap logic analyzer requires:

- **Disable runtime configurable options**—disabling runtime configurability for advanced trigger conditions or runtime configurable options in the State-based triggering flow results in fewer LEs.
- **Minimize the number of segments in the acquisition buffer**—you can reduce the logic resources that the Signal Tap logic analyzer requires if you limit the segments in your sampling buffer.
- **Disable the Data Enable for signals that you use only for triggering**—by default, the Signal Tap logic analyzer enables **data enable** options for all signals. Turning off the **data enable** option for signals you use only as trigger inputs saves memory resources.

2.6. Step 4: Program the Target Hardware

After you add the Signal Tap logic analyzer instance to your project and fully compile the design, you configure the FPGA target device with your design that includes the Signal Tap logic analyzer instance. You can also program multiple devices with different designs and simultaneously debug them.

When you debug a design with the Signal Tap logic analyzer, you can program a target device directly using the supported JTAG hardware from the Signal Tap window, without using the Intel Quartus Prime Programmer.

Related Information

- Managing Multiple Signal Tap Configurations on page 101
- Intel Quartus Prime Pro Edition User Guide: Programmer

2.6.1. Ensure Compatibility Between .stp and .sof Files

The `.stp` file is compatible with a `.sof` file if the logic analyzer instance parameters, such as the size of the capture buffer and the monitoring and triggering signals, match the programming settings for the target device.

If the files are not compatible, you can still program the device, but you cannot run or control the logic analyzer from the Signal Tap logic analyzer GUI.

Use either of the following methods to ensure compatibility between `.stp` and `.sof` files

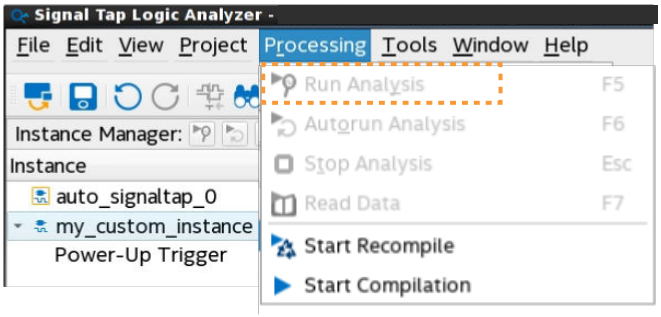
- Attach the .sof file to the .stp file in the SOF Manager. The SOF Manager ensures compatibility between any attached `.sof` files and the current `.stp` file settings automatically, as SOF Manager on page 102 describes.
- To ensure programming compatibility, program the FPGA device with the most recent `.sof` file.

Note: When the Signal Tap logic analyzer detects incompatibility after the analysis starts, the Intel Quartus Prime software generates a system error message containing two CRC values: the expected value and the value retrieved from the `.stp` instance on the device. The CRC value comes from all Signal Tap settings that affect the compilation.

2.7. Step 5: Run the Signal Tap Logic Analyzer

Debugging signals with the Signal Tap logic analyzer GUI is similar to debugging with an external logic analyzer. During normal device operation, you control the logic analyzer through the JTAG connection, specifying the start time for trigger conditions to begin capturing data.

Figure 75. Starting Signal Tap Analysis



1. Select the Signal Tap instance, and then initialize the logic analyzer for that instance by clicking **Processing ➤ Run Analysis** in the Signal Tap logic analyzer GUI.
2. When a trigger event occurs, the logic analyzer stores the captured data in the FPGA device's memory buffer, and then transfers this data to the **Signal Configuration** pane **Data** tab. You can perform the equivalent of a force trigger instruction that allows you to view the captured data currently in the buffer without a trigger event occurring.

You can also use In-System Sources and Probes in conjunction with the Signal Tap logic analyzer to force trigger conditions. The In-System Sources and Probes feature allows you to drive and sample values on to selected signals over the JTAG chain.

2.7.1. Changing the Post-Fit Signal Tap Target Nodes

After performing full compilation of your design and Signal Tap instance, you can subsequently make iterative changes to the post-fit Signal Tap nodes that you want to target, without rerunning full compilation to implement the changes.

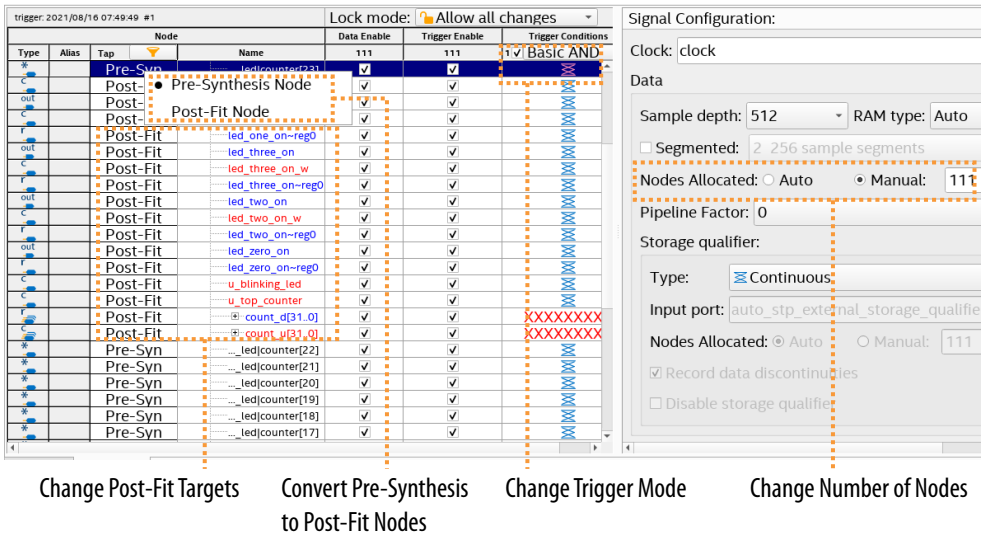
The Signal Tap **Node** list displays whether a target node is **Pre-Syn** (pre-synthesis) or **Post-Fit** in the filterable **Tap** column.

To modify the post-fit Signal Tap nodes:

1. Optionally, mark signals for debug, as Preserving Signals for Monitoring and Debugging describes.

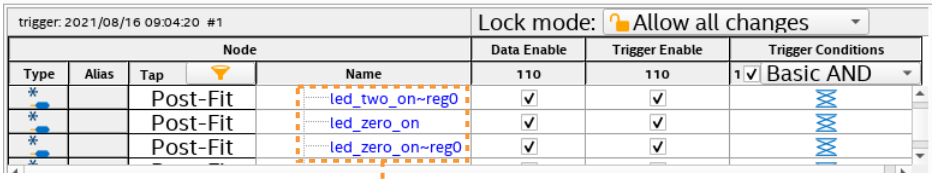
   Note: You cannot change all pre-synthesis nodes to post-fit nodes, unless you are changing the nodes before running full compilation. Once you preserve any signal with `preserve_for_debug`, you can change those preserved pre-synthesis nodes to post-fit nodes.
2. In the **Signal Configuration** pane, modify any of the following properties for nodes with a **Tap** of **Post-Fit**:

Figure 76. Changing the Post-Fit Signal Tap Nodes



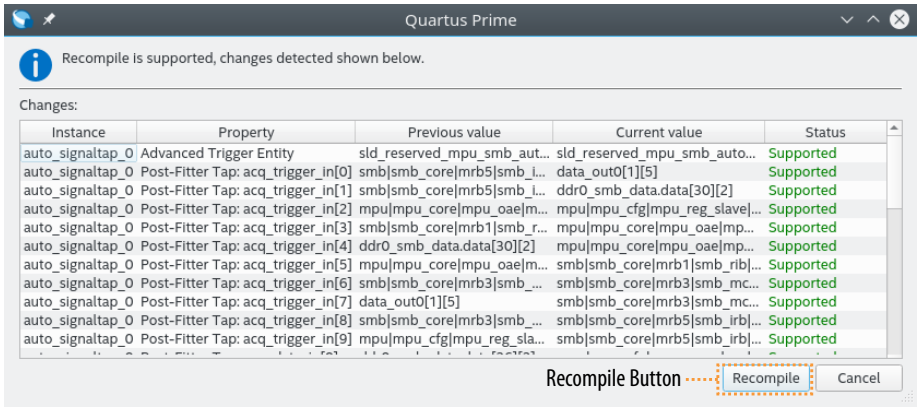
- In the **Name** column, modify or add a new post-fit Signal Tap node target, regardless of the trigger mode.
- In the **Trigger Conditions** column, modify the trigger mode. You can add the post-fit Signal Tap node inputs to a **Basic AND** or **Basic OR** trigger.
- In **Nodes Allocated**, you can specify the **Manual** option to increase or decrease the number of post-fit node targets. You can use manual allocation to help you avoid any major logic change that may require a full recompilation. The data input width affects memory use. The trigger input and storage input width affects the complexity of the condition logic, which can increase the device resource use and the complexity of timing closure.
- Right-click any pre-synthesis Signal Tap node to convert to a post-fit Signal Tap node. The conversion is only successful if Signal Tap can resolve pre-synthesis to post-fit name mapping. Otherwise, the node appears in red and connected to ground. When conversion is successful the post-fit taps names appear in blue text.

Figure 77. Post-fit Taps Names Appear in Blue Text



3. After your post-fit node changes are complete, click **Processing ➤ Start Recompile** to implement only the Signal Tap node changes. A dialog box appears that lists the changes you are implementing, and whether recompilation supports the change.

Figure 78. Recompilation Changes List



4. For any change with **Status** of **Unsupported**, you must either revert the change to **Previous value**, or perform a full compilation to implement the change.
5. Click the **Recompile** button, as Recompilation Changes List shows. Recompilation uses the Engineering Change Order (ECO) compilation flow to append your Signal Tap node changes to the existing finalized snapshot, without changing placement and routing outside the Signal Tap partition.

   Note: The recompilation only applies to the project database if the recompilation is successful. Otherwise, the last successful compilation results remain unchanged.
6. View the changes in the following Compilation Reports following recompilation:

Figure 79. Connections to In-System Debugging Report

Lists each tap target and whether the connection successfully routes (is Connected after recompilation)

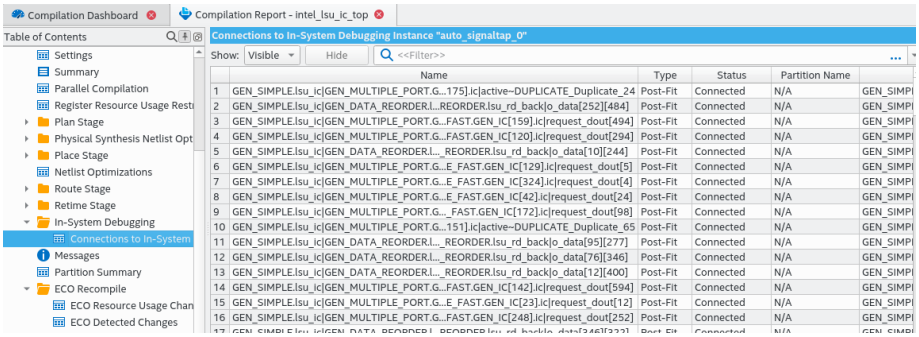


Figure 80. ECO Detected Changes Report

Lists each tap change that you implement with recompilation.

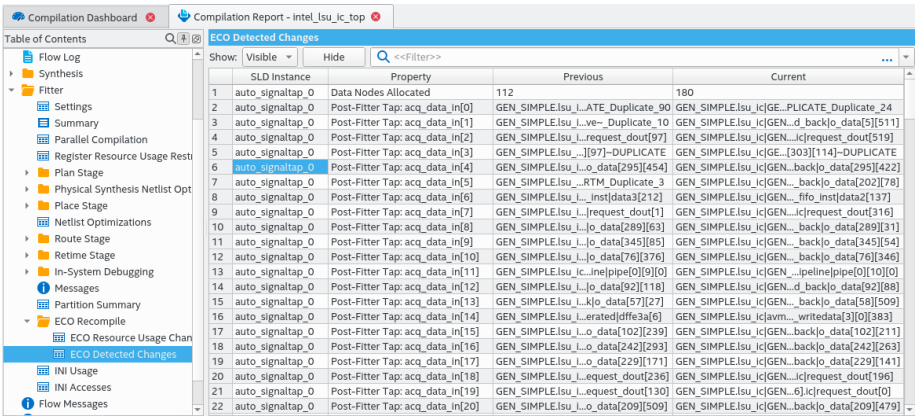
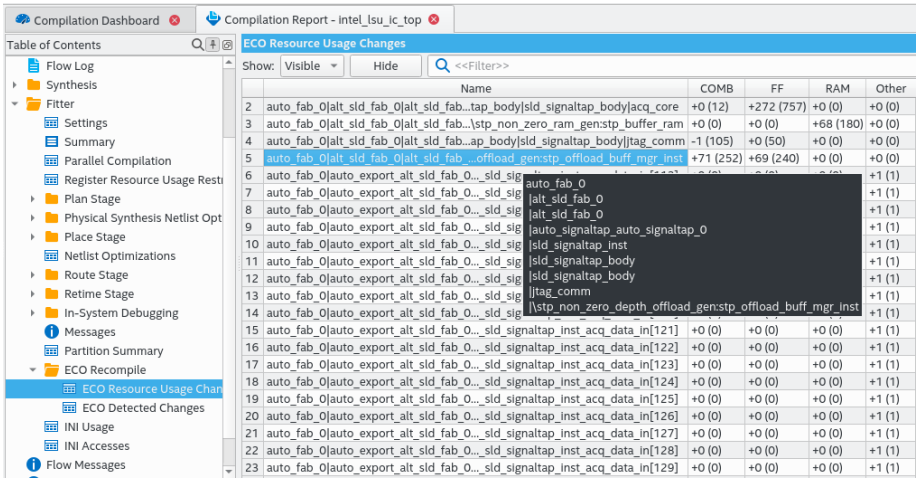


Figure 81. ECO Resource Usage Change

Shows the device resource area change that recompilation implements. Use this report to approximate whether additional changes to the Signal Tap configuration are likely to succeed in combination with the overall design utilization reports.



Related Information

- Recompiling Signal Tap Configuration Changes on page 82
- Using the ECO Compilation Flow in *Intel Quartus Prime Pro Edition User Guide: Design Optimization*

2.7.2. Runtime Reconfigurable Options

When you use Runtime Trigger mode, you can change certain settings in the `.stp` without requiring recompilation of the design.

Table 21. Runtime Reconfigurable Features

Runtime Reconfigurable Setting	Description
Basic Trigger Conditions and Basic Storage Qualifier Conditions	Change without recompiling all signals that have the Trigger condition turned on to any basic trigger condition value
Comparison Trigger Conditions and Comparison Storage Qualifier Conditions	All the comparison operands, the comparison numeric values, and the interval bound values are runtime-configurable. You can also switch from Comparison to **Basic OR** trigger at runtime without recompiling.
Advanced Trigger Conditions and Advanced Storage Qualifier Conditions	Many operators include runtime configurable settings. For example, all comparison operators are runtime-configurable. Configurable settings appear with a white background in the block representation. This runtime reconfigurable option is turned on in the **Object Properties** dialog box.
Switching between a storage-qualified and a continuous acquisition	Within any storage-qualified mode, you can switch to continuous capture mode without recompiling the design. To enable this feature, turn on **disable storage qualifier**.
State-based trigger flow parameters	Refer to *Runtime Reconfigurable Settings, State-Based Triggering Flow*

Runtime Reconfigurable options can save time during the debugging cycle by allowing you to cover a wider possible range of events, without requiring design recompilation. You may experience a slight impact to the performance and logic utilization. You can turn off runtime re-configurability for advanced trigger conditions and the state-based trigger flow parameters, boosting performance and decreasing area utilization.

To configure the `.stp` file to prevent changes that normally require recompilation in the **Setup** tab, select the **Allow Trigger Condition changes only** lock mode above the node list.

This example illustrates a potential use case for Runtime Reconfigurable features, by providing a storage qualified enabled State-based trigger flow description, and showing how to modify the size of a capture window at runtime without a recompile. This example gives you equivalent functionality to a segmented buffer with a single trigger condition where the segment sizes are runtime reconfigurable.

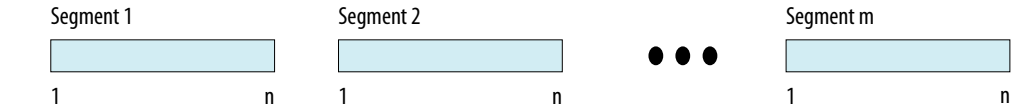
```
state ST1:
if ( condition1 && (c1 <= m) )// each "segment"  triggers on condition // 1
begin                            // m  = number of total "segments"
   start_store;
   increment c1;
   goto ST2:
end

else (c1 > m )                 // This else condition handles the last
                               // segment.
begin
    start_store
    trigger (n-1)
end

state ST2:
if ( c2 >= n)                 //n = number of samples to capture in each
                              //segment.
begin
    reset c2;
    stop_store;
    goto ST1;
end

else (c2 < n)
begin
```

```
    increment c2;
    goto ST2;
end
```

Note: *m* x *n* must equal the sample depth to efficiently use the space in the sample buffer.

The next figure shows the segmented buffer that the trigger flow example describes.

Figure 82. Segmented Buffer Created with Storage Qualifier and State-Based Trigger



During runtime, you can modify the values `m` and `n`. Changing the `m` and `n` values in the trigger flow description adjust the segment boundaries without recompiling.

You can add states into the trigger flow description and selectively mask out specific states and enable other ones at runtime with status flags.

This example is like the previous example with an additional state inserted. You use this extra state to specify a different trigger condition that does not use the storage qualifier feature. You insert status flags into the conditional statements to control the execution of the trigger flow.

```
state ST1 :
    if (condition2  && f1)             // additional state for non-segmented
                                       // acquisition set f1 to enable state
        begin
             start_store;
             trigger
        end
    else if (! f1)
        goto ST2;
state ST2:
    if ( (condition1 && (c1 <= m)  && f2) // f2 status flag used to mask state.
Set f2
                                          // to enable
        begin
           start_store;
           increment c1;
           goto ST3:
        end
    else (c1 > m )
           start_store;
   trigger (n-1)
    end
state ST3:
    if ( c2 >= n)
        begin
            reset c2;
            stop_store;
            goto ST1;
        end
    else (c2 < n)
    begin
        increment c2;
        goto ST2;
    end
```

2.7.3. Signal Tap Status Messages

The following table describes the text messages that might appear in the Signal Tap Status Indicator in the **Instance Manager** pane before, during, or after data acquisition. These messages allow you to monitor the state of the logic analyzer and identify the operation that the logic analyzer is performing.

Table 22. Messages in the Signal Tap Status Indicator

Message	Message Description
Not running	The Signal Tap logic analyzer is not running. This message appears when there is no connection to a device, or the device is not configured.
(Power-Up Trigger) Waiting for clock (1)	The Signal Tap logic analyzer is performing a Runtime or Power-Up Trigger acquisition and is waiting for the clock signal to transition.
Acquiring (Power-Up) pre-trigger data (1)	The trigger condition is not yet evaluated. If the acquisition mode is non-segmented buffer, and the storage qualifier type is continuous, the Signal Tap logic analyzer collects a full buffer of data.
Trigger In conditions met	Trigger In conditions are met. The Signal Tap logic analyzer is waiting for the first trigger condition to occur. This message only appears when a Trigger In condition exists.
Waiting for (Power-up) trigger (1)	The Signal Tap logic analyzer is waiting for the trigger event to occur.
Trigger level <x> met	Trigger condition *x* occurred. The Signal Tap logic analyzer is waiting for condition x + 1 to occur.
Acquiring (power-up) post-trigger data (1)	The entire trigger event occurred. The Signal Tap logic analyzer is acquiring the post-trigger data. You define the amount of post-trigger data to collect (between 12%, 50%, and 88%) when you select the non-segmented buffer acquisition mode.
Offload acquired (Power-Up) data (1)	The JTAG chain is transmitting data to the Intel Quartus Prime software.
Ready to acquire	The Signal Tap logic analyzer is waiting for you to initialize the analyzer.

1. This message can appear for both Runtime and Power-Up Trigger events. When referring to a Power-Up Trigger, the text in parentheses appears.

Note: In segmented acquisition mode, pre-trigger and post-trigger do not apply.

2.8. Step 6: Analyze Signal Tap Captured Data

The Signal Tap logic analyzer GUI allows you to examine the data that you capture manually or with a trigger. In the Data view, you can isolate the data of interest with the drag-to-zoom feature, enabled with a left-click. You can save the data for later analysis, or convert the data to other formats for sharing and further study.

- To simplify reading and interpreting the signal data you capture, set up mnemonic tables, either manually or with a plug-in.
- To speed up debugging, use the **Locate** feature in the **Signal Tap node** list to find the locations of problem nodes in other tools in the Intel Quartus Prime software.

The following topics describe viewing, saving, and exporting Signal Tap analysis captured data:

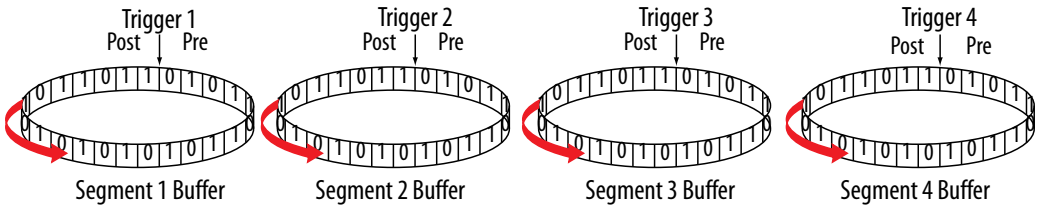
2.8.1. Viewing Capture Data Using Segmented Buffers

Segmented buffers allow you to capture recurring events or sequences of events that span over a long period.

Each acquisition segment acts as a non-segmented buffer, continuously capturing data after activation. When you run analyses with segmented buffers, the Signal Tap logic analyzer captures back-to-back data for each acquisition segment within the data buffer. You define the trigger flow, or the type and order in which the trigger conditions evaluate for each buffer, either in the Sequential trigger flow control or in the Custom State-based trigger flow control.

The following figure shows a segmented acquisition buffer with four segments represented as four separate non-segmented buffers.

Figure 83. Segmented Acquisition Buffer

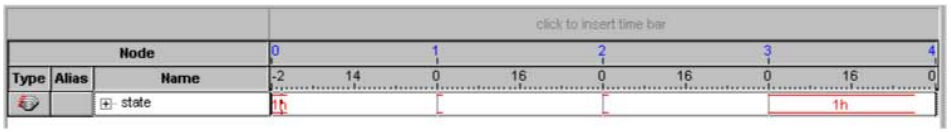


When the Signal Tap logic analyzer finishes an acquisition with a segment and advances to the next segment to start a new acquisition, the data capture that appears in the waveform viewer depends on when a trigger condition occurs. The figure illustrates the data capture method. The Trigger markers—Trigger 1, Trigger 2, Trigger 3 and Trigger 4—refer to the evaluation of the `segment_trigger` and `trigger` commands in the Custom State-based trigger flow. In sequential flows, the Trigger markers for segments 2 through 4 refer to the final trigger condition that you specify within the **Setup** tab.

If the Segment 1 Buffer is the active segment and Trigger 1 occurs, the Signal Tap logic analyzer starts evaluating Trigger 2 immediately. Data Acquisition for the Segment 2 buffer starts when either the Segment 1 Buffer finishes its post-fill count, or when Trigger 2 evaluates as TRUE, whichever condition occurs first. Thus, trigger conditions associated with the next buffer in the data capture sequence can preempt the post-fill count of the current active buffer. This allows the Signal Tap logic analyzer to accurately capture all the trigger conditions that occurred. Unused samples appear as a blank space in the waveform viewer.

Figure 84. Segmented Capture with Preemption of Acquisition Segments

The figure shows a capture using sequential flow control with the trigger condition for each segment specified as **Don't Care**.



Each segment before the last captures only one sample, because the next trigger condition immediately preempts capture of the current buffer. The trigger position for all segments is specified as pre-trigger (12% of the data is before the trigger condition and 88% of the data is after the trigger position). Because the last segment starts immediately with the trigger condition, the segment contains only post-trigger data. The three empty samples in the last segment are left over from the pre-trigger samples that the Signal Tap logic analyzer allocated to the buffer.

For the sequential trigger flow, the **Trigger Position** option applies to every segment in the buffer. A custom state-based trigger flow provides maximum flexibility defining the trigger position. By adjusting the trigger position specific to the debugging requirements, you can help maximize the use of the allocated buffer space.

Related Information

Segmented Buffer on page 45

2.8.2. Viewing Data with Different Acquisition Modes

Different acquisition modes capture different amounts of data immediately after running the Signal Tap logic analyzer and before any trigger conditions occur.

Non-Segmented Buffers in Continuous Mode

In configurations with non-segmented buffers running in continuous mode, the buffer must be full of sampled data before evaluating any trigger condition. Only after the buffer is full, the Signal Tap logic analyzer starts retrieving data through the JTAG connection and evaluates the trigger condition.

If you click the **Stop Analysis** button, Signal Tap prevents the buffer from dumping data during the first acquisition prior to a trigger condition.

Buffers with Storage Qualification

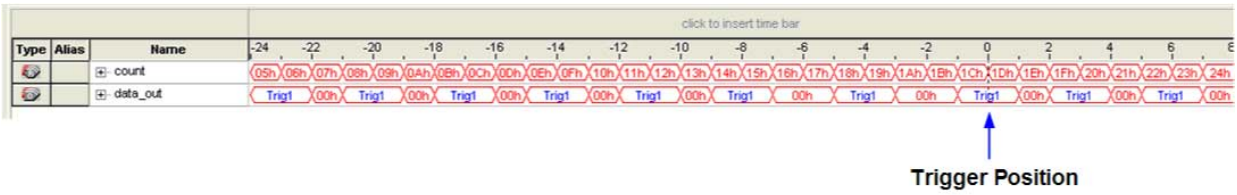
For buffers using a storage qualification mode, the Signal Tap logic analyzer immediately evaluates all trigger conditions while writing samples into the acquisition memory. This evaluation is especially important when using any storage qualification on the data set. The logic analyzer may miss a trigger condition if it waits to capture a full buffer's worth of data before evaluating any trigger conditions.

If a trigger activates before the specified amount of pre-trigger data has occurred, the Signal Tap logic analyzer begins filling memory with post-trigger data, regardless of the amount of pre-trigger data you specify. For example, if you set the trigger position to 50% and set the logic analyzer to trigger on a processor reset, start the logic analyzer, and then power on the target system, the trigger activates. However, the logic analyzer memory contains only post-trigger data, and not any pre-trigger data, because the trigger event has higher precedence than the capture of pre-trigger data.

2.8.2.1. Continuous Mode and a Storage Qualifier Examples

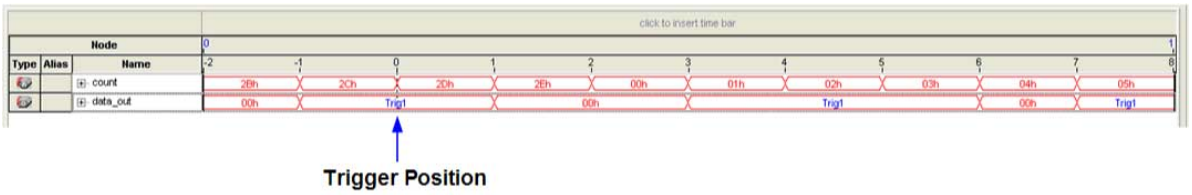
The following show the capture differences between a non-segmented buffer in continuous mode and a non-segmented buffer using a storage qualifier. The configuration of the logic analyzer waveforms is a base trigger condition, sample depth of 64 bits, and **Post trigger position**.

Figure 85. Signal Tap Logic Analyzer Continuous Data Capture



In the continuous data capture, Trig1 occurs several times in the data buffer before the Signal Tap logic analyzer trigger activates. The buffer must be full before the logic analyzer evaluates any trigger condition. After the trigger condition occurs, the logic analyzer continues acquisition for eight additional samples (12% of the buffer, as defined by the "post-trigger" position).

Figure 86. Signal Tap Logic Analyzer Conditional Data Capture



Note to figure:

1. Conditional capture, storage always enabled, post-fill count.
2. The Signal Tap logic analyzer captures a recurring pattern using a non-segmented buffer in conditional mode. The configuration of the logic analyzer is a basic trigger condition "Trig1" and sample depth of 64 bits. The **Trigger in** condition is **Don't care**, so the buffer captures all samples.

In conditional capture the logic analyzer triggers immediately. As in continuous capture, the logic analyzer completes the acquisition with eight samples, or 12% of 64, the sample capacity of the acquisition buffer.

2.8.3. Creating Mnemonics for Bit Patterns

A mnemonic table allows you to assign a meaningful name to a set of bit patterns, such as a bus. To create a mnemonic table:

1. Right-click the **Setup** or **Data** tab of a Signal Tap instance, and click **Mnemonic Table Setup**.
2. Create a mnemonic table by entering sets of bit patterns and specifying a label to represent each pattern.
3. Assign the table to a group of signals by right-clicking the group, clicking **Bus Display Format**, and selecting the mnemonic table.
4. On the **Setup** tab, you can create basic triggers with meaningful names by right-clicking an entry in the **Trigger Conditions** column and selecting a label from the table you assigned to the signal group.

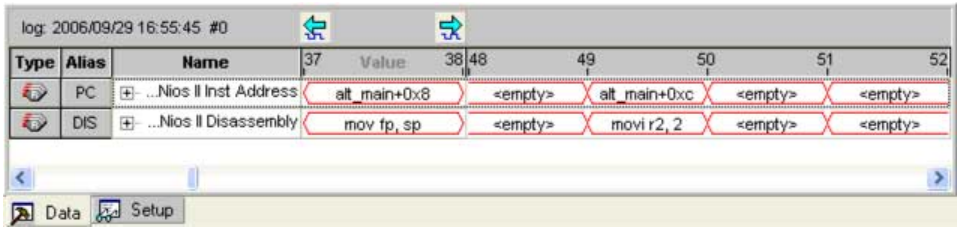
On the **Data** tab, if data captured matches a bit pattern contained in an assigned mnemonic table, the Signal Tap GUI replaces the signal group data with the appropriate label, simplifying the visual inspection of expected data patterns.

2.8.3.1. Adding Mnemonics with a Plug-In

When you use a plug-in to add signals to an `.stp`, mnemonic tables for the added signals are automatically created and assigned to the signals defined in the plug-in. To enable these mnemonic tables manually, right-click the name of the signal or signal group. On the **Bus Display Format** shortcut menu, click the name of the mnemonic table that matches the plug-in.

As an example, the Nios II plug-in helps you to monitor signal activity for your design as the code is executed. If you set up the logic analyzer to trigger on a function name in your Nios II code based on data from an `.elf`, you can see the function name in the **Instruction Address** signal group at the trigger sample, along with the corresponding disassembled code in the **Disassembly** signal group, as shown in Figure 13–52. Captured data samples around the trigger are referenced as offset addresses from the trigger function name.

Figure 87. Data Tab when the Nios II Plug-In is Used



2.8.4. Locating a Node in the Design

When you find the source of an error in your design using the Signal Tap logic analyzer, you can use the node locate feature to locate that signal in various Intel Quartus Prime design visualization tools, as well as in the design file. Locating the node allows you to visualize the source of the problem quickly and correct the issue. To locate a signal from the Signal Tap logic analyzer, right-click the signal in the `.stp`, and click **Locate in ➤ <tool name>**.

You can locate a signal from the node list with the following tools:

- Assignment Editor
- Pin Planner
- Timing Closure Floorplan
- Chip Planner
- Resource Property Editor
- Technology Map Viewer
- RTL Viewer
- Design File

2.8.5. Saving Captured Signal Tap Data

When you save a data capture, the Signal Tap logic analyzer stores this data in the active `.stp` file, and the **Data Log** adds the capture as a log entry under the current configuration.

When you set Signal Tap analysis to **Autorun Analysis**, which starts the Signal Tap logic analyzer in a repetitive acquisition mode, the logic analyzer creates a separate entry in the **Data Log** to store the data captured each time the trigger occurs. This preservation allows you to review the captured data for each trigger event.

The default name for a log derives from the time stamp when the logic analyzer acquires the data. As a best practice, rename the data log with a more meaningful name.

The organization of logs is hierarchical; the logic analyzer groups similar logs of captured data in trigger sets.

Related Information

Data Log Pane on page 101

2.8.6. Exporting Captured Signal Tap Data

You can export captured data to the following file formats, for use with other EDA simulation tools:

- Comma Separated Values File (`.csv`)
- Table File (`.tbl`)
- Value Change Dump File (`.vcd`)
- Vector Waveform File (`.vwf`)
- Graphics format files (`.jpg`, `.bmp`)

To export the captured data from the Signal Tap logic analyzer, click **File ➤ Export**, and then specify the **File Name**, **Export Format**, and **Clock Period**.

2.8.7. Creating a Signal Tap List File

You can generate a Signal Tap list file that contains all the data the logic analyzer captures for a trigger event, in text format.

The Signal Tap list file is especially useful when combined with a plug-in that includes instruction code disassembly. You can view the order of instruction code execution during the same time period of the trigger event.

To create a Signal Tap list file, click **File ➤ Create/Update ➤ Create Signal Tap List File**.

Each row of the list file corresponds to one captured sample in the buffer. Columns correspond to the value of each of the captured signals or signal groups for that sample. If you defined a mnemonic table for the captured data, a matching entry from the table replaces the numerical values in the list.

2.9. Other Signal Tap Debugging Flows

Refer to the following information about more advanced (non-standard) Signal Tap debugging flows and alternative methods.

2.9.1. Signal Tap and Simulator Integration

You can use Signal Tap signal and acquisition data directly in your supported simulator for enhanced visibility into internal signal states in a design hierarchy. The **Add Simulator Aware Nodes** command intelligently analyzes the circuit to determine the minimum set of nodes needed to tap to gain full visibility into the selected hierarchy's cone of logic.

Signal Tap can also transform the Signal Tap data into an RTL simulation testbench for any level of the design hierarchy. This simulation testbench allows you to export acquired Signal Tap hardware data directly into your RTL simulator and observe signal states beyond Signal Tap observability.

The following topics describe these Signal Tap and Simulator Integration features in detail:

- Adding Simulator-Aware Signal Tap Nodes on page 48
- Generating a Simulation Testbench from Signal Tap Data on page 98

Simulator Integration Beta Limitations

This version of the Signal Tap and simulator integration feature is a beta release. The following known limitations apply to this beta release:

- Supports only Verilog HDL simulation.
- Supports testbench generation only within the current project directory.

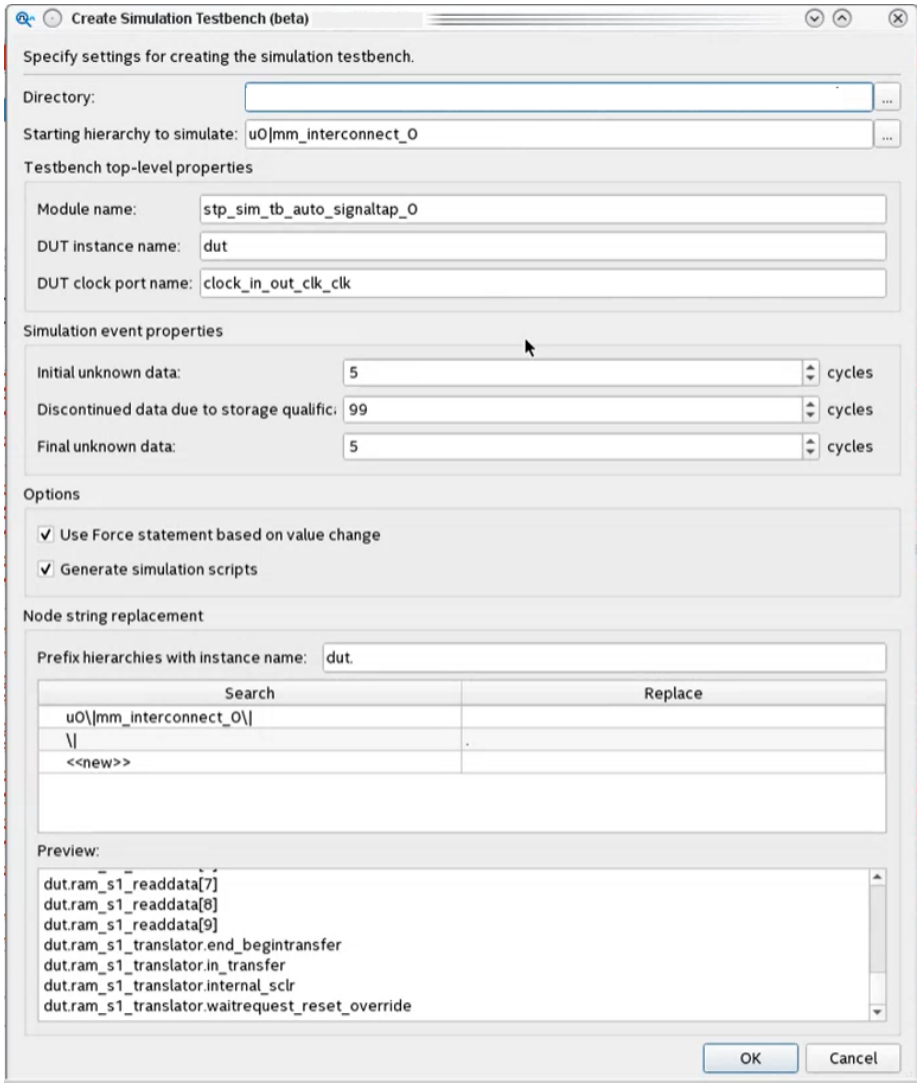
2.9.1.1. Generating a Simulation Testbench from Signal Tap Data

You can use Signal Tap to capture signal data about your running system, and then automatically generate an RTL simulation testbench directly from this capture data for use in your supported simulator.

To generate a simulation testbench from Signal Tap data, follow these steps:

1. Add simulator-aware Signal Tap nodes to the logic analyzer, as Adding Simulator-Aware Signal Tap Nodes on page 48 describes.
2. Run Signal Tap analysis, as Step 5: Run the Signal Tap Logic Analyzer on page 86 describes.

Figure 88. Create Simulation Testbench



3. In the Signal Tap window, click **File ➤ Create Simulation Testbench**. Retain defaults and click **OK**. The testbench generates in a vendor-specific directory. Refer to Create Simulation Testbench Dialog Box Settings on page 100.
4. Source the generated simulator setup script in your supported simulator. For example:

```
source msim_setup.tcl
```

5. Use the commands in the setup script to compile and load the testbench into a supported simulator. For example, in the Questa or ModelSim simulators:

```
ld_debug
```

   Note: Signal Tap uses a Verilog HDL `force` statement to inject the Signal Tap data into the simulator.

6. Add signals to the waveform and run the simulation in your simulator.
7. View the results of simulation in your simulator.

2.9.1.2. Create Simulation Testbench Dialog Box Settings

The following options are available for RTL simulation testbench generation from the Signal Tap **Create Simulation Testbench Dialog Box**. The default values derive from Signal Tap signal data.

Table 23. Create Simulation Testbench Dialog Box (Signal Tap Logic Analyzer)

Name	Description
Directory	Specifies the directory to generated save RTL simulation testbench files. *Note:* Signal Tap currently supports testbench generation only within the current project directory.
Starting hierarchy to simulate	Specifies the design hierarchy level to include in the simulation. The default location is a subdirectory of the project with the hierarchy name.
Testbench top level properties	Specifies the following testbench properties. By default, these values populate from the Signal Tap data: • **Module name**—specifies the name of the design module that you want to simulate, as specified in Signal Tap • **DUT instance name**—specifies the default instance name for the design under test (DUT) in your simulator. The default is **DUT**. This name appears in your simulator. • **DUT clock port name**—specifies the clock port name of the design under test (DUT) for simulation. Signal Tap automatically derives this value based on the **DUT instance name**.
Simulation event properties	Specifies the following testbench properties. By default, these values populate from the Signal Tap data: • **Initial unknown data**—specifies the number of clock cycles for which the data value is initially unknown at the start of simulation. • **Discontinued data due to storage qualification**—specifies the number of clock cycles for which the data is discontinued because of lack of storage. • **Final unknown data**—specifies the number of clock cycles for which the data is unknown initially at the end of simulation.
Options	The following options must be enabled for testbench generation: • **Use force statement based on value change**—specifies the number of clock cycles for which the data value is initially unknown at the start of simulation. *Note:* Signal Tap uses a Verilog HDL `force` statement to inject the Signal Tap data into the simulator. • **Generate simulation scripts**—specifies that simulation scripts generate in vendor specific subdirectories during testbench generation. Source these scripts in your simulator to setup simulation.
Node string replacement	Specifies options for nomenclature and syntax within the generated testbench:

continued...

Name	Description
	• **Prefix hierarchies with instance name**—specifies the instance name that prepends to hierarchy names in the testbench. In general, the derived default value is suitable. • **Search|Replace**—specifies the search and replace strings for **Node string replacement**.
Preview	Displays the result of the **Node string replacement** settings within the testbench.

2.9.2. Managing Multiple Signal Tap Configurations

You can debug different blocks in your design by grouping related monitoring signals. Similarly, you can use a group of signals to define multiple trigger conditions. Each combination of signals, capture settings, and trigger conditions determines a debug configuration, and one configuration can have zero or more associated data logs.

You can save each debug configuration as a different `.stp` file. Alternatively, you can embed multiple configurations within the same `.stp` file, and use the **Data Log** to view and manage each debug configuration.

Note: Each `.stp` pertains to a specific programming (`.sof`) file. To function correctly, the settings in the `.stp` file you use at runtime must match the Signal Tap specifications in the `.sof` file that you use to program the device.

Related Information

Ensure Compatibility Between .stp and .sof Files on page 85

2.9.2.1. Data Log Pane

The **Data Log** pane displays all Signal Tap configurations and data capture results that a single `.stp` file stores.

- To save the current configuration or capture in the **Data Log** of the current `.stp` file, click **Edit ➤ Save to Data Log**.
- To automatically generate a log entry after every data capture, click **Edit ➤ Enable Data Log**. Alternatively, enable the box at the top of the **Data Log** pane.

The **Data Log** displays its contents in a tree hierarchy. The active items display a different icon.

Table 24. Data Log Items

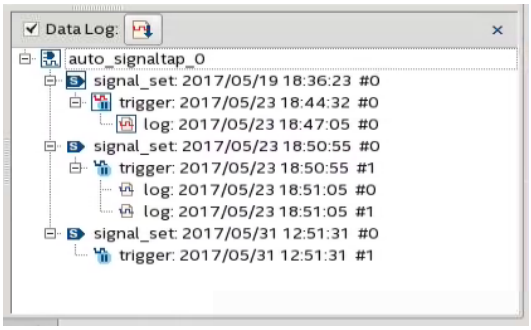
Item	Icon		Contains one or more	Comments
	Unselected	Selected		
Instance			Signal Set	The top-level for a particular Signal Tap instance.
Signal Set			Trigger	The Signal Set changes whenever you add a new signal to a Signal Tap instance. After a change in the Signal Set, you need to recompile.
Trigger			Capture Log	A trigger changes when you change any trigger condition. Some of these changes do not require recompilation.
Capture Log				Contains captured sample data for this particular trigger configuration, for the particular signal set for this particular Signal Tap instance. There can be multiple capture logs for a particular setup if you run the logic analyzer multiple times, as Simple Data Log shows.

The name on each entry displays the wall-clock time when the Signal Tap logic analyzer triggers, and the time elapsed from start acquisition to trigger activation. You can rename entries.

To switch between configurations, double-click an entry in the **Data Log**. As a result, the **Setup** and **Data** tabs update to display the active signal list, trigger conditions, or specified captured data.

Figure 89. Simple Data Log

In this example, the **Data Log** displays one instance with three signal set configurations, two trigger condition setups, and three different captured data sets.



2.9.2.2. SOF Manager

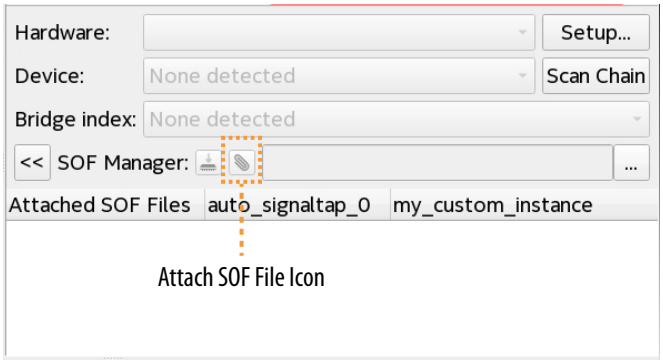
The SOF Manager is in the **JTAG Chain Configuration** pane.

With the SOF Manager you can attach multiple `.sof` files to a single `.stp` file. This attachment allows you to move the `.stp` file to a different location, either on the same computer or across a network, without including the attached `.sof` separately.

The SOF Manager also ensures compatibility between any attached `.sof` files and the current `.stp` file settings automatically, as Ensure Compatibility Between .stp and .sof Files on page 85 describes.

To attach a new `.sof` in the `.stp` file, click the **Attach SOF File** icon .

Figure 90. SOF Manager



As you switch between configurations in the **Data Log**, you can extract the `.sof` that is compatible with that configuration.

To download the new `.sof` to the FPGA, click the **Program Device** icon in the SOF Manager, after ensuring that the configuration of your `.stp` is compatible with the design to program into the target device.

Related Information

Data Log Pane on page 101

2.9.3. Debugging Partial Reconfiguration Designs with Signal Tap

You can debug a Partial Reconfiguration (PR) design with the Signal Tap logic analyzer. The Signal Tap logic analyzer supports data acquisition in the static and PR regions. You can debug multiple personas present in a PR region and multiple PR regions.

For examples on debugging PR designs targeting specific devices, refer to *AN 841: Signal Tap Tutorial for Intel Stratix® 10 Partial Reconfiguration Design* or *AN 845: Signal Tap Tutorial for Intel Arria® 10 Partial Reconfiguration Design*.

Related Information

- AN 841: Signal Tap Tutorial for Intel Stratix 10 Partial Reconfiguration Design
- AN 845: Signal Tap Tutorial for Intel Arria 10 Partial Reconfiguration Design

2.9.3.1. Signal Tap Guidelines for PR Designs

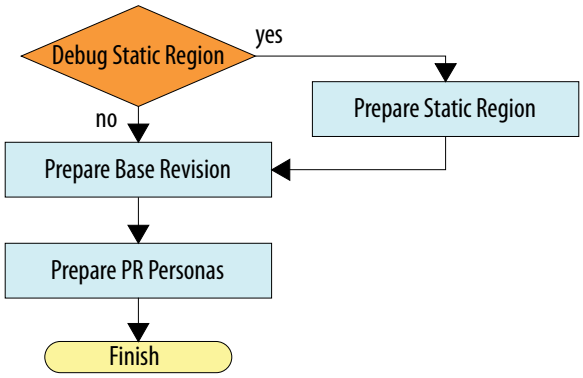
Follow these guidelines to obtain the best results when debugging PR designs with the Signal Tap logic analyzer:

- Include one `.stp` file per project revision.
- Tap pre-synthesis nodes only. In the Node Finder, filter by **Signal Tap: pre-synthesis**.
- Do not tap nodes in the default persona (the personas you use in the base revision compile). Create a new PR implementation revision that instantiates the default persona, and tap nodes in the new revision.

- Store all the tapped nodes from a PR persona in one `.stp` file, to enable debugging the entire persona using only one Signal Tap window.
- Do not tap across PR regions, or from a static region to a PR region in the same `.stp` file.
- Each Signal Tap window opens only one `.stp` file. Therefore, to debug more than one partition simultaneously, you must use stand-alone Signal Tap from the command-line.

2.9.3.2. PR Design Setup for Signal Tap Debug

Figure 91. Setting Up PR Design for Debug with Signal Tap



To debug a PR design, you must instantiate SLD JTAG bridges when generating the base revision, and then define debug components for all PR personas. Optionally, you can specify signals to tap in the static region. After configuring all the PR personas in the design, you can continue the PR design flow.

Related Information

- Debug Fabric for Partial Reconfiguration Designs on page 20
- Partial Reconfiguration Design Flow, Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration

2.9.3.2.1. Preparing the Static Region for Signal Tap Debugging

To debug the static region in your PR design:

1. Tap nodes in the static region exclusively.
2. Save the `.stp` file with a name that identifies the file with the static region.
3. Enable Signal Tap in your project, and include the `.stp` file in the base revision.

Note: Do not tap signals in the default PR personas.

2.9.3.2.2. Preparing the Base Revision for Signal Tap Debugging

In the base revision, for each PR region that you want to debug in the design:

1. Instantiate the SLD JTAG Bridge Agent Intel FPGA IP in the static region.
2. Instantiate the SLD JTAG Bridge Host Intel FPGA IP in the PR region of the default persona.

You can use the IP Catalog or Platform Designer to instantiate SLD JTAG Bridge components.

Related Information

- Instantiating the SLD JTAG Bridge Agent on page 17
- Instantiating the SLD JTAG Bridge Host on page 18

2.9.3.2.3. Preparing PR Personas for Signal Tap Debugging

Before you create revisions for personas in your design, you must instantiate debug IP components and tap signals.

For each PR persona that you want to debug:

1. Instantiate the SLD JTAG Bridge Host Intel FPGA IP in the PR persona.
2. Tap pre-synthesis nodes in the PR persona only.
3. Save in a new `.stp` file with a name that identifies the persona.
4. Use the new `.stp` file in the implementation revision.

If you do not want to debug a particular persona, drive the `tdo` output signal to 0.

2.9.3.3. Performing Data Acquisition in a PR design

After generating the `.sof` and `.rbf` files for the revisions you want to debug, you are ready to program your device and debug with the Signal Tap logic analyzer.

To perform data acquisition:

1. Program the base image into your device.
2. Partially reconfigure the device with the persona that you want to debug.
3. Open the Signal Tap logic analyzer by clicking **Tools ➤ Signal Tap logic analyzer** in the Intel Quartus Prime software.

   The logic analyzer opens and loads the `.stp` file set in the current active revision.
4. To debug other regions in your design, open new Signal Tap windows by opening the other region's `.stp` file from the Intel Quartus Prime main window.

   Alternatively, use the command-line:

   ```
   quartus_stpw <stp_file_other_region.stp>
   ```
5. Debug your design with Signal Tap.

To debug another revision, you must partially reconfigure your design with the corresponding `.rbf` file.

2.9.4. Debugging Block-Based Designs with Signal Tap

The Intel Quartus Prime Pro Edition software supports verification of block-based design flows with the Signal Tap logic analyzer.

Verifying a block-based design requires planning to ensure visibility of logic inside partitions and communication with the Signal Tap logic analyzer. The preparation steps depend on whether you are reusing a core partition or a root partition.

For information about designing with reusable blocks, refer to the *Intel Quartus Prime Pro Edition User Guide: Block-Based Design*. For step-by-step block-based design debugging instructions, refer to *AN 847: Signal Tap Tutorial with Design Block Reuse for Intel Arria 10 FPGA Development Board*.

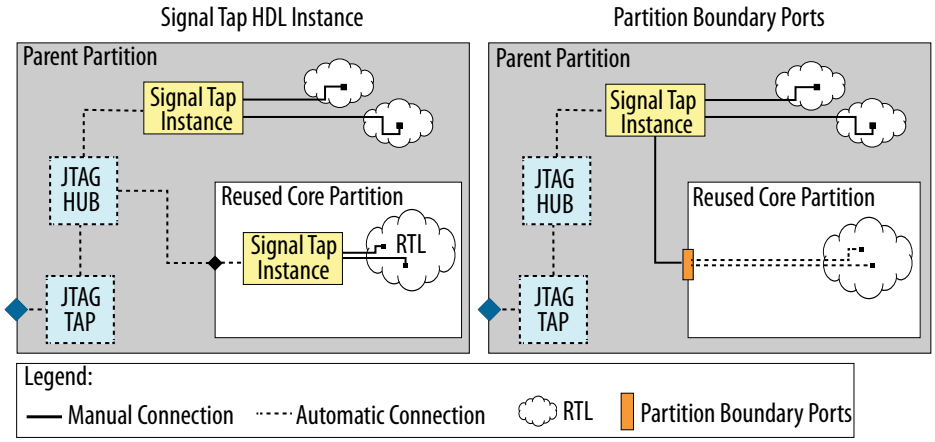
Related Information

Intel Quartus Prime Pro Edition User Guide: Block-Based Design

2.9.4.1. Signal Tap Debugging with a Core Partition

To perform Signal Tap debugging in a core design partition that you reuse from another project, you identify the signals of interest, and then make those signals visible to a Signal Tap logic analyzer instance. The Intel Quartus Prime software supports two methods to make the reused core partition signals visible for Signal Tap monitoring: by creating partition boundary ports, or by Signal Tap HDL instantiation.

Figure 92. Debug Setup with Reused Core Partition



2.9.4.1.1. Partition Boundary Ports Method

Partition boundary ports expose core partition nodes to the top-level partition. Boundary ports simplify the management of hierarchical blocks by tunneling through layers of logic without making RTL changes. The partition boundary ports method includes these high-level steps:

1. In the project that exports the partition, define boundary ports for all potential Signal Tap nodes in the core partition. Define partition boundary ports with the **Create Partition Boundary Ports** assignment in the Assignment Editor. When you assign a bus, the assignment applies to the root name of the debug port, with each bit enumerated.
2. In the project that exports the partition, create a black box file that includes the partition boundary ports, to allows tapping these ports as pre-synthesis or post-fit nodes in another project.
3. In the project that reuses the partition, run Analysis & Synthesis on the reused partition. All valid ports with the **Create Partition Boundary Ports** become visible in the project. After synthesis you can verify the partition boundary ports in the Create Partition Boundary Ports report in the **In-System Debugging** folder under **Synthesis** reports.
4. Tap the partition boundary ports to connect to a Signal Tap instance in the top-level partition. You can also tap logic from the top-level partition to this Signal Tap instance. When using this method, the project requires only one Signal Tap instance to debug both the top-level and the reused core partition.

The following procedures explain these steps in more detail.

2.9.4.1.2. Debug a Core Partition through Partition Boundary Ports

To use Signal Tap to debug a design that includes a core partition exported with partition boundary ports from another project, follow these steps:

1. Add to your project the black-box file that you create in Export a Core Partition with Partition Boundary Ports on page 107.
2. To run synthesis, double-click **Analysis & Synthesis** on the Compilation Dashboard.
3. Define a Signal Tap instance with the Signal Tap GUI, or by instantiating a Signal Tap HDL instance in the top level root partition, as Step 1: Add the Signal Tap Logic Analyzer to the Project on page 34 describes.
4. Connect the partition boundary ports of the reused core partition to the HDL instance, or add post-synthesis or post-fit nodes to the **Signal Configuration** tab in the Signal Tap logic analyzer GUI.
5. To create a design partition, click **Assignments ➤ Design Partitions Window**. Define a partition and assign the exported partition `.qdb` file as the **Partition Database File** option.
6. Compile the design, including all partitions and the Signal Tap instance.
7. Program the Intel FPGA device with the design and Signal Tap instances.
8. Perform data acquisition with the Signal Tap logic analyzer GUI.

2.9.4.1.3. Export a Core Partition with Partition Boundary Ports

To export a core partition with partition boundary ports for reuse and Signal Tap debugging in another project, follow these steps:

1. To run synthesis, double-click **Analysis & Synthesis** on the Compilation Dashboard.
2. Define a design partition for reuse that contains only core logic. Click **Assignments ➤ Design Partitions Window** to define the partition.
3. To create partition boundary ports for the core partition, specify the **Create Partition Boundary Ports** assignment in the Assignment Editor for partition ports.
4. Click **Project ➤ Export Design Partition**. By default, the `.qdb` file you export includes any Signal Tap HDL instances for the partition.
5. Compile the design and Signal Tap instance.
6. Create a black box file that defines only the port and module or entity definitions, without any logic.
7. Manually copy the exported partition `.qdb` file and any black box file to the other project.

Optionally, you can verify signals in the root and core partitions in the Developer project with the Signal Tap logic analyzer.

2.9.4.1.4. Signal Tap HDL Instance Method

To use the Signal Tap HDL instance method, you first create a Signal Tap HDL instance in the reusable core partition, and then connect the signals of interest to that instance. The Compiler ensures top-level visibility of Signal Tap instances inside partitions. Since the root partition and the core partition have separated HDL instances, the Signal Tap files are also separate.

When you reuse the partition in another project, you must generate one Signal Tap file in the target project for each HDL instance present in the reused partition.

Debug a Core Partition Exported with Signal Tap HDL Instances

To use Signal Tap to debug a design that includes a core partition exported with Signal Tap HDL instances, follow these steps:

1. Add to your project the black-box file that you create in Export a Core Partition with Signal Tap HDL Instances on page 109.
2. To create a design partition, click **Assignments ➤ Design Partitions Window**. Define a partition and assign the exported partition `.qdb` file as the **Partition Database File** option.
3. Create a Signal Tap file for the top-level partition as Step 1: Add the Signal Tap Logic Analyzer to the Project on page 34 describes.
4. Compile the design and Signal Tap instances.
5. Generate a Signal Tap file for the reused Core Partition with the **File ➤ Create/Update ➤ Create Signal Tap File from Design Instance** command.
6. Program the Intel FPGA device with the design and Signal Tap instances.
7. Perform hardware verification of top-level partition with the Signal Tap instance defined in Step 3.
8. Perform hardware verification of the Reused Core Partition with the Signal Tap instance defined in Step 5.

2.9.4.1.5. Export a Core Partition with Signal Tap HDL Instances

To export a core partition with Signal Tap HDL instances for reuse and eventual Signal Tap debugging in another project, follow these steps:

1. To run synthesis, double-click **Analysis & Synthesis** on the Compilation Dashboard.
2. Define a design partition for reuse that contains only core logic. Click **Assignments ➤ Design Partitions Window** to define the partition.
3. Add a Signal Tap HDL instance to the core partition, connecting it to nodes of interest.
4. Click **Project ➤ Export Design Partition**. By default, the `.qdb` file you export includes any Signal Tap HDL instances for the partition.
5. Create a black box file that defines only the port and module or entity definitions, without any logic.
6. Manually copy the exported partition `.qdb` file and any black box file to the other project.

2.9.4.1.6. Debug a Core Partition Exported with Signal Tap HDL Instances

To use Signal Tap to debug a design that includes a core partition exported with Signal Tap HDL instances, follow these steps:

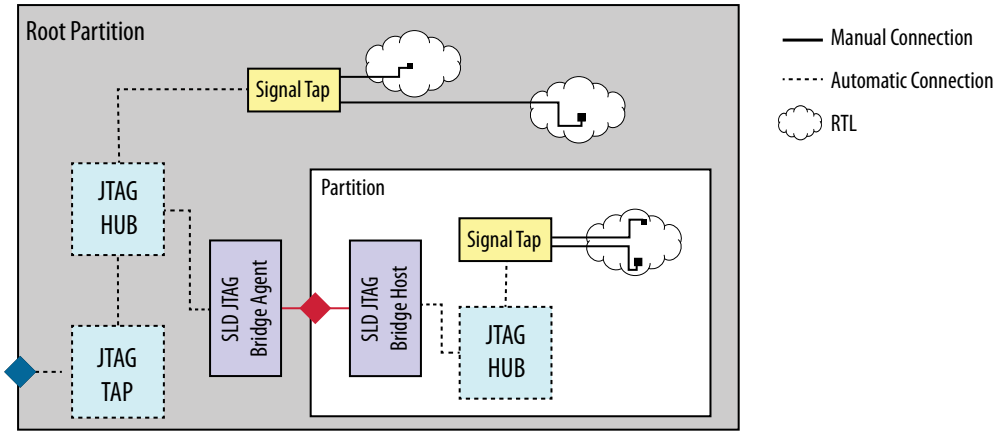
1. Add to your project the black-box file that you create in Export a Core Partition with Signal Tap HDL Instances on page 109.
2. To create a design partition, click **Assignments ➤ Design Partitions Window**. Define a partition and assign the exported partition `.qdb` file as the **Partition Database File** option.
3. Create a Signal Tap file for the top-level partition as Step 1: Add the Signal Tap Logic Analyzer to the Project on page 34 describes.
4. Compile the design and Signal Tap instances.
5. Generate a Signal Tap file for the reused Core Partition with the **File ➤ Create/Update ➤ Create Signal Tap File from Design Instance** command.
6. Program the Intel FPGA device with the design and Signal Tap instances.
7. Perform hardware verification of top-level partition with the Signal Tap instance defined in Step 3.
8. Perform hardware verification of the Reused Core Partition with the Signal Tap instance defined in Step 5.

2.9.4.2. Signal Tap Debugging with a Root Partition

In a project that reuses a root partition, you enable debugging of the root partition and the core partition independently, with separate Signal Tap instances in each partition. In the project that exports the partition, you add the Signal Tap instance to the root partition. Additionally, you extend the debug fabric into the reserved core partition with a debug bridge. This bridge allows subsequent instantiation of Signal Tap when reusing the partition in another project.

You implement the debug bridge with the SLD JTAG Bridge Agent Intel FPGA IP and SLD JTAG Bridge Host Intel FPGA IP pair for each reserved core boundary in the design. You instantiate the SLD JTAG Bridge Agent IP in the root partition, and the SLD JTAG Bridge Host IP in the core partition.

Figure 93. Debug Setup with Reused Root Partition



For details about the debug bridge, refer to the *SLD JTAG Bridge* in the *System Debugging Tools Overview* chapter.

Related Information

SLD JTAG Bridge on page 14

2.9.4.2.1. Export the Root Partition with SLD JTAG Bridge

To export a reusable root partition with SLD JTAG Bridge that allows debugging of core partitions in another project, follow these steps.

1. Create a reserved core partition and define a Logic Lock region.
2. Generate and instantiate SLD JTAG Bridge Agent in the root partition.

   The combination of agent and host allows debugging the reserved core partition in Consumer projects.
3. Generate and instantiate the SLD JTAG Bridge Host in the reserved core partition.
4. Add a Signal Tap instance to the root partition, as Step 1: Add the Signal Tap Logic Analyzer to the Project on page 34 describes.
5. In the Signal Tap instance, specify the signals for monitoring. This action allows debugging the root partition in the Developer and Consumer projects.
6. Compile the design and Signal Tap instance.
7. Click **Project ➤ Export Design Partition**. By default, the `.qdb` file you export includes any Signal Tap HDL instances for the partition.
8. Manually copy files to the project that reuses the root partition:
   - In designs targeting the Intel Arria 10 device family, copy `.qdb` and `.sdc` files.
   - In designs targeting the Intel Stratix 10 device family copy the `.qdb` file.

In designs with multiple child partitions, you must provide the hierarchy path and the associated index of the JTAG Bridge Instance Agents in the design to the Consumer.

2.9.4.2.2. Debugging an Exported Root Partition and Core Partition Simultaneously using the SLD JTAG Bridge

When you reuse an exported root partition in another project, the exported `.qdb` includes the Signal Tap connection to signals in the root partition, and the SLD JTAG Bridge Agent IP, which allows debugging logic in the core partition.

To perform Signal Tap debugging in a project that includes a reused root partition:

1. Add the exported `.qdb` (and `.sdc`) files to the project that reuses them.
2. From the IP Catalog, parameterize and instantiate the SLD JTAG Bridge Host Intel FPGA IP in the core partition.
3. Run the Analysis & Synthesis stage of the Compiler.
4. Create a Signal Tap instance in the core partition, as Step 1: Add the Signal Tap Logic Analyzer to the Project on page 34 describes.
5. In the Signal Tap instance, specify post-synthesis signals for monitoring.

   Note: You can only tap signals in the core partition.
6. Compile the design and Signal Tap instance.
7. Generate a Signal Tap file for the reused root partition with the `quartus_stp` command.
8. Program the device.
9. Perform hardware verification of the reserved core partition with the Signal Tap instance defined in Step 3.
10. Perform hardware verification of the reused root partition with the Signal Tap instance defined in Step 7.

2.9.4.3. Compiler Snapshots and Signal Tap Debugging

When you reuse a design partition exported from another project, the design partition preserves the results of a specific snapshot of the compilation. Whenever possible, it is easiest to specify the signals for monitoring in the original project that exports the partition.

Adding new signals to a Signal Tap instance in a reused partition requires the Fitter to connect and route these signals. This is only possible when:

- The reused partition contains the Synthesis snapshot—reused partitions that contain the Placed or Final snapshot do not allow adding more signals to the Signal Tap instance for monitoring, because you cannot create additional boundary ports.
- The signal that you want to tap is a post-fit signal—adding pre-synthesis Signal Tap signals is not possible, because that requires resynthesis of the partition.

Related Information

Signals Unavailable for Signal Tap Debugging on page 51

2.9.4.3.1. Add Post-Fit Nodes when Reusing a Partition Containing a Synthesis Snapshot

You can add post-fit nodes for Signal Tap debug when reusing a design partition containing the synthesis snapshot exported from another project.

To add post-fit nodes to Signal Tap for monitoring:

1. Open the project that reuses the partition, and then compile the reused partition through the Fitter stage.
2. Add a Signal Tap instance to the project that reuses the partition, as Step 1: Add the Signal Tap Logic Analyzer to the Project on page 34 describes.
3. In the Signal Tap GUI, add the post-fit Signal Tap nodes to the **Signal Configuration** tab.
4. Recompile the design from the Place stage by clicking **Processing ➤ Start ➤ Start Fitter (Place)**.

   The Fitter attaches the Signal Tap nodes to the existing synthesized nodes.

2.9.5. Debugging Devices that use Configuration Bitstream Security

Some Intel FPGA device families support bitstream decryption during configuration using an on-device AES decryption engine. You can still use the Signal Tap logic analyzer to analyze functional data within the FPGA with such devices. However, JTAG configuration is not possible after programming the security key into the device.

Use an unencrypted bitstream during the prototype and debugging phases of the design, to allow programming file generation and reconfiguration of the device over the JTAG connection while debugging.

If you must use the Signal Tap logic analyzer with an encrypted bitstream, first configure the device with an encrypted configuration file using Passive Serial (PS), Fast Passive Parallel (FPP), or Active Serial (AS) configuration modes. The design must contain at least one instance of the Signal Tap logic analyzer. After configuring the FPGA with a Signal Tap instance and the design, you can open the Signal Tap logic analyzer GUI and scan the chain to acquire data with the JTAG connection.

Related Information

Intel Quartus Prime Pro Edition User Guide: Programmer

2.9.6. Signal Tap Data Capture with the MATLAB MEX Function

When you use MATLAB for DSP design, you can acquire data from the Signal Tap logic analyzer directly into a matrix in the MATLAB environment. To use this method, you call the MATLAB MEX function, `alt_signaltap_run`, that the Intel Quartus Prime software includes. If you use the MATLAB MEX function in a loop, you can perform as many acquisitions in the same amount of time as when using Signal Tap in the Intel Quartus Prime software environment.

Note: The MATLAB MEX function for Signal Tap is available in the Windows* version and Linux version of the Intel Quartus Prime software. This function is compatible with MATLAB Release 14 Original Release Version 7 and later.

To set up the Intel Quartus Prime software and the MATLAB environment to perform Signal Tap acquisitions:

1. In the Intel Quartus Prime software, create an `.stp` file.
2. In the node list in the **Data** tab of the Signal Tap logic analyzer Editor, organize the signals and groups of signals into the order in which you want them to appear in the MATLAB matrix.

   Each column of the imported matrix represents a single Signal Tap acquisition sample, while each row represents a signal or group of signals in the order you defined in the **Data** tab.

   Note: Signal groups that the Signal Tap logic analyzer acquires and transfers into the MATLAB MEX function have a width limit of 32 signals. To use the MATLAB MEX function with a bus or signal group that contains more than 32 signals, split the group into smaller groups that do not exceed the limit.
3. Save the `.stp` file and compile your design. Program your device and run the Signal Tap logic analyzer to ensure your trigger conditions and signal acquisition work correctly.
4. In the MATLAB environment, add the Intel Quartus Prime binary directory to your path with the following command:

   ```
   addpath <Quartus install directory>\win
   ```

   You can view the help file for the MEX function by entering the following command in MATLAB without any operators:

   ```
   alt_signaltap_run
   ```
5. Use the MATLAB MEX function to open the JTAG connection to the device and run the Signal Tap logic analyzer to acquire data. When you finish acquiring data, close the JTAG connection.

   To open the JTAG connection and begin acquiring captured data directly into a MATLAB matrix called `stp`, use the following command:

   ```
   stp = alt_signaltap_run \
   ('<stp filename>'[,('signed'|'unsigned')[,'<instance names>'[, \
   '<signalset name>'[,'<trigger name>']]]]);
   ```

   When capturing data, you must assign a filename, for example, *<stp filename>* as a requirement of the MATLAB MEX function. The following table describes other MATLAB MEX function options:

Table 25. Signal Tap MATLAB MEX Function Options

Option	Usage	Description
`signed` `unsigned`	`'signed'` `'unsigned'`	The **signed** option turns signal group data into 32-bit two's-complement signed integers. The MSB of the group as defined in the Signal Tap **Data** tab is the sign bit. The **unsigned** option keeps the data as an unsigned integer. The default is **signed**.
<instance name>	`'auto_signaltap_0'`	Specify a Signal Tap instance if more than one instance is defined. The default is the first instance in the `.stp`, `auto_signaltap_0`.
<signal set name> *<trigger name>*	`'my_signalset'` `'my_trigger'`	Specify the signal set and trigger from the Signal Tap data log if multiple configurations are present in the `.stp`. The default is the active signal set and trigger in the file.

During data acquisition, you can enable or disable verbose mode to see the status of the logic analyzer. To enable or disable verbose mode, use the following commands:

```
alt_signaltap_run('VERBOSE_ON');-alt_signaltap_run('VERBOSE_OFF');
```

When you finish acquiring data, close the JTAG connection with the following command:

```
alt_signaltap_run('END_CONNECTION');
```

For more information about the use of MATLAB MEX functions in MATLAB, refer to the MATLAB Help.

2.10. Signal Tap Logic Analyzer Design Examples

Application Note 845: Signal Tap Tutorial for Intel Arria 10 Partial Reconfiguration Design includes a design example that demonstrates Signal Tap debugging with a partial reconfiguration design. The design example has one 32-bit counter. At the board level, the design connects the clock to a 50MHz source, and connects the output to four LEDs on the FPGA. Selecting the output from the counter bits in a specific sequence causes the LEDs to blink at a specific frequency example demonstrates initiating a DMA transfer. The tutorial demonstrates how to tap signals in a PR design by extending the debug fabric to the PR regions when creating the base revision, and then defining debug components in the implementation revisions.

Application Note 446: Debugging Nios II Systems with the Signal Tap Logic Analyzer includes a design example with a Nios II processor, a direct memory access (DMA) controller, on-chip memory, and an interface to external SDRAM memory. After you press a button, the processor initiates a DMA transfer, which you analyze using the Signal Tap logic analyzer. In this example, the Nios II processor executes a simple C program from on-chip memory and waits for you to press a button.

Related Information

- AN 845: Signal Tap Tutorial for Intel Arria 10 Partial Reconfiguration Design
- AN 446: Debugging Nios II Systems with the Signal Tap Logic Analyzer

2.11. Custom State-Based Triggering Flow Examples

The custom state-based triggering flow in the Signal Tap logic analyzer GUI can organize multiple triggering conditions for precise control over the acquisition buffer. The following examples demonstrate defining a custom triggering flow. You can easily copy the examples directly into the state machine description box by specifying the **All states in one window** option.

Related Information

On-chip Debugging Design Examples website

2.11.1. Trigger Example 1: Custom Trigger Position

Actions to the acquisition buffer can accept an optional post-count argument. This post-count argument enables you to define a custom triggering position for each segment in the acquisition buffer.

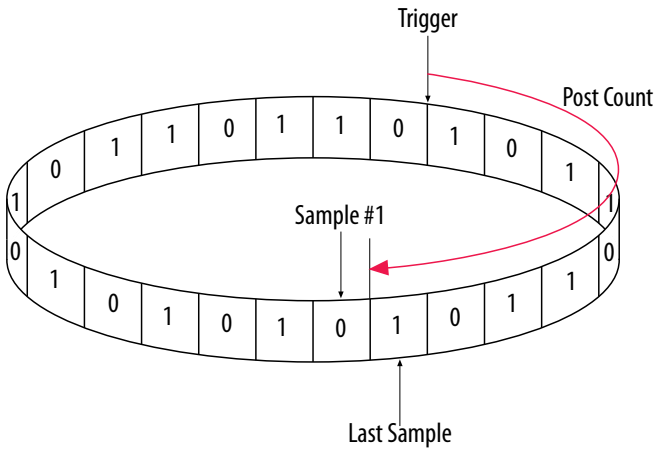
The following example shows how to apply a trigger position to all segments in the acquisition buffer. The example describes a triggering flow for an acquisition buffer split into four segments. If each acquisition segment is 64 samples in depth, the trigger position for each buffer is at sample #34. The acquisition stops after all segments are filled once.

```
if (c1 == 3 && condition1)
    trigger 30;
else if ( condition1 )
begin
    segment_trigger 30;
    increment c1;
end
```

Each segment acts as a non-segmented buffer that continuously updates the memory contents with the signal values.

The **Data** tab displays the last acquisition before stopping the buffer as the last sample number in the affected segment. The trigger position in the affected segment is then defined by *N* – `post count fill`, where *N* is the number of samples per segment.

Figure 94. Specifying a Custom Trigger Position



2.11.2. Trigger Example 2: Trigger When triggercond1 Occurs Ten Times between triggercond2 and triggercond3

You can use a custom trigger flow to count a sequence of events before triggering the acquisition buffer, as the following example shows. This example uses three basic triggering conditions configured in the Signal Tap **Setup** tab.

This example triggers the acquisition buffer when `condition1` occurs after `condition3` and occurs ten times prior to `condition3`. If `condition3` occurs prior to ten repetitions of `condition1`, the state machine transitions to a permanent wait state.

```
state ST1:
if ( condition2  )
begin
   reset c1;
   goto ST2;
end
```

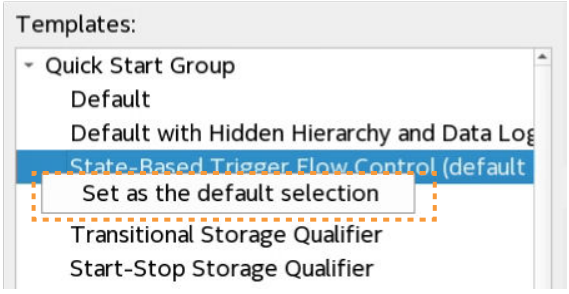
```
State ST2 :
if ( condition1 )
    increment c1;
else if (condition3 && c1 < 10)
    goto ST3;
else if ( condition3 && c1 >= 10)
    trigger;
ST3:
goto ST3;
```

2.12. Signal Tap File Templates

Signal Tap file templates provide preset settings for various trigger conditions, interfaces, and state-based triggering flows. The following Signal Tap file templates are available whenever you open a new Signal Tap session or create a new `.stp` file.

Right-click any template in the **New File from Template** dialog box, and then click **Set as the default selection** to always open new `.stp` files in that template by default.

Figure 95. Settings Default Signal Tap File Template



Note: Refer to the **New File from Template** dialog box for complete descriptions of all templates.

Table 26. Quick Start Signal Tap File Templates

Template	Summary Description
Default	The most basic and compact setup that is suitable for many debugging needs
Default with Hidden Hierarchy and Data Log	The same setup as the Default template, with additional Hierarchy Display and Data Log windows for trigger condition setup.
State-Based Trigger Flow Control	Starts with three conditions setup to replicate the basic sequential trigger flow control.
Conditional Storage Qualifier	Enables the **Conditional** storage qualifier and **Basic OR** condition. This setup provides a versatile storage qualifier condition expression.
Transitional Storage Qualifier	Enables the **Transitional** storage qualifier. The **Transitional** storage qualifier simply detects changes in data.
Start-Stop Storage Qualifier	Enables the **Start/Stop** storage qualifier and the **Basic OR** condition. Provides two conditions to frame the data.
State-Based Storage Qualifier	Provides more sophisticated qualification conditions for use with state machine expressions. You must use the State-Based Storage Qualifier template in conjunction with the state-based trigger flow control

continued...

Template	Summary Description
Input Port Storage Qualifier	Enables the **Input port** storage qualifier to provide total control of the storage qualifier condition by supporting development of custom logic outside of the Signal Tap logic hierarchy.
Trivial Advanced Trigger Condition	Enables the **Advanced** trigger condition. The **Advanced** condition provides the most flexibility to express complex conditions. The **Advanced** trigger condition scales from a simple wire to the most complex logical expression. This template starts with the simplest condition.
Trigger Position Defined Using Sample Count	Supports specifying an exact number of samples to store after the trigger position, using the **State-Based Trigger Flow Control** template as a reference.
Cross-triggering Between STP Instances	Enables "Cross-triggering by using the **Trigger out** from one instance as the **Trigger in** of another instance, when using multiple Signal Tap instances.
Setup for Incremental Compilation	Specifies a fixed input width for signal inputs. This technique allows efficient incremental compilation by reducing the amount of Signal Tap logic change, and by adding only post-fit nodes to tap.
Define Trigger Condition in RTL	Supports defining a custom trigger condition in the RTL language of your choice.

Table 27. Standard Interface Signal Tap File Templates

Template	Summary Description
Capture Avalon Memory Mapped Transactions	Allows you to use the storage qualifier feature to store only meaningful Avalon memory-mapped interface transactions.
Simple Avalon Streaming Interface Bus Performance Analysis	Supports recording of event time for analysis of the data packet flow in an Avalon streaming interface.
Use Counters in the State-based Flow Control to Collect Stats	Use counters to track of the number of packets produced (`pkt_counter`), number of data beats produced (`pkt_beat_counter`), and number of data beats consumed (`stream_beat_counter`).

Table 28. State-Based Triggering Design Flow Examples Signal Tap File Templates

Template	Setup Description
Trigger on an Event Absent for Greater Than or Equal to 5 Clock Cycles	Requires setup of one basic trigger condition in the **Setup** tab to the value that you want.
Trigger on Event Absent for Less Than 5 Clock Cycles	Requires setup of one basic trigger condition in the **Setup** tab to the value that you want.
Trigger on 5th Occurrence of a Group Value	Requires setup of one basic trigger condition in the **Setup** tab to the value that you want.
Trigger on the 5th Transition of a Group Value	Requires setup of an edge-sensitive trigger condition to detect all bus transitions to the desired group value. Requires edge detection for any data bus bit logically ANDed with a comparison to the desired group value. An advanced trigger condition is necessary in this case.
Trigger After Condition1 is Followed by Condition2	Requires setup of three basic trigger conditions in the **Setup** tab to the values you specify. The first two trigger conditions are set to the desired group values. The third trigger condition is set to capture some type of idle transaction across the bus between the first and second conditions.
Trigger on Condition1 Immediately Followed by Condition2	Requires setup of two basic trigger conditions in the **Setup** tab to the group values that you want.
Trigger on Condition2 Not Occurring Between Condition1 and Condition3	Requires setup of three basic trigger conditions in the **Setup** tab to the group values that you want.

continued...

Template	Setup Description
Trigger on the 5th Consecutive Occurrence of Condition1	Requires setup of one basic trigger condition in the **Setup** tab to the value you want.
Trigger After a Violation of Sequence From Condition1 To Condition4	Requires setup of four basic trigger conditions to the sequence values that you want.
Trigger on a Sequence of Edges	Requires setup of three edge-sensitive basic trigger conditions for the sequence that you want.
Trigger on Condition1 Followed by Condition2 After 5 Clock Cycles	Requires setup of two basic trigger conditions to the group values that you want.
Trigger on Condition1 Followed by Condition2 Within 5 Samples	Requires setup of two basic trigger conditions to the group values that you want.
Trigger on Condition1 Not Followed by Condition2 Within 5 Samples	Requires setup of two basic trigger conditions to the group values that you want.
Trigger After 5 Consecutive Transitions	Requires setup of a trigger condition to capture any transition activity on the monitored bus. This example requires an Advanced trigger condition because the example requires an OR condition.
Trigger When Condition1 Occurs Less Than 5 Times Between Condition2 and Condition3	Requires setup of three edge-sensitive trigger conditions, with each trigger condition containing a comparison to the desired group value.

2.13. Running the Stand-Alone Version of Signal Tap

You can optionally install a stand-alone version of the Signal Tap logic analyzer, rather than using the Signal Tap logic analyzer integrated with the Intel Quartus Prime software.

The stand-alone version of Signal Tap is particularly useful in a lab environment that lacks a suitable workstation for a complete Intel Quartus Prime installation, or lacks a full Intel Quartus Prime software license.

The standalone version of the Signal Tap logic analyzer includes and requires use of the Intel Quartus Prime stand-alone Programmer, which is also available from the Download Center for FPGAs.

2.14. Signal Tap Scripting Support

The Intel Quartus Prime software supports automation of Signal Tap controls in a Tcl scripting environment, or with the `quartus_stp` executable. For detailed information about scripting command options, refer to the Intel Quartus Prime command-line and Tcl API help by typing `quartus_sh --qhelp` at the command prompt.

Related Information

Intel Quartus Prime Pro Edition User Guide: Scripting

2.14.1. Signal Tap Command-Line Options

You can use the following options with the `quartus_stp` executable:

Table 29. quartus_stp Command-Line Options

Option	Usage	Description
`--stp_file <stp_filename>`	Required	Specifies the name of the `.stp` file.
`--enable`	Optional	Sets the `ENABLE_SIGNALTAP` option to `ON` in the project's `.qsf` file, so the Signal Tap logic analyzer runs in the next compilation. If you omit this option, the Intel Quartus Prime software uses the current value of `ENABLE_SIGNALTAP` in the `.qsf` file. Writes subsequent Signal Tap assignments to the `.stp` that appears in the `.qsf` file. If the `.qsf` file does not specify a `.stp` file, you must use the `--stp_file` option.
`--disable`	Optional	Sets the `ENABLE_SIGNALTAP` option to `OFF` in the project's `.qsf` file, so the Signal Tap logic analyzer does not in the next compilation. If you omit the `--disable` option, the Intel Quartus Prime software uses the current value of `ENABLE_SIGNALTAP` in the `.qsf` file.

2.14.2. Data Capture from the Command Line

The `quartus_stp` executable supports a Tcl interface that allows you to capture data without running the Intel Quartus Prime GUI.

Note: You cannot execute Signal Tap Tcl commands from within the Tcl console in the Intel Quartus Prime software.

To execute a Tcl script containing Signal Tap logic analyzer Tcl commands, use:

```
quartus_stp -t <Tcl file>
```

Example 2. Continuously Capturing Data

This excerpt shows commands you can use to continuously capture data. Once the capture meets trigger condition, the Signal Tap logic analyzer starts the capture and stores the data in the data log.

```
#  Open Signal Tap session
open_session -name stp1.stp

###  Start acquisition of instances auto_signaltap_0 and
###  auto_signaltap_1 at the same time

# Calling run_multiple_end starts all instances
run_multiple_start

run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger \
     trigger_1 -data_log log_1 -timeout 5
run -instance auto_signaltap_1 -signal_set signal_set_1 -trigger \
     trigger_1 -data_log log_1 -timeout 5

run_multiple_end

# Close Signal Tap session
close_session
```

Related Information

::quartus::stp
In *Intel Quartus Prime Help*

2.15. Signal Tap File Version Compatibility

If you open a `.stp` file created in a previous version of the Intel Quartus Prime software in a newer version of the software, you can no longer open that `.stp` file in the previous version of the Intel Quartus Prime software.

If you open an Intel Quartus Prime project that includes a `.stp` file from a previous version of the software in a later version of the Intel Quartus Prime software, the software may require you to update the `.stp` configuration file before you can compile the project. Update the configuration file by simply opening the `.stp` in the Signal Tap logic analyzer GUI. If configuration update is required, Signal Tap confirms that you want to update the `.stp` to match the current version of the Intel Quartus Prime software.

Note: The Intel Quartus Prime Pro Edition software uses a new methodology for settings and assignments. For example, Signal Tap assignments include only the `instance` name, not the `entity:instance` name. Refer to *Migrating to Intel Quartus Prime Pro Edition* for more information about migrating existing Signal Tap files (`.stp`) to Intel Quartus Prime Pro Edition.

Related Information

Migrating to Intel Quartus Prime Pro Edition, Intel Quartus Prime Pro Edition User Guide: Getting Started

2.16. Design Debugging with the Signal Tap Logic Analyzer Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2023.12.04	23.4	• Added support for != operator to *Comparison Trigger Conditions* topic.
2023.10.02	23.3	• Enhanced the information in *Enabling a Power-Up Trigger*. • Made a minor correction in the image in *Specifying the Buffer Acquisition Mode*. • Made a minor correction in the description in *Managing Signal Tap Instances with Run-Time and Power-Up Trigger Conditions*.
2022.07.08	22.1	• Fixed broken link in *Custom State-Based Triggering Flow Examples*.
2022.03.28	22.1	• Added *Organizing Signals in the Signal Tap Logic Analyzer* topic. • Removed incorrect links from *Intel Quartus Prime Pro Edition User Guide Debug Tools Archives* topic.
2021.10.13	21.3	• Added *Recompiling Only Signal Tap Changes* topic. • Changed title of *Prevent Changes Requiring Recompilation* to *Preventing Changes that Require Full Recompilation* and revised figures.

continued...

Document Version	Intel Quartus Prime Version	Changes
2021.10.04	21.3	• Updated *Signal Tap Logic Analyzer Introduction* with *Signal Tap Logic Analyzer and Simulator Integration* section. • Added description of Autorun mode to *Managing Signal Tap Instances* topic. • Added new *Adding Simulator-Aware Signal Tap Nodes* topic. • Added new *Add Simulator Aware Node Finder Settings* topic. • Added new *Signal Tap and Simulator Integration* topic. • Added new *Generating a Simulation Testbench from Signal Tap Data* topic. • Added new *Create Simulation Testbench Dialog Box Settings* topic. • Revised *Preserving Signals for Monitoring and Debugging* topic for latest techniques and links to other resources. • Revised *Adding Pre-Synthesis or Post-Fit Nodes* for latest techniques and links to other resources. • Added new *Changing the Post-Fit Signal Tap Target Nodes* topic. • Updated *Adding Pre-Synthesis or Post-Fit Nodes* topic for preserve for debug filters. • Added details about SOF Manager to *Ensure Compatibility Between .stp and .sof Files* topic.
2020.09.28	20.3	• Revised "Signal Tap Logic Analyzer Introduction" for screenshot and details about role of Signal Tap Intel FPGA IP. • Revised graphic and wording in "Signal Tap Hardware and Software Requirements" topic. • Revised wording and link to download in "Running the Stand-Alone Version of Signal Tap." • Updated flow diagram and added links to retitled "Signal Tap Debugging Flow" topic. • Retitled "Add the Signal Tap Logic Analyzer to Your Design" to "Step 1: Add the Signal Tap Logic Analyzer to the Project," and referenced new template and added links to next steps. • Added "Creating a Signal Tap Instance with the Signal Tap GUI" topic. • Added new "Signal Tap File Templates" topic. • Added new "Creating a Signal Tap Instance by HDL Instantiation" topic. • Added new "Signal Tap Intel FPGA IP Parameters" topic. • Retitled "Configure the Signal Tap Logic Analyzer" to "Step 2: Configure the Signal Tap Logic Analyzer," and referenced new template and added links to next steps. • Enhanced description in Step 5: Run the Signal Tap Logic Analyzer" topic. • Revised "Adding Signals to the Signal Tap Logic Analyzer" to add detailed steps and screenshot. • Retitled and revised "Adding Nios II Processor Signals" to reflect there is only one plug-in in Intel Quartus Prime Pro Edition. • Revised "Disabling or Enabling Signal Tap Instances" and added screenshot. • Replaced outdated links to AN446 with links to AN845. • Revised headings and steps in "Debugging Block-Based Designs with Signal Tap" section. • Retitled "Debugging Imported Snapshots" to "Compiler Snapshots and Signal Tap Debugging". • Retitled "Backward Compatibility" to "Signal Tap File Version Compatibility." • Removed incorrect statement about debugging multiple designs from "Step 4: Program the Target Hardware" topic.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Removed reference to obsolete resource checking function from "Ensure Compatibility Between STP and SOF Files" topic. • Removed obsolete "Remote Debugging Using the Signal Tap Logic Analyzer" section. • Removed obsolete "Estimating FPGA Resources" topic.
2019.06.11	18.1.0	Added more explanation to Comparing Continuous and Input Port Capture Mode in Data Acquisition of a Recurring Data Pattern about continuous and input mode.
2019.05.01	18.1.0	In *Adding Signals with a Plug-In* topic, removed outdated information from step 1 about turning on **Create debugging nodes for IP cores**.
2018.09.24	18.1.0	• Added content about debugging designs in block-based flows. • Renamed topic: *Untappable Signals* to *Signals Unavailable for Signal Tap Debugging*.
2018.08.07	18.0.0	Reverted document title to *Debug Tools User Guide: Intel Quartus Prime Pro Edition*.
2018.07.30	18.0.0	Updated Partial Reconfiguration sections to reflect changes in the PR flow.
2018.05.07	18.0.0	• Added note stating Signal Tap IP not optimized for Stratix 10 Devices. • Moved information about debug fabric on PR designs to the *System Debugging Tools Overview* chapter. • Removed restrictions of Rapid Recompile support for Intel Stratix 10 devices.
2017.11.06	17.1.0	• Added support for Incremental Routing in Intel Stratix 10 devices. • Removed unsupported FSM auto detection. • Clarified information about the Data Log Pane. • Updated Figure: Data Log and renamed to Simple Data Log. • Added Figure: Accessing the Advanced Trigger Condition Tab. • Removed outdated information about command-line flow.
2017.05.08	17.0.0	• Added: Open Standalone Signal Tap Logic Analyzer GUI. • Added: Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer. • Updated figures on Create Signal Tap File from Design Instance(s).
2016.10.31	16.1.0	• Implemented Intel rebranding. • Added: Create SignalTap II File from Design Instance(s). • Removed reference to unsupported Talkback feature.
2016.05.03	16.0.0	• Added: Specifying the Pipeline Factor • Added: Comparison Trigger Conditions
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*. • Updated content to reflect SignalTap II support in Intel Quartus Prime Pro Edition
2015.05.04	15.0.0	Added content for Floating Point Display Format in table: SignalTap II Logic Analyzer Features and Benefits.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings.
December 2014	14.1.0	• Added MAX 10 as supported device. • Removed Full Incremental Compilation setting and Post-Fit (Strict) netlist type setting information. • Removed outdated GUI images from "Using Incremental Compilation with the SignalTap II Logic Analyzer" section.

continued...

Document Version	Intel Quartus Prime Version	Changes
June 2014	14.0.0	• DITA conversion. • Replaced MegaWizard Plug-In Manager and Megafunction content with IP Catalog and parameter editor content. • Added flows for custom trigger HDL object, Incremental Route with Rapid Recompile, and nested groups with Basic OR. • GUI changes: toolbar, drag to zoom, disable/enable instance, trigger log time-stamping.
November 2013	13.1.0	Removed HardCopy material. Added section on using cross-triggering with DS-5 tool and added link to white paper 01198. Added section on remote debugging an Altera SoC and added link to application note 693. Updated support for MEX function.
May 2013	13.0.0	• Added recommendation to use the state-based flow for segmented buffers with separate trigger conditions, information about Basic OR trigger condition, and hard processor system (HPS) external triggers. • Updated "Segmented Buffer" on page 13-17, Conditional Mode on page 13-21, Creating Basic Trigger Conditions on page 13-16, and Using External Triggers on page 13-48.
June 2012	12.0.0	Updated Figure 13–5 on page 13–16 and "Adding Signals to the SignalTap II File" on page 13–10.
November 2011	11.0.1	Template update. Minor editorial updates.
May 2011	11.0.0	Updated the requirement for the standalone SignalTap II software.
December 2010	10.0.1	Changed to new document template.
July 2010	10.0.0	• Add new acquisition buffer content to the "View, Analyze, and Use Captured Data" section. • Added script sample for generating hexadecimal CRC values in programmed devices. • Created cross references to Quartus II Help for duplicated procedural content.
November 2009	9.1.0	No change to content.
March 2009	9.0.0	• Updated Table 13–1 • Updated "Using Incremental Compilation with the SignalTap II Logic Analyzer" on page 13–45 • Added new Figure 13–33 • Made minor editorial updates
November 2008	8.1.0	Updated for the Quartus II software version 8.1 release: • Added new section "Using the Storage Qualifier Feature" on page 14–25 • Added description of `start_store` and `stop_store` commands in section "Trigger Condition Flow Control" on page 14–36 • Added new section "Runtime Reconfigurable Options" on page 14–63
May 2008	8.0.0	Updated for the Quartus II software version 8.0: • Added "Debugging Finite State machines" on page 14-24 • Documented various GUI usability enhancements, including improvements to the resource estimator, the bus find feature, and the dynamic display updates to the counter and flag resources in the State-based trigger flow control tab • Added "Capturing Data Using Segmented Buffers" on page 14–16 • Added hyperlinks to referenced documents throughout the chapter • Minor editorial updates

3. Quick Design Verification with Signal Probe

This chapter describes a technique that provides debug access to internal device signals without affecting the design.

The Signal Probe feature in the Intel Quartus Prime Pro Edition software allows you to route an internal node to a top-level I/O. When you start with a fully routed design, you can select and route debugging signals to I/O pins that you previously reserve or are currently unused.

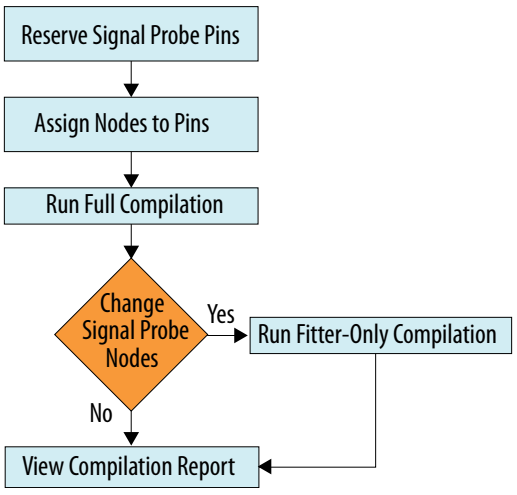
Related Information

System Debugging Tools Overview on page 6

3.1. Signal Probe Debugging Flow

Use the following flow to add Signal Probe debugging and verification capabilities to your design:

Figure 96. Signal Probe Debugging Flow



Step 1: Reserve Signal Probe Pins on page 125
Step 2: Assign Nodes to Signal Probe Pins on page 125
Step 3: Connect the Signal Probe Pin to an Output Pin on page 125
Step 4: Compile the Design on page 126
(Optional) Step 5: Modify the Signal Probe Pins Assignments on page 126
Step 6: Run Fitter-Only Compilation on page 126
Step 7: Check Connection Table in Fitter Report on page 127

3.1.1. Step 1: Reserve Signal Probe Pins

You must first create and reserve a pin for Signal Probe with a Tcl command:

```
set_global_assignment –name CREATE_SIGNALPROBE_PIN <pin_name>
```

pin_name Specifies the name of the Signal Probe pin.

Optionally, you can assign locations for the Signal Probe pins. If you do not assign a location, the Fitter places the pins automatically.

Note: If from the onset of the debugging process you know which internal signals you want to route, you can reserve pins and assign nodes before compilation. This early assignment removes the recompilation step from the flow.

Example 3. Tcl Command to Reserve Signal Probe Pins

```
set_global_assignment -name CREATE_SIGNALPROBE_PIN wizard
set_global_assignment -name CREATE_SIGNALPROBE_PIN probey
```

Related Information

Constraining Designs with Tcl Scripts

3.1.2. Step 2: Assign Nodes to Signal Probe Pins

You can assign any node in the post-compilation netlist to a Signal Probe pin. In the Intel Quartus Prime software, click **View ➤ Node Finder**, and filter by **Signal Tap: post-fitting** to view the nodes you can route.

You specify the node that connects to a Signal Probe pin with a Tcl command:

```
set_instance_assignment –name CONNECT_SIGNALPROBE_PIN <pin_name> \
     –to <node_name>
```

pin_name Specifies the name of the Signal Probe pin that connects to the node.

node_name Specifies the full hierarchy path of the node you want to route.

Example 4. Tcl Commands to Connect Pins to Internal Nodes

```
# Make assignments to connect nodes of interest to pins
set_instance_assignment -name CONNECT_SIGNALPROBE_PIN wizard -to sprobe_me1
set_instance_assignment -name CONNECT_SIGNALPROBE_PIN probey -to sprobe_me2
```

3.1.3. Step 3: Connect the Signal Probe Pin to an Output Pin

Once you reserve pins and assign internal nodes to the Signal Probe pins, you must connect the Signal Probe pin to an external output pin.

Example 5. Tcl Command to Specify Signal Probe Pin Assignment

```
set_instance_assignment –name CONNECT_SIGNALPROBE_PIN <pin_name> –to <node_name>
```

3.1.4. Step 4: Compile the Design

Perform a full compilation of the design. You can use Intel Quartus Prime software GUI, a command line executable, or the following Tcl command to start the Compiler

Example 6. Tcl Command to Compile the Design

```
execute_flow -compile
```

At this point in the design flow, you can determine the nodes that you want to debug.

Related Information

Design Compilation

3.1.5. (Optional) Step 5: Modify the Signal Probe Pins Assignments

As long as you reserve the pins (with CREATE_SIGNALPROBE_PIN) before running full compilation, you can optionally add or modify the node that connects to a reserved Signal Probe pin (with CONNECT_SIGNALPROBE_PIN) without rerunning a full compilation. Rather, you can run a Fitter-only compilation to implement the Signal Probe pin assignment change.

Note: If you modify the physical I/O pin assignments with (with CREATE_SIGNALPROBE_PIN) after running compilation, you must rerun full compilation to implement those changes before using Signal Probe.

Example 7. Tcl Command to Specify Signal Probe Pin Assignment

```
set_instance_assignment –name CONNECT_SIGNALPROBE_PIN <pin_name> –to <node_name>
```

3.1.6. Step 6: Run Fitter-Only Compilation

After re-assigning nodes to the Signal Probe pins, you can run a Fitter-only compilation (using `--recompile`) to implement the post-fit change without rerunning a full compilation.

Example 8. Tcl Command to Run Fitter-Only Compile

```
# Run the fitter with --recompile to preserve timing
# and quickly connect the Signal Probe pins
execute_module -tool fit -args {--recompile}
```

After recompilation, you are ready to program the device and debug the design.

Related Information

- Using the ECO Compilation Flow in *Intel Quartus Prime Pro Edition User Guide: Design Optimization*
- Using the ECO Compilation Flow in *Intel Quartus Prime Pro Edition User Guide: Design Optimization*

3.1.7. Step 7: Check Connection Table in Fitter Report

When you compile a design with Signal Probe pins, Compiler generates a connection report showing the connection status to Signal Probe pins.

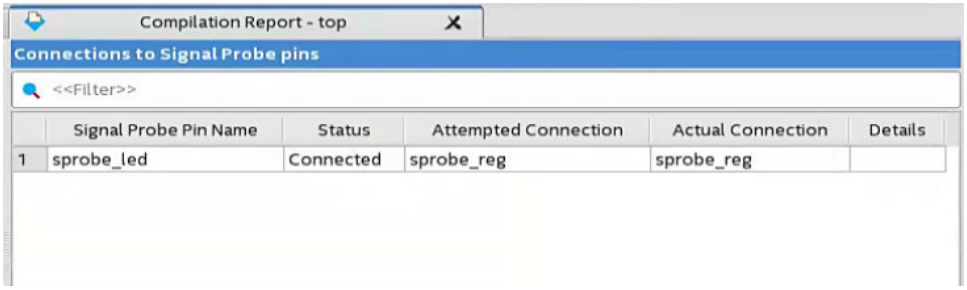
To view this report, click **Processing ➤ Compilation Report**, open the **Fitter ➤ In-System Debugging** folder, and click **Connections to Signal Probe pins**.

The **Status** column indicates whether or not the routing attempt from the nodes to the Signal Probe pins is successful.

Table 30. Status of Signal Probe Connection

Status	Description
Connected	Routing succeeded.
Unconnected	Routing did not succeed. Possible reasons are: • Node belongs to an I/O cell or another hard IP, thus cannot be routed. • Node hierarchy path does not exist in the design. • Node is not **Signal Tap: post-fitting**.

Example 9. Connections to Signal Probe Pins in the Compilation Report



Alternatively, you can find the Signal Probe connection information in the Fitter report file (<*project_name*>.fit.rpt).

Example 10. Connections to Signal Probe Pins in top.fit.rpt

```
+----------------------------------------------------------------------------
+
; Connections to Signal Probe
pins                                                ;
+----------------------------------------------------------------------------
+
Signal Probe Pin Name : probey
Status                : Connected
Attempted Connection  : sprobe_me2
Actual Connection     : sprobe_me2
Details               :

Signal Probe Pin Name : wizard
Status                : Connected
Attempted Connection  : sprobe_me1
Actual Connection     : sprobe_me1
Details               :
+----------------------------------------------------------------------------
+
```

Related Information

- Signals Unavailable for Signal Tap Debugging on page 51
- Text-Based Report Files

3.2. Quick Design Verification with Signal Probe Revision History

Document Version	Intel Quartus Prime Version	Changes
2021.10.04	21.3	• Removed references to obsolete Rapid Recompile feature. • Updated *Signal Probe Debugging Flow* topic for new optional step and added flow diagram. • Added step numbers to tasks in flow to emphasize order of operations. • Added *(Optional) Step 4: Modify the Signal Probe Pins Assignments* topic. • Revised wording in *Step 5: Run Fitter-Only Compilation*. • Revised screenshot and wording in *Step 6: Check Connection Table in Fitter Report*. • Added new *Step 3: Connect the Signal Probe Pin to an Output Pin* topic.
2018.05.07	18.0.0	Initial release for Intel Quartus Prime Pro Edition software.

4. In-System Debugging Using External Logic Analyzers

4.1. About the Intel Quartus Prime Logic Analyzer Interface

The Intel Quartus Prime Logic Analyzer Interface (LAI) allows you to use an external logic analyzer and a minimal number of Intel-supported device I/O pins to examine the behavior of internal signals while your design is running at full speed on your Intel-supported device.

The LAI connects a large set of internal device signals to a small number of output pins. You can connect these output pins to an external logic analyzer for debugging purposes. In the Intel Quartus Prime LAI, the internal signals are grouped together, distributed to a user-configurable multiplexer, and then output to available I/O pins on your Intel-supported device. Instead of having a one-to-one relationship between internal signals and output pins, the Intel Quartus Prime LAI enables you to map many internal signals to a smaller number of output pins. The exact number of internal signals that you can map to an output pin varies based on the multiplexer settings in the Intel Quartus Prime LAI.

Note: The term "logic analyzer" when used in this document includes both logic analyzers and oscilloscopes equipped with digital channels, commonly referred to as mixed signal analyzers or MSOs.

The LAI does not support Hard Processor System (HPS) I/Os.

Related Information

Devices Support Center

4.2. Choosing a Logic Analyzer

The Intel Quartus Prime software offers the following two general purpose on-chip debugging tools for debugging a large set of RTL signals from your design:

- The Signal Tap Logic Analyzer
- An external logic analyzer, which connects to internal signals in your Intel-supported device by using the Intel Quartus Prime LAI

Table 31. Comparing the Signal Tap Logic Analyzer with the Logic Analyzer Interface

Feature	Description	Recommended Logic Analyzer
Sample Depth	You have access to a wider sample depth with an external logic analyzer. In the Signal Tap Logic Analyzer, the maximum sample depth is set to	LAI

continued...

Feature	Description	Recommended Logic Analyzer
	128 Kb, which is a device constraint. However, with an external logic analyzer, there are no device constraints, providing you a wider sample depth.	
Debugging Timing Issues	Using an external logic analyzer provides you with access to a "timing" mode, which enables you to debug combined streams of data.	LAI
Performance	You frequently have limited routing resources available to place and route when you use the Signal Tap Logic Analyzer with your design. An external logic analyzer adds minimal logic, which removes resource limits on place-and-route.	LAI
Triggering Capability	The Signal Tap Logic Analyzer offers triggering capabilities that are comparable to external logic analyzers.	LAI or Signal Tap
Use of Output Pins	Using the Signal Tap Logic Analyzer, no additional output pins are required. Using an external logic analyzer requires the use of additional output pins.	Signal Tap
Acquisition Speed	With the Signal Tap Logic Analyzer, you can acquire data at speeds of over 200 MHz. You can achieve the same acquisition speeds with an external logic analyzer; however, you must consider signal integrity issues.	Signal Tap

Related Information

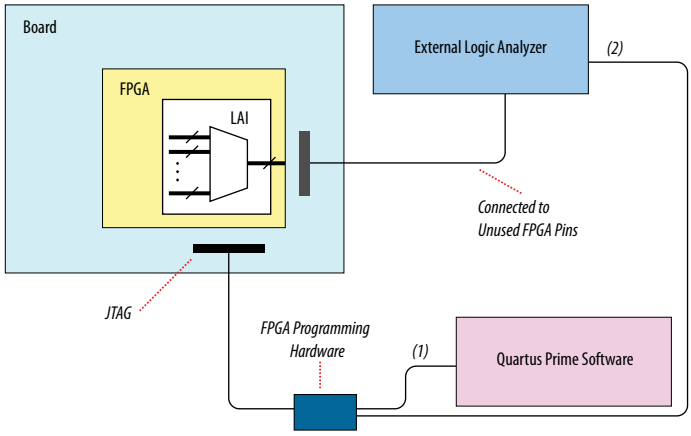
System Debugging Tools Overview on page 6

4.2.1. Required Components

To perform analysis using the LAI you need the following components:

- Intel Quartus Prime software version 15.1 or later
- The device under test
- An external logic analyzer
- An Intel FPGA communications cable
- A cable to connect the Intel-supported device to the external logic analyzer

Figure 97. LAI and Hardware Setup

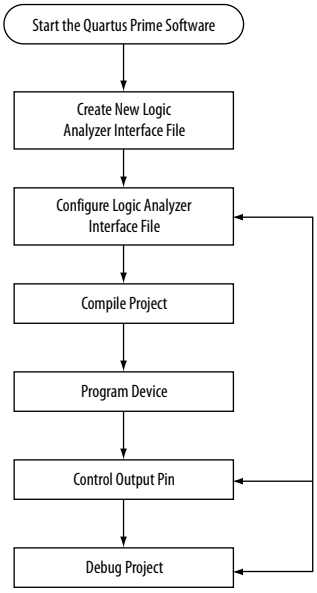


Notes to figure:

1. Configuration and control of the LAI using a computer loaded with the Intel Quartus Prime software via the JTAG port.
2. Configuration and control of the LAI using a third-party vendor logic analyzer via unused FPGA pins. Support varies by vendor.

4.3. Flow for Using the LAI

Figure 98. LAI Workflow

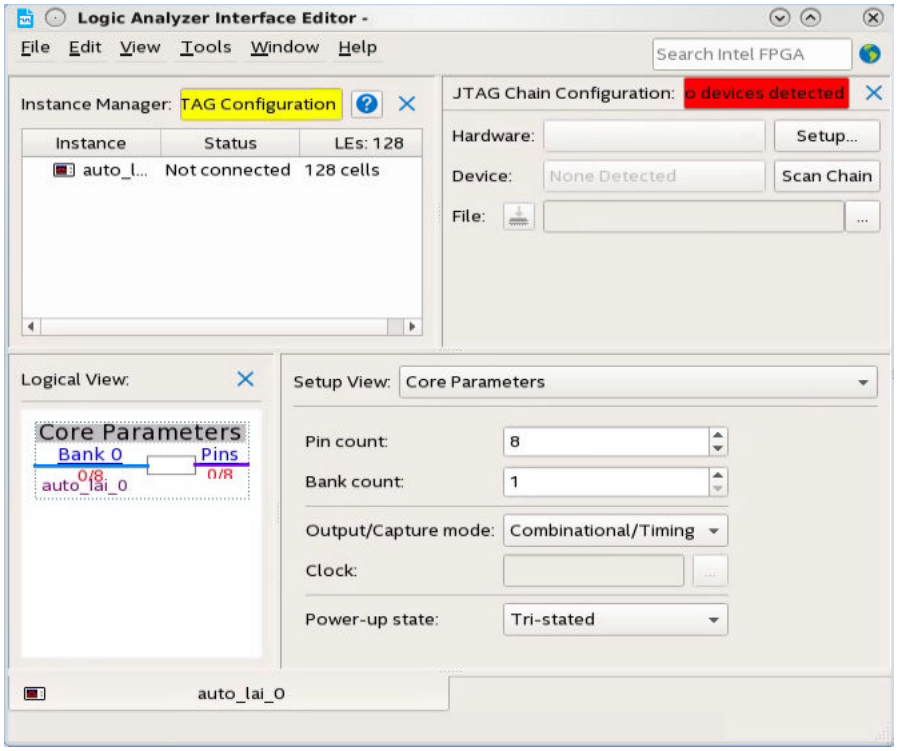


4.3.1. Defining Parameters for the Logic Analyzer Interface

The **Logic Analyzer Interface Editor** allows you to define the parameters of the LAI.

- Click **Tools ➤ Logic Analyzer Interface Editor**.

Figure 99. Logic Analyzer Interface Editor



- In the **Setup View** list, select **Core Parameters**.
- Specify the parameters of the LAI instance.

Related Information

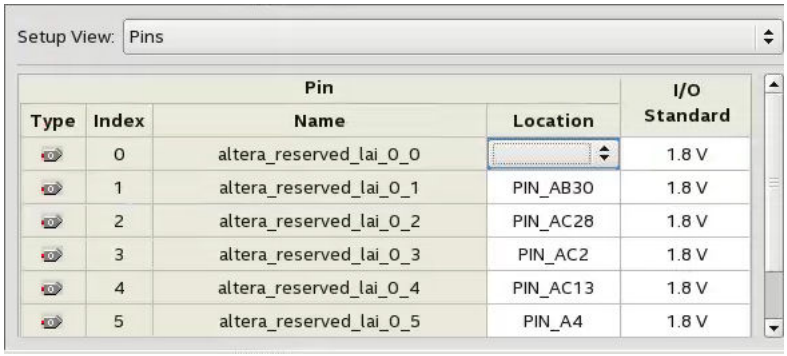
LAI Core Parameters on page 135

4.3.2. Mapping the LAI File Pins to Available I/O Pins

To assign pin locations for the LAI:

1. Select **Pins** in the **Setup View** list

Figure 100. Mapping LAI file Pins



2. Double-click the **Location** column next to the reserved pins in the **Name** column, and select a pin from the list.
3. Right-click the selected pin and locate in the Pin Planner.

Related Information

Managing Device I/O Pins
In *Intel Quartus Prime Pro Edition User Guide: Design Constraints*

4.3.3. Mapping Internal Signals to the LAI Banks

After specifying the number of banks to use in the **Core Parameters** settings page, you must assign internal signals for each bank in the LAI.

1. Click the **Setup View** arrow and select **Bank n** or **All Banks**.
2. To view all the bank connections, click **Setup View** and then select **All Banks**.
3. Before making bank assignments, right click the Node list and select **Add Nodes** to open the **Node Finder**.
4. Find the signals that you want to acquire.
5. Drag the signals from the **Node Finder** dialog box into the bank **Setup View**.

   When adding signals, use **Signal Tap: pre-synthesis** for non-incrementally routed instances and **Signal Tap: post-fitting** for incrementally routed instances

   As you continue to make assignments in the bank **Setup View**, the schematic of the LAI in the **Logical View** pane begins to reflect the changes.
6. Continue making assignments for each bank in the **Setup View** until you add all the internal signals that you want to acquire.

Related Information

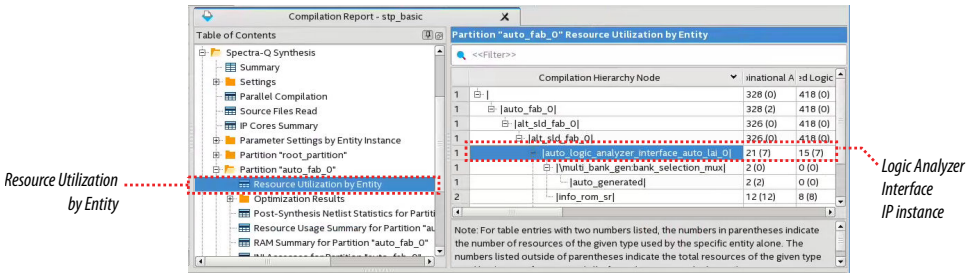
Node Finder Command
In *Intel Quartus Prime Help*

4.3.4. Compiling Your Intel Quartus Prime Project

After you save your `.lai` file, a dialog box prompts you to enable the Logic Analyzer Interface instance for the active project. Alternatively, you can define the `.lai` file your project uses in the **Global Project Settings** dialog box. After specifying the name of your `.lai` file, compile your project.

To verify the Logic Analyzer Interface is properly compiled with your project, open the **Compilation Report** tab and select Resource Utilization by Entity, nested under Partition "auto_fab_0". The LAI IP instance appears in the Compilation Hierarchy Node column, nested under the internal module of `auto_fab_0`

Figure 101. LAI Instance in Compilation Report



4.3.5. Programming Your Intel-Supported Device Using the LAI

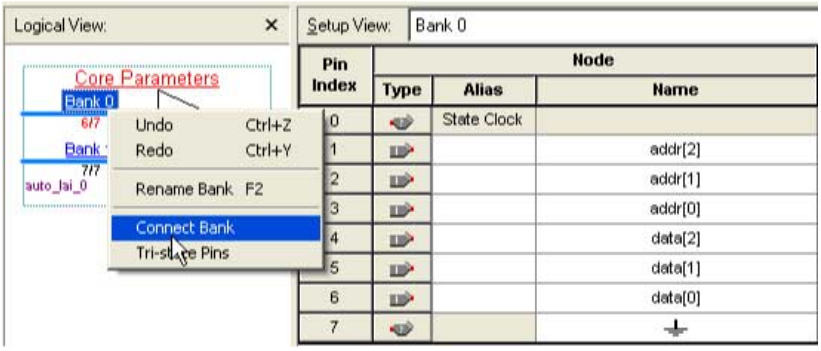
After compilation completes, you must configure your Intel-supported device before using the LAI.

You can use the LAI with multiple devices in your JTAG chain. Your JTAG chain can also consist of devices that do not support the LAI or non-Intel, JTAG-compliant devices. To use the LAI in more than one Intel-supported device, create an `.lai` file and configure an `.lai` file for each Intel-supported device that you want to analyze.

4.4. Controlling the Active Bank During Runtime

When you have programmed your Intel-supported device, you can control which bank you map to the reserved `.lai` file output pins. To control which bank you map, in the schematic in the Logical View, right-click the bank and click **Connect Bank**.

Figure 102. Configuring Banks



4.4.1. Acquiring Data on Your Logic Analyzer

To acquire data on your logic analyzer, you must establish a connection between your device and the external logic analyzer. For more information about this process and for guidelines about how to establish connections between debugging headers and logic analyzers, refer to the documentation for your logic analyzer.

4.5. LAI Core Parameters

The table lists the LAI file core parameters:

Table 32. LAI File Core Parameters

Parameter	Range Value	Description
Pin Count	1 - 255	Number of pins dedicated to the LAI. You must connect the pins to a debug header on the board. Within the device, The Compiler maps each pin to a user-configurable number of internal signals.
Bank Count	1 - 255	Number of internal signals that you want to map to each pin. For example, a **Bank Count** of 8 implies that you connect eight internal signals to each pin.
Output/Capture Mode		Specifies the acquisition mode. The two options are: • **Combinational/Timing**—This acquisition mode uses the external logic analyzer's internal clock to determine when to sample data. This acquisition mode requires you to manually determine the sample frequency to debug and verify the system, because the data sampling is asynchronous to the Intel-supported device. This mode is effective if you want to measure timing information such as channel-to-channel skew. For more information about the sampling frequency and the speeds at which it can run, refer to the external logic analyzer's data sheet. • **Registered/State**—This acquisition mode determines when to sample from a signal on the system under test. Consequently, the data samples are synchronous with the Intel-supported device. The **Registered/State** mode provides a functional view of the Intel-supported device while it is running. This mode is effective when you verify the functionality of the design.
Clock		Specifies the sample clock. You can use any signal in the design as a sample clock. However, for best results, use a clock with an operating frequency fast enough to sample the data that you want to acquire. *Note:* The **Clock** parameter is available only when **Output/Capture Mode** is set to **Registered State**.
Power-Up State		Specifies the power-up state of the pins designated for use with the LAI. You can select tri-stated for all pins, or selecting a particular bank that you enable.

Related Information

Defining Parameters for the Logic Analyzer Interface on page 131

4.6. In-System Debugging Using External Logic Analyzers Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2023.10.02	23.3	• Made a minor correction in the description in *Required Components*. • Removed the figure notes in *Flow for Using the LAI*.
2022.07.08	22.1	• Fixed broken link in "About the Intel Quartus Prime Logic Analyzer Interface"

continued...

Document Version	Intel Quartus Prime Version	Changes
2018.05.07	18.0.0	• Moved list of LAI File Core Parameters from *Configuring the File Core Parameters* to its own topic, and added links.
2017.05.08	17.0.0	• Updated *Compiling Your Intel Quartus Prime Project* • Updated figure: LAI Instance in Compilation Report.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	• Dita conversion • Added limitation about HPS I/O support
June 2012	12.0.0	Removed survey link
November 2011	10.1.1	Changed to new document template
December 2010	10.1.0	• Minor editorial updates • Changed to new document template
August 2010	10.0.1	Corrected links
July 2010	10.0.0	• Created links to the Intel Quartus Prime Help • Editorial updates • Removed Referenced Documents section
November 2009	9.1.0	• Removed references to APEX devices • Editorial updates
March 2009	9.0.0	• Minor editorial updates • Removed Figures 15–4, 15–5, and 15–11 from 8.1 version
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content
May 2008	8.0.0	• Updated device support list on page 15–3 • Added links to referenced documents throughout the chapter • Added "Referenced Documents" • Added reference to *Section V. In-System Debugging* • Minor editorial updates

5. In-System Modification of Memory and Constants

The Intel Quartus Prime In-System Memory Content Editor (ISMCE) allows to view and update memories and constants at runtime through the JTAG interface. By testing changes to memory contents in the FPGA while the design is running, you can identify, test, and resolve issues.

The ability to read data from memories and constants can help you identify the source of problems, and the write capability allows you to bypass functional issues by writing expected data.

When you use the In-System Memory Content Editor in conjunction with the Signal Tap logic analyzer, you can view and debug your design in the hardware lab.

Related Information

- System Debugging Tools Overview on page 6
- Design Debugging with the Signal Tap Logic Analyzer on page 29

5.1. IP Cores Supporting In System Memory Content Editor

You can use the In System Memory Content Editor (ISMCE) with the following Intel FPGA IP cores in the current version of the Intel Quartus Prime Pro Edition software:

Table 33. IP Cores Supporting ISMCE

Device Family	IP Supported for ISMCE
Intel Agilex® 7 devices Intel Stratix 10 devices Intel Cyclone® 10 GX devices Intel Arria 10 devices	• RAM: 1-PORT Intel FPGA IP • ROM: 1-PORT Intel FPGA IP

Note: To use the ISMCE tool with designs migrated from an older device to the Intel Stratix 10 device or the Intel Agilex 7 device, you must first replace instances of the altsyncram Intel FPGA IP with the altera_syncram Intel FPGA IP.

Related Information

- Intel Stratix 10 Embedded Memory IP Core References
  In *Intel Stratix 10 Embedded Memory User Guide*
- About Embedded Memory IP Cores
  In *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*
- Intel Agilex 7 Embedded Memory User Guide

5.2. Debug Flow with the In-System Memory Content Editor

To debug a design with the In-System Memory Content Editor:

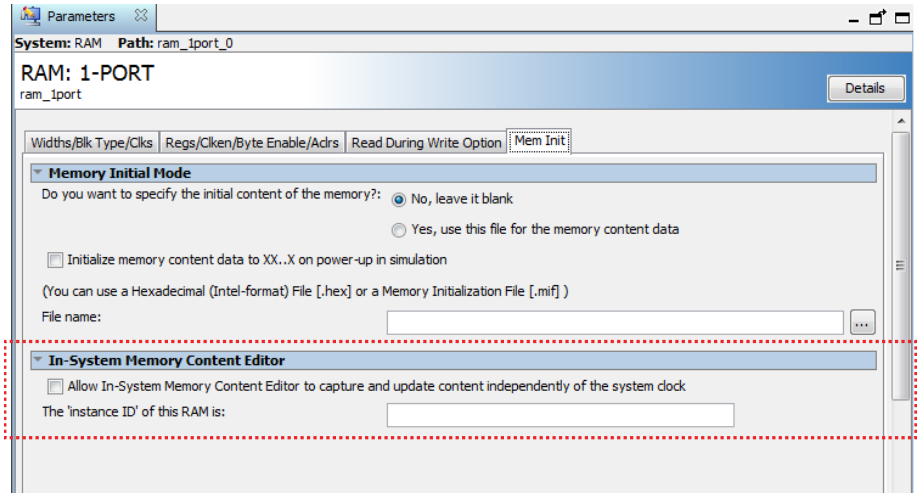
1. Identify the memories and constants that you want to access at runtime.
2. Specify in the design the memory or constant that must be run-time modifiable.
3. Perform a full compilation.
4. Program the device.
5. Launch the In-System Memory Content Editor.
   The In-System Memory Content Editor retrieves all instances of run-time configurable memories and constants by scanning the JTAG chain and sending a query to the device selected in the JTAG Chain Configuration pane.
6. Modify the values of the memories or constants, and check the results.

   For example, if a parity bit in a memory is incorrect, you can use the In-System Memory Content Editor to write the correct parity bit values into the RAM, allowing the system to continue functioning. To check the error handling functionality of a design, you can intentionally write incorrect parity bit values into the RAM.

5.3. Enabling Runtime Modification of Instances in the Design

To make an instance of a memory or constant runtime-modifiable:

1. Open the instance with the Parameter Editor.
2. In the Parameter Editor, turn on **Allow In-System Memory Content Editor to capture and update content independently of the system clock**.



3. Recompile the design.

When you specify that a memory or constant is run-time modifiable, the Intel Quartus Prime software changes the default implementation to enable run-time modification without changing the functionality of your design, by:

- Converting single-port RAMs to dual-port RAMs
- Adding logic to avoid memory write collision and maintain read write coherency in device families that do not support true dual-port RAMs, such as the Intel Stratix 10 device family.

5.4. Programming the Device with the In-System Memory Content Editor

After compilation, you must program the design in the FPGA. You can use the **JTAG Chain Configuration** pane to program the device from within the In-System Memory Content Editor.

Related Information

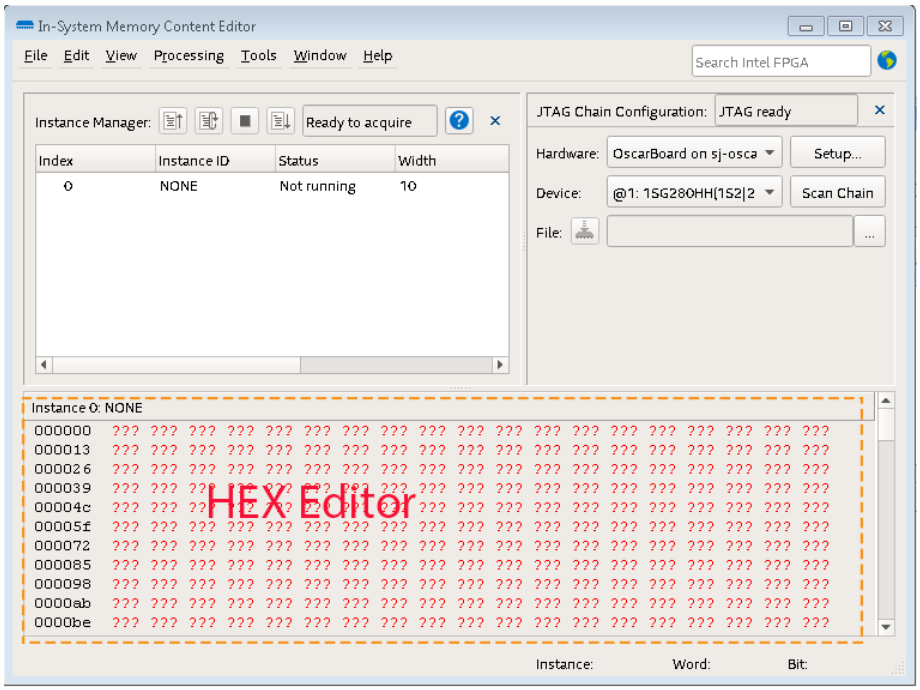
JTAG Chain Configuration Pane (In-System Memory Content Editor)
In *Intel Quartus Prime Help*

5.5. Loading Memory Instances to the ISMCE

To view the content of reconfigurable memory instances:

1. On the Intel Quartus Prime software, click **Tools ➤ In-System Memory Content Editor**.

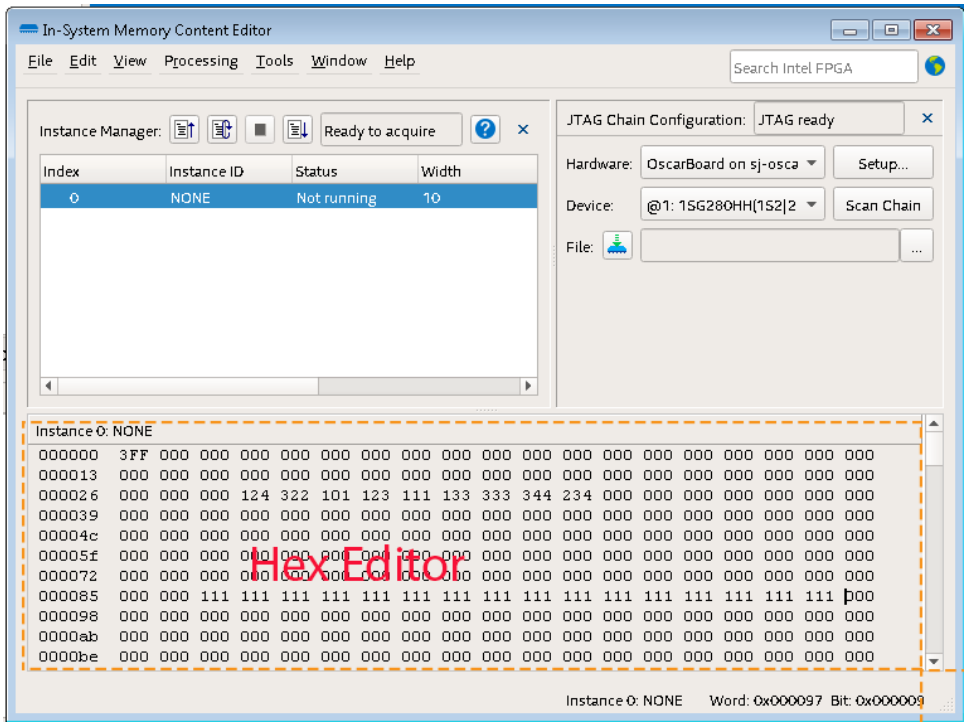
Figure 103. Hex Editor After Scanning JTAG Chain



2. In the **JTAG Chain Configuration** pane, click **Scan Chain**. The In-System Memory Content Editor sends a query to the device in the **JTAG Chain Configuration** pane and retrieves all instances of run-time configurable memories and constants.
   The **Instance Manager** pane lists all the instances of constants and memories that are runtime-modifiable. The **Hex Editor** pane displays the contents of each memory or constant instance. The memory contents in the **Hex Editor** pane appear as red question marks until you read the device.
3. Click an instance from the **Instance manager**, and then click to load the contents of that instance.

The Hex Editor now displays the contents of the instance.

Figure 104. Hex Editor Displaying Instance



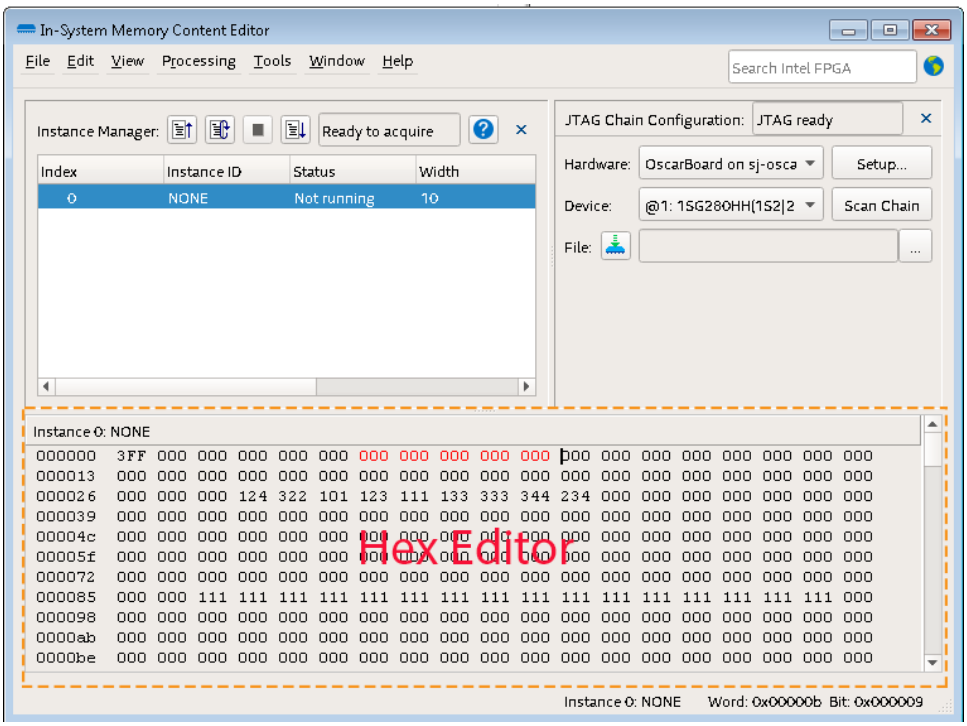
5.6. Monitoring Locations in Memory

The ISMCE allows you to monitor information in memory regions. For example, you can determine if a counter increments, or if a given word changes. For memories connected to a Nios II processor, you can observe how the software uses key regions of memory.

- Click to synchronize the Hex Editor to the current instance's content. The Hex Editor displays in red content that changed with respect to the last device synchronization.
- If you want a live output of the memory contents instead of manually synchronizing, click . Continuous read is analogous to using the Signal Tap logic analyzer in continuous acquisition, with the memory values appearing as words in the Hex Editor instead of toggling waveforms.

Note: For Intel Stratix 10 and Intel Agilex 7 devices only, ISMCE logic can perform read and write operations only when the design logic is idle. If the design logic attempts a write or an address change operation, the design logic prevails, and the ISMCE operation times out. An error message lets you know that the memory connected to the In-System Memory Content Editor instance is in use, and memory content is not updated.

Figure 105. Hex Editor After Manually Editing Content



Related Information

- Read Information from In-System Memory Commands (Processing Menu)
  In *Intel Quartus Prime Help*
- Stop In-System Memory Analysis Command (Processing Menu)
  In *Intel Quartus Prime Help*

5.7. Editing Memory Contents with the Hex Editor Pane

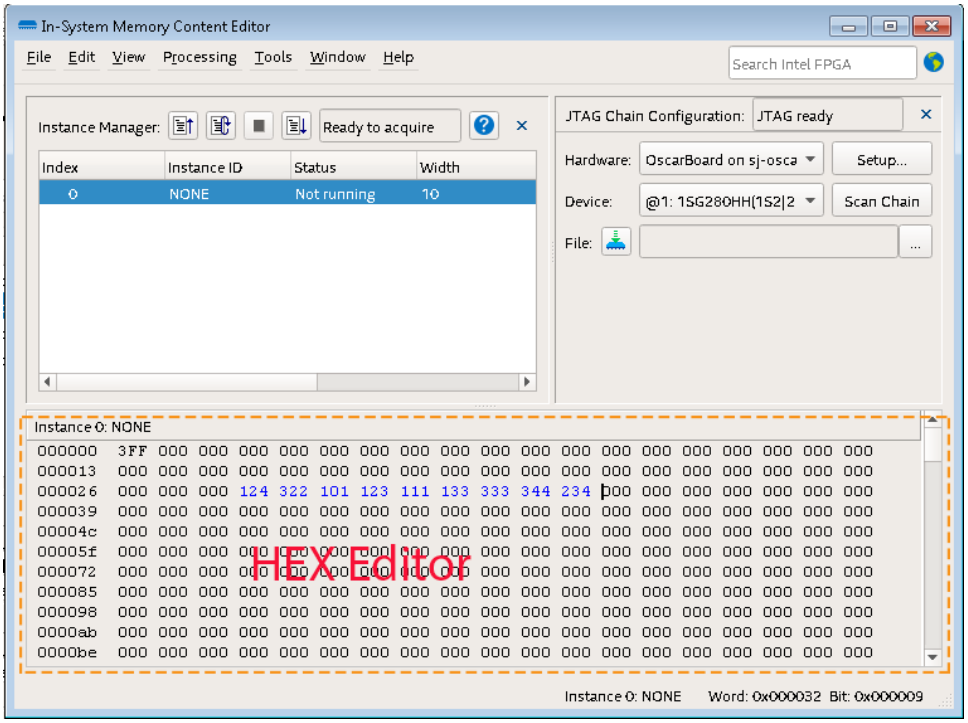
You can edit the contents of instances by typing values directly into the **Hex Editor** pane.

Black content on the **Hex Editor** pane means that the value read is the same as last synchronization.

1. Type content on the pane.
   The **Hex Editor** displays in blue changed content that has not been synchronized to the device.
2. Click to synchronize the content to the device.

Note: For Intel Stratix 10 and Intel Agilex 7 devices only, ISMCE logic can perform read and write operations only when the design logic is idle. If the design logic attempts a write or an address change operation, the design logic prevails, and the ISMCE operation times out. An error message lets you know that the memory connected to the In-System Memory Content Editor instance is in use, and reports the number of successful writes before the design logic requested access to the memory.

Figure 106. Hex Editor After Manually Editing Content



Related Information

- Custom Fill Dialog Box
  In *Intel Quartus Prime Help*
- Write Information to In-System Memory Commands (Processing Menu)
  In *Intel Quartus Prime Help*
- Go To Dialog Box
  In *Intel Quartus Prime Help*
- Select Range Dialog Box
  In *Intel Quartus Prime Help*

5.8. Importing and Exporting Memory Files

The In-System Memory Content Editor allows you to import and export data values for memories that are runtime modifiable. Importing from a data file enables you to quickly load an entire memory image. Exporting to a data file allows you to save the contents of the memory for future use.

You can import or export files in `hex` or `mif` formats.

1. To import a file, click **Edit ➤ Import Data from File...**, and then select the file to import.

   If the file is not compatible, unexpected data appears in the Hex Editor.
2. To export memory contents to a file, click **Edit ➤ Export Data to File...**, and then specify the name.

Related Information

- Import Data
  In *Intel Quartus Prime Help*
- Export Data
  In *Intel Quartus Prime Help*
- Hexadecimal (Intel-Format) File (.hex) Definition
  In *Intel Quartus Prime Help*
- Memory Initialization File (.mif) Definition
  In *Intel Quartus Prime Help*

5.9. Access Two or More Devices

If you have more than one device with in-system configurable memories or constants in a JTAG chain, you can launch multiple In-System Memory Content Editors within the Intel Quartus Prime software to access the memories and constants in each of the devices. Each window of the In-System Memory Content Editor can access the memories and constants of a single device.

5.10. Scripting Support

The Intel Quartus Prime software allows you to perform runtime modification of memories and constants in scripted flows.

You can enable memory and constant instances to be runtime modifiable from the HDL code. Additionally, the In-System Memory Content Editor supports reading and writing of memory contents via Tcl commands from the `insystem_memory_edit` package.

Related Information

- Tcl Scripting
  In *Intel Quartus Prime Pro Edition User Guide: Scripting*
- Command Line Scripting
  In *Intel Quartus Prime Pro Edition User Guide: Scripting*

5.10.1. The insystem_memory_edit Tcl Package

The **::quartus::insystem_memory_edit** Tcl package contains the set of Tcl functions for reading and editing the contents of memory in an Intel FPGA device using the In-System Memory Content Editor. The `quartus_stp` and `quartus_stp_tcl` command line executables load this package by default.

For the most up-to-date information about the **::quartus::insystem_memory_edit**, refer to the *Intel Quartus Prime Pro Edition User Guide: Scripting*.

Related Information

Intel Quartus Prime Pro Edition User Guide: Scripting
In *Intel Quartus Prime Pro Edition User Guide: Scripting*

5.10.1.1. Getting Information about the insystem_memory_edit Package

You can also get information on the `insystem_memory_edit` package directly from the command line:

- For general information about the package, type:

```
quartus_stp --tcl_eval help -pkg insystem_memory_edit
```

- For information about a command in the package, type:

```
quartus_stp --tcl_eval help -cmd <command_name>
```

5.11. In-System Modification of Memory and Constants Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2023.04.03	23.1	• Updated product family name to "Intel Agilex 7."
2022.12.12	22.4	• Added support for the Intel Agilex 7 device family.
2018.05.07	18.0.0	• Added support for the Intel Stratix 10 device family. • Removed obsolete example.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	• Dita conversion. • Removed references to megafunction and replaced with IP core.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.3	Template update.
December 2010	10.0.2	Changed to new document template. No change to content.
August 2010	10.0.1	Corrected links
July 2010	10.0.0	• Inserted links to Intel Quartus Prime Help • Removed Reference Documents section
November 2009	9.1.0	• Delete references to APEX devices • Style changes
March 2009	9.0.0	No change to content
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	• Added reference to Section V. In-System Debugging in volume 3 of the Intel Quartus Prime Handbook on page 16-1 • Removed references to the Mercury device, as it is now considered to be a “Mature” device • Added links to referenced documents throughout document • Minor editorial updates

6. Design Debugging Using In-System Sources and Probes

The Signal Tap Logic Analyzer and Signal Probe allow you to read or "tap" internal logic signals during run time as a way to debug your logic design.

Traditional debugging techniques often involve using an external pattern generator to exercise the logic and a logic analyzer to study the output waveforms during run time.

You can make the debugging cycle more efficient when you can drive any internal signal manually within your design, which allows you to perform the following actions:

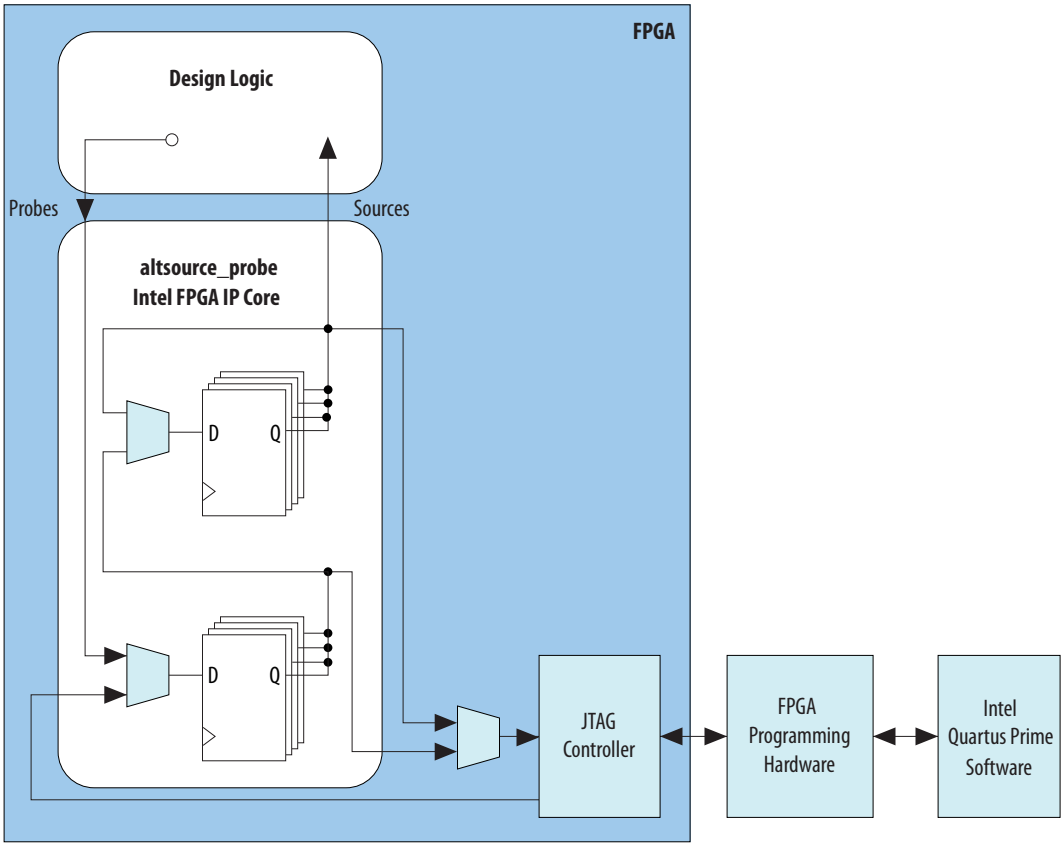
- Force the occurrence of trigger conditions set up in the Signal Tap Logic Analyzer
- Create simple test vectors to exercise your design without using external test equipment
- Dynamically control run time control signals with the JTAG chain

The In-System Sources and Probes Editor in the Intel Quartus Prime software extends the portfolio of verification tools, and allows you to easily control any internal signal and provides you with a completely dynamic debugging environment. Coupled with either the Signal Tap Logic Analyzer or Signal Probe, the In-System Sources and Probes Editor gives you a powerful debugging environment in which to generate stimuli and solicit responses from your logic design.

The Virtual JTAG IP core and the In-System Memory Content Editor also give you the capability to drive virtual inputs into your design. The Intel Quartus Prime software offers a variety of on-chip debugging tools.

The In-System Sources and Probes Editor consists of the ALTSOURCE_PROBE IP core and an interface to control the ALTSOURCE_PROBE IP core instances during run time. Each ALTSOURCE_PROBE IP core instance provides you with source output ports and probe input ports, where source ports drive selected signals and probe ports sample selected signals. When you compile your design, the ALTSOURCE_PROBE IP core sets up a register chain to either drive or sample the selected nodes in your logic design. During run time, the In-System Sources and Probes Editor uses a JTAG connection to shift data to and from the ALTSOURCE_PROBE IP core instances. The figure shows a block diagram of the components that make up the In-System Sources and Probes Editor.

Figure 107. In-System Sources and Probes Editor Block Diagram



The ALTSOURCE_PROBE IP core hides the detailed transactions between the JTAG controller and the registers instrumented in your design to give you a basic building block for stimulating and probing your design. Additionally, the In-System Sources and Probes Editor provides single-cycle samples and single-cycle writes to selected logic nodes. You can use this feature to input simple virtual stimuli and to capture the current value on instrumented nodes. Because the In-System Sources and Probes Editor gives you access to logic nodes in your design, you can toggle the inputs of low-level components during the debugging process. If used in conjunction with the Signal Tap Logic Analyzer, you can force trigger conditions to help isolate your problem and shorten your debugging process.

The In-System Sources and Probes Editor allows you to easily implement control signals in your design as virtual stimuli. This feature can be especially helpful for prototyping your design, such as in the following operations:

- Creating virtual push buttons
- Creating a virtual front panel to interface with your design
- Emulating external sensor data
- Monitoring and changing run time constants on the fly

The In-System Sources and Probes Editor supports Tcl commands that interface with all your ALTSOURCE_PROBE IP core instances to increase the level of automation.

Related Information

System Debugging Tools
For an overview and comparison of all the tools available in the Intel Quartus Prime software on-chip debugging tool suite

6.1. Hardware and Software Requirements

The following components are required to use the In-System Sources and Probes Editor:

- Intel Quartus Prime software

or

- Intel Quartus Prime Lite Edition
- Download Cable (USB-Blaster™ download cable or ByteBlaster™ cable)
- Intel FPGA development kit or user design board with a JTAG connection to device under test

The In-System Sources and Probes Editor supports the following device families:

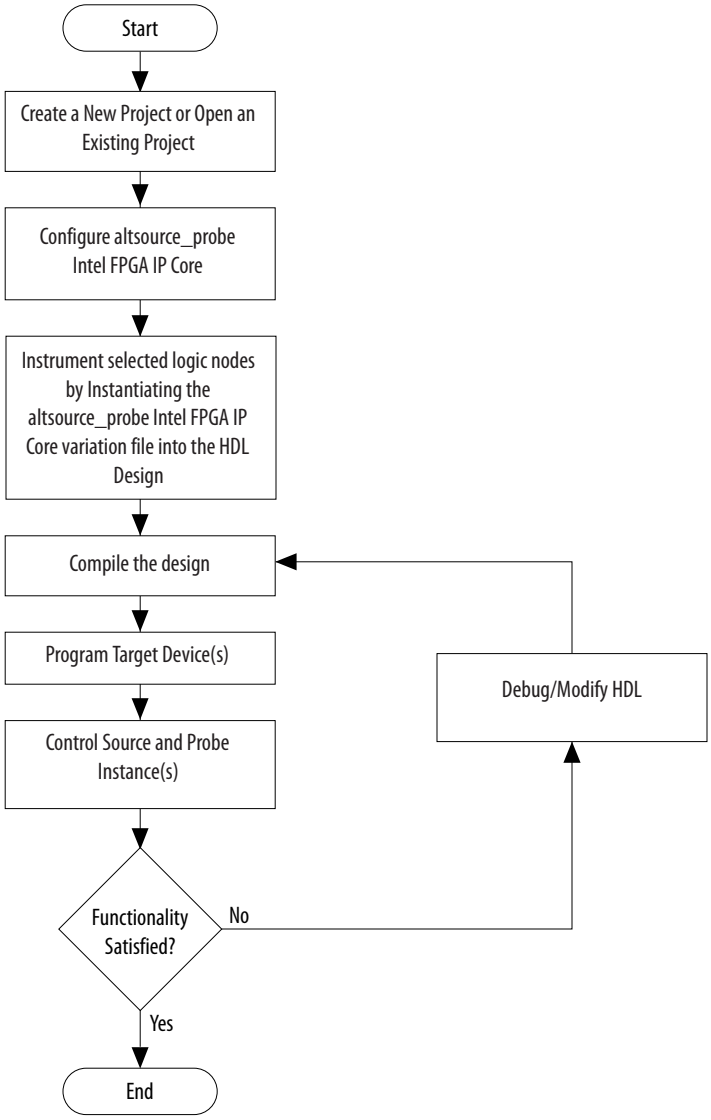
- Arria series
- Stratix series
- Cyclone series
- MAX® series

6.2. Design Flow Using the In-System Sources and Probes Editor

The In-System Sources and Probes Editor supports an RTL flow. Signals that you want to view in the In-System Sources and Probes editor are connected to an instance of the In-System Sources and Probes IP core.

After you compile the design, you can control each instance via the **In-System Sources and Probes Editor** pane or via a Tcl interface.

Figure 108. FPGA Design Flow Using the In-System Sources and Probes Editor



6.2.1. Instantiating the In-System Sources and Probes IP Core

To instantiate the In-System Sources and Probes IP core in a design:

1. In the IP Catalog (**Tools ➤ IP Catalog**), type `In-System Sources and Probes`.
2. Double-click **In-System Sources and Probes** to open the parameter editor.
3. Specify a name for the IP variation.
4. Specify the parameters for the IP variation.

The IP core supports up to 512 bits for each source, and design can include up to 128 instances of this IP core.

5. Click **Generate** or **Finish** to generate IP core synthesis and simulation files matching your specifications.
6. Using the generated template, instantiate the In-System Sources and Probes IP core in your design.

Note: The In-System Sources and Probes Editor does not support simulation. Remove the In-System Sources and Probes IP core before you create a simulation netlist.

6.2.2. In-System Sources and Probes IP Core Parameters

Use the template to instantiate the variation file in your design.

Table 34. In-System Sources and Probes IP Port Information

Port Name	Required?	Direction	Comments
`probe[]`	No	Input	The outputs from your design.
`source_clk`	No	Input	Source Data is written synchronously to this clock. This input is required if you turn on **Source Clock** in the **Advanced Options** box in the parameter editor.
`source_ena`	No	Input	Clock enable signal for `source_clk`. This input is required if specified in the **Advanced Options** box in the parameter editor.
`source[]`	No	Output	Used to drive inputs to user design.

You can include up to 128 instances of the in-system sources and probes IP core in your design, if your device has available resources. Each instance of the IP core uses a pair of registers per signal for the width of the widest port in the IP core. Additionally, there is some fixed overhead logic to accommodate communication between the IP core instances and the JTAG controller. You can also specify an additional pair of registers per source port for synchronization.

6.3. Compiling the Design

When you compile your design that includes the In-System Sources and Probes IP core, the In-System Sources and Probes and SLD Hub Controller IP core are added to your compilation hierarchy automatically. These IP cores provide communication between the JTAG controller and your instrumented logic.

You can modify the number of connections to your design by editing the In-System Sources and Probes IP core. To open the design instance you want to modify in the parameter editor, double-click the instance in the Project Navigator. You can then modify the connections in the HDL source file. You must recompile your design after you make changes.

6.4. Running the In-System Sources and Probes Editor

The In-System Sources and Probes Editor gives you control over all ALTSOURCE_PROBE IP core instances within your design. The editor allows you to view all available run time controllable instances of the ALTSOURCE_PROBE IP core in your design, provides a push-button interface to drive all your source nodes, and provides a logging feature to store your probe and source data.

To run the In-System Sources and Probes Editor:

- On the **Tools** menu, click **In-System Sources and Probes Editor.**

6.4.1. In-System Sources and Probes Editor GUI

The In-System Sources and Probes Editor contains three panes:

- **JTAG Chain Configuration**—Allows you to specify programming hardware, device, and file settings that the In-System Sources and Probes Editor uses to program and acquire data from a device.
- **Instance Manager**—Displays information about the instances generated when you compile a design, and allows you to control data that the In-System Sources and Probes Editor acquires.
- **In-System Sources and Probes Editor**—Logs all data read from the selected instance and allows you to modify source data that is written to your device.

When you use the In-System Sources and Probes Editor, you do not need to open an Intel Quartus Prime software project. The In-System Sources and Probes Editor retrieves all instances of the ALTSOURCE_PROBE IP core by scanning the JTAG chain and sending a query to the device selected in the **JTAG Chain Configuration** pane. You can also use a previously saved configuration to run the In-System Sources and Probes Editor.

Each **In-System Sources and Probes Editor** pane can access the ALTSOURCE_PROBE IP core instances in a single device. If you have more than one device containing IP core instances in a JTAG chain, you can launch multiple **In-System Sources and Probes Editor** panes to access the IP core instances in each device.

6.4.2. Programming Your Device With JTAG Chain Configuration

After you compile your project, you must configure your FPGA before you use the In-System Sources and Probes Editor.

To configure a device to use with the In-System Sources and Probes Editor, perform the following steps:

1. Open the In-System Sources and Probes Editor.
2. In the **JTAG Chain Configuration** pane, point to **Hardware,** and then select the hardware communications device. You may be prompted to configure your hardware; in this case, click **Setup**.
3. From the **Device** list, select the FPGA device to which you want to download the design (the device may be automatically detected). You may need to click **Scan Chain** to detect your target device.
4. In the **JTAG Chain Configuration** pane, click to browse for the SRAM Object File (**.sof**) that includes the In-System Sources and Probes instance or instances. (The **.sof** may be automatically detected).
5. Click **Program Device** to program the target device.

6.4.3. Instance Manager

The **Instance Manager** pane provides a list of all ALTSOURCE_PROBE instances in the design, and allows you to configure data acquisition.

The **Instance Manager** pane contains the following buttons and sub-panes:

- **Read Probe Data**—Samples the probe data in the selected instance and displays the probe data in the **In-System Sources and Probes Editor** pane.
- **Continuously Read Probe Data**—Continuously samples the probe data of the selected instance and displays the probe data in the **In-System Sources and Probes Editor** pane; you can modify the sample rate via the **Probe read interval** setting.
- **Stop Continuously Reading Probe Data**—Cancels continuous sampling of the probe of the selected instance.
- **Read Source Data**—Reads the data of the sources in the selected instances.
- **Probe Read Interval**—Displays the sample interval of all the In-System Sources and Probe instances in your design; you can modify the sample interval by clicking **Manual**.
- **Event Log**—Controls the event log that appears in the **In-System Sources and Probes Editor** pane.
- **Write Source Data**—Allows you to manually or continuously write data to the system.

Beside each entry, the **Instance Manager** pane displays the instance status. The possible instance statuses are **Not running Offloading data**, **Updating data**, and **Unexpected JTAG communication error**.

6.4.4. In-System Sources and Probes Editor Pane

The **In-System Sources and Probes Editor** pane allows you to view data from all sources and probes in your design.

The data is organized according to the index number of the instance. The editor provides an easy way to manage your signals, and allows you to rename signals or group them into buses. All data collected from in-system source and probe nodes is recorded in the event log and you can view the data as a timing diagram.

6.4.4.1. Reading Probe Data

You can read data by selecting the ALTSOURCE_PROBE instance in the **Instance Manager** pane and clicking **Read Probe Data**.

This action produces a single sample of the probe data and updates the data column of the selected index in the **In-System Sources and Probes Editor** pane. You can save the data to an event log by turning on the **Save data to event log** option in the **Instance Manager** pane.

If you want to sample data from your probe instance continuously, in the **Instance Manager** pane, click the instance you want to read, and then click **Continuously read probe data**. While reading, the status of the active instance shows **Unloading**. You can read continuously from multiple instances.

You can access read data with the shortcut menus in the **Instance Manager** pane.

To adjust the probe read interval, in the **Instance Manager** pane, turn on the **Manual** option in the **Probe read interval** sub-pane, and specify the sample rate in the text field next to the **Manual** option. The maximum sample rate depends on your computer setup. The actual sample rate is shown in the **Current interval** box. You can adjust the event log window buffer size in the **Maximum Size** box.

6.4.4.2. Writing Data

To modify the source data you want to write into the ALTSOURCE_PROBE instance, click the name field of the signal you want to change. For buses of signals, you can double-click the data field and type the value you want to drive out to the ALTSOURCE_PROBE instance. The In-System Sources and Probes Editor stores the modified source data values in a temporary buffer.

Modified values that are not written out to the ALTSOURCE_PROBE instances appear in red. To update the ALTSOURCE_PROBE instance, highlight the instance in the **Instance Manager** pane and click **Write source data**. The **Write source data** function is also available via the shortcut menus in the **Instance Manager** pane.

The In-System Sources and Probes Editor provides the option to continuously update each ALTSOURCE_PROBE instance. Continuous updating allows any modifications you make to the source data buffer to also write immediately to the ALTSOURCE_PROBE instances. To continuously update the ALTSOURCE_PROBE instances, change the **Write source data** field from **Manually** to **Continuously**.

6.4.4.3. Organizing Data

The **In-System Sources and Probes Editor** pane allows you to group signals into buses, and also allows you to modify the display options of the data buffer.

To create a group of signals, select the node names you want to group, right-click and select **Group**. You can modify the display format in the Bus Display Format and the Bus Bit order shortcut menus.

The **In-System Sources and Probes Editor** pane allows you to rename any signal. To rename a signal, double-click the name of the signal and type the new name.

The event log contains a record of the most recent samples. The buffer size is adjustable up to 128k samples. The time stamp for each sample is logged and is displayed above the event log of the active instance as you move your pointer over the data samples.

You can save the changes that you make and the recorded data to a Sources and Probes File (**.spf**). To save changes, on the File menu, click **Save**. The file contains all the modifications you made to the signal groups, as well as the current data event log.

6.5. Tcl interface for the In-System Sources and Probes Editor

To support automation, the In-System Sources and Probes Editor supports the procedures described in this chapter in the form of Tcl commands. The Tcl package for the In-System Sources and Probes Editor is included by default when you run **quartus_stp**.

The Tcl interface for the In-System Sources and Probes Editor provides a powerful platform to help you debug your design. The Tcl interface is especially helpful for debugging designs that require toggling multiple sets of control inputs. You can combine multiple commands with a Tcl script to define a custom command set.

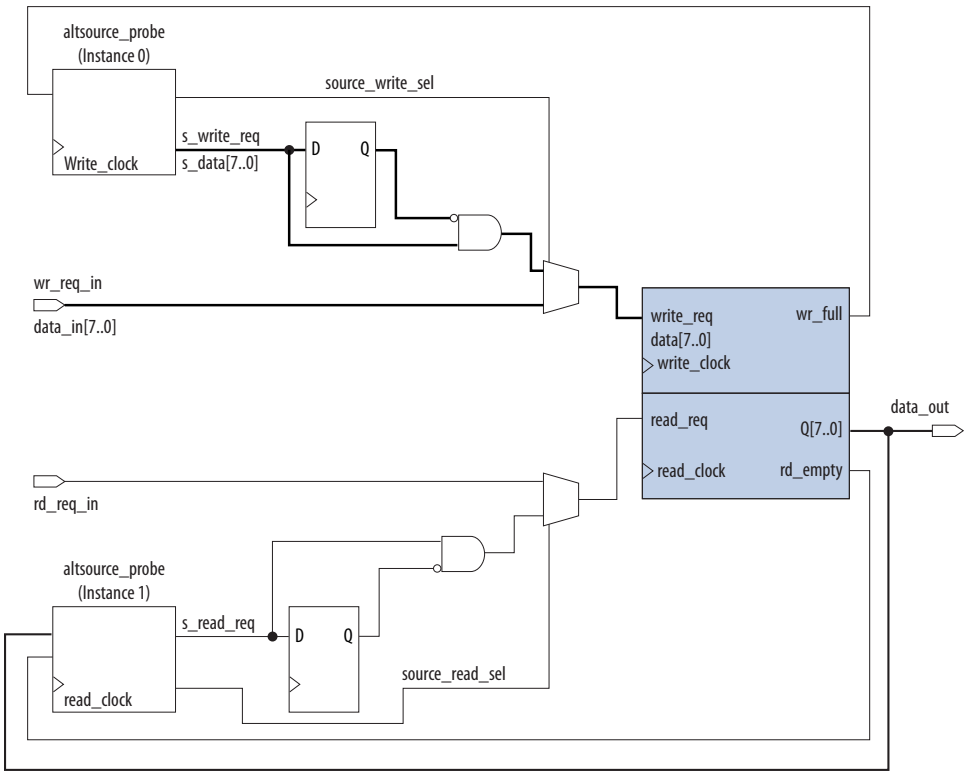
Table 35. In-System Sources and Probes Tcl Commands

Command	Argument	Description
`start_insystem_source_probe`	`-device_name<`*device name*`> -hardware_name <`*hardware name*`>`	Opens a handle to a device with the specified hardware. Call this command before starting any transactions.
`get_insystem_source_probe_instance_info`	`-device_name <`*device name*`> -hardware_name <`*hardware name*`>`	Returns a list of all ALTSOURCE_PROBE instances in your design. Each record returned is in the following format: {*<instance Index>*, *<source width>*, *<probe width>*, *<instance name>*}
`read_probe_data`	`-instance_index <`*instance_index*`> -value_in_hex` (optional)	Retrieves the current value of the probe. A string is returned that specifies the status of each probe, with the MSB as the left-most bit.
`read_source_data`	`-instance_index <`*instance_index*`> -value_in_hex` (optional)	Retrieves the current value of the sources. A string is returned that specifies the status of each source, with the MSB as the left-most bit.
`write_source_data`	`-instance_index <`*instance_index*`> -value <`*value*`> -value_in_hex` (optional)	Sets the value of the sources. A binary string is sent to the source ports, with the MSB as the left-most bit.
`end_insystem_source_probe`	None	Releases the JTAG chain. Issue this command when all transactions are finished.

The example shows an excerpt from a Tcl script with procedures that control the ALTSOURCE_PROBE instances of the design as shown in the figure below. The example design contains a DCFIFO with ALTSOURCE_PROBE instances to read from and write to the DCFIFO. A set of control muxes are added to the design to control the flow of data to the DCFIFO between the input pins and the ALTSOURCE_PROBE instances. A pulse generator is added to the read request and write request control lines to guarantee a single sample read or write. The ALTSOURCE_PROBE instances, when used with the script in the example below, provide visibility into the contents of the FIFO by performing single sample write and read operations and reporting the state of the full and empty status flags.

Use the Tcl script in debugging situations to either empty or preload the FIFO in your design. For example, you can use this feature to preload the FIFO to match a trigger condition you have set up within the Signal Tap logic analyzer.

Figure 109. DCFIFO Example Design Controlled by Tcl Script




```
## Setup USB hardware  - assumes only USB Blaster is installed and
## an FPGA is the only device in the JTAG chain
set usb [lindex [get_hardware_names] 0]
set device_name [lindex [get_device_names -hardware_name $usb] 0]
## write procedure :  argument value is integer
proc write {value} {
global device_name usb
variable full
start_insystem_source_probe -device_name $device_name -hardware_name $usb
#read full flag
set full [read_probe_data -instance_index 0]
if {$full == 1} {end_insystem_source_probe
return "Write Buffer Full"
}
##toggle select line, drive value onto port, toggle enable
##bits 7:0 of instance 0 is S_data[7:0]; bit 8 = S_write_req;
##bit 9 = Source_write_sel
##int2bits is custom procedure that returns a bitstring from an integer
    ## argument
write_source_data -instance_index 0 -value /[int2bits [expr 0x200 | $value]]
write_source_data -instance_index 0 -value [int2bits [expr 0x300 | $value]]
##clear transaction
write_source_data -instance_index 0 -value 0
end_insystem_source_probe
}
proc read {} {
global device_name usb
variable empty
start_insystem_source_probe -device_name $device_name -hardware_name $usb
##read empty flag : probe port[7:0] reads FIFO output; bit 8 reads empty_flag
set empty [read_probe_data -instance_index 1]
if {[regexp {1........} $empty]} { end_insystem_source_probe
return "FIFO empty" }
## toggle select line for read transaction
```

```
## Source_read_sel = bit 0; s_read_reg = bit 1
## pulse read enable on DC FIFO
write_source_data -instance_index 1 -value 0x1 -value_in_hex
write_source_data -instance_index 1 -value 0x3 -value_in_hex
set x [read_probe_data -instance_index 1 ]
end_insystem_source_probe
return $x
}
```

Related Information

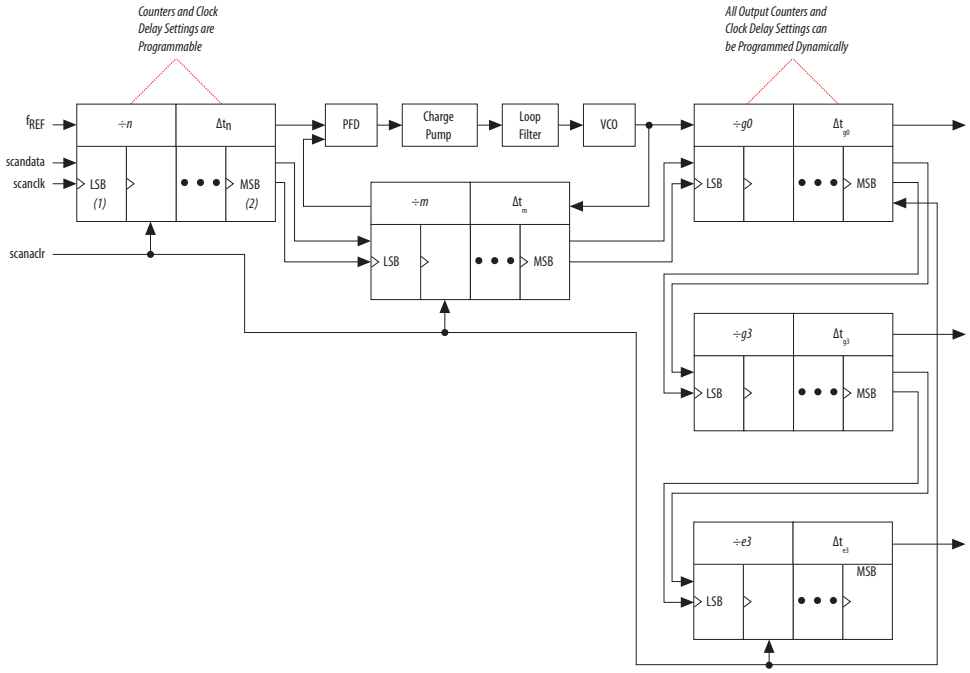
- Tcl Scripting in *Intel Quartus Prime Pro Edition User Guide: Scripting*
- Intel Quartus Prime Pro Edition Settings File Manual
- Command Line Scripting in *Intel Quartus Prime Pro Edition User Guide: Scripting*

6.6. Design Example: Dynamic PLL Reconfiguration

The In-System Sources and Probes Editor can help you create a virtual front panel during the prototyping phase of your design. You can create relatively simple, high functioning designs of in a short amount of time. The following PLL reconfiguration example demonstrates how to use the In-System Sources and Probes Editor to provide a GUI to dynamically reconfigure a Stratix PLL.

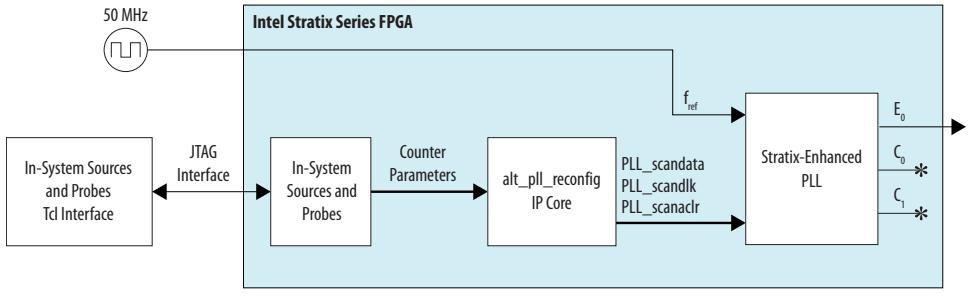
Stratix PLLs allow you to dynamically update PLL coefficients during run time. Each enhanced PLL within the Stratix device contains a register chain that allows you to modify the pre-scale counters (m and n values), output divide counters, and delay counters. In addition, the ALTPLL_RECONFIG IP core provides an easy interface to access the register chain counters. The ALTPLL_RECONFIG IP core provides a cache that contains all modifiable PLL parameters. After you update all the PLL parameters in the cache, the ALTPLL_RECONFIG IP core drives the PLL register chain to update the PLL with the updated parameters. The figure shows a Stratix-enhanced PLL with reconfigurable coefficients.

Figure 110. Stratix-Enhanced PLL with Reconfigurable Coefficients



The following design example uses an ALTSOURCE_PROBE instance to update the PLL parameters in the ALTPLL_RECONFIG IP core cache. The ALTPLL_RECONFIG IP core connects to an enhanced PLL in a Stratix FPGA to drive the register chain containing the PLL reconfigurable coefficients. This design example uses a Tcl/Tk script to generate a GUI where you can enter in new m and n values for the enhanced PLL. The Tcl script extracts the m and n values from the GUI, shifts the values out to the ALTSOURCE_PROBE instances to update the values in the ALTPLL_RECONFIG IP core cache, and asserts the reconfiguration signal on the ALTPLL_RECONFIG IP core. The reconfiguration signal on the ALTPLL_RECONFIG IP core starts the register chain transaction to update all PLL reconfigurable coefficients.

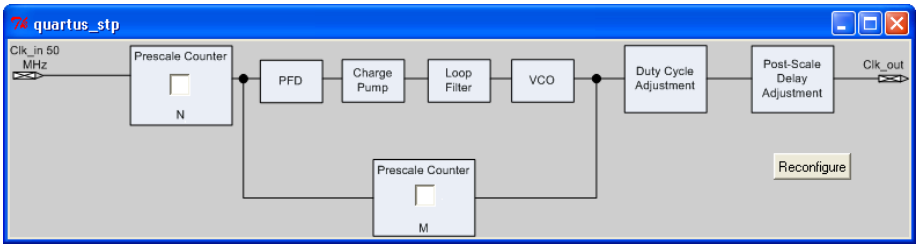
Figure 111. Block Diagram of Dynamic PLL Reconfiguration Design Example



This design example was created using a Nios II Development Kit, Stratix Edition. The file `sourceprobe_DE_dynamic_pll.zip` contains all the necessary files for running this design example, including the following:

- `Readme.txt`—A text file that describes the files contained in the design example and provides instructions about running the Tk GUI shown in the figure below.
- `Interactive_Reconfig.qar`—The archived Intel Quartus Prime project for this design example.

Figure 112. Interactive PLL Reconfiguration GUI Created with Tk and In-System Sources and Probes Tcl Package



Related Information

On-chip Debugging Design Examples
to download the In-System Sources and Probes Example

6.7. Design Debugging Using In-System Sources and Probes Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2023.07.31	22.1	• Corrected incorrect space characters in *Tcl interface for the In-System Sources and Probes Editor*.
2022.07.08	22.1	• Fixed broken link in "Design Example: Dynamic PLL Reconfiguration".
2019.06.11	18.1.0	Rebranded megafunction to Intel FPGA IP core
2018.05.07	18.0.0	Added details on finding the In-System Sources and Probes in the IP Catalog.
2016.10.31	16.1.0	Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	Updated formatting.
June 2012	12.0.0	Removed survey link.
November 2011	10.1.1	Template update.
December 2010	10.1.0	Minor corrections. Changed to new document template.
July 2010	10.0.0	Minor corrections.
November 2009	9.1.0	• Removed references to obsolete devices. • Style changes.

continued...

Document Version	Intel Quartus Prime Version	Changes
March 2009	9.0.0	No change to content.
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	• Documented that this feature does not support simulation on page 17–5 • Updated Figure 17–8 for Interactive PLL reconfiguration manager • Added hyperlinks to referenced documents throughout the chapter • Minor editorial updates

7. Analyzing and Debugging Designs with System Console

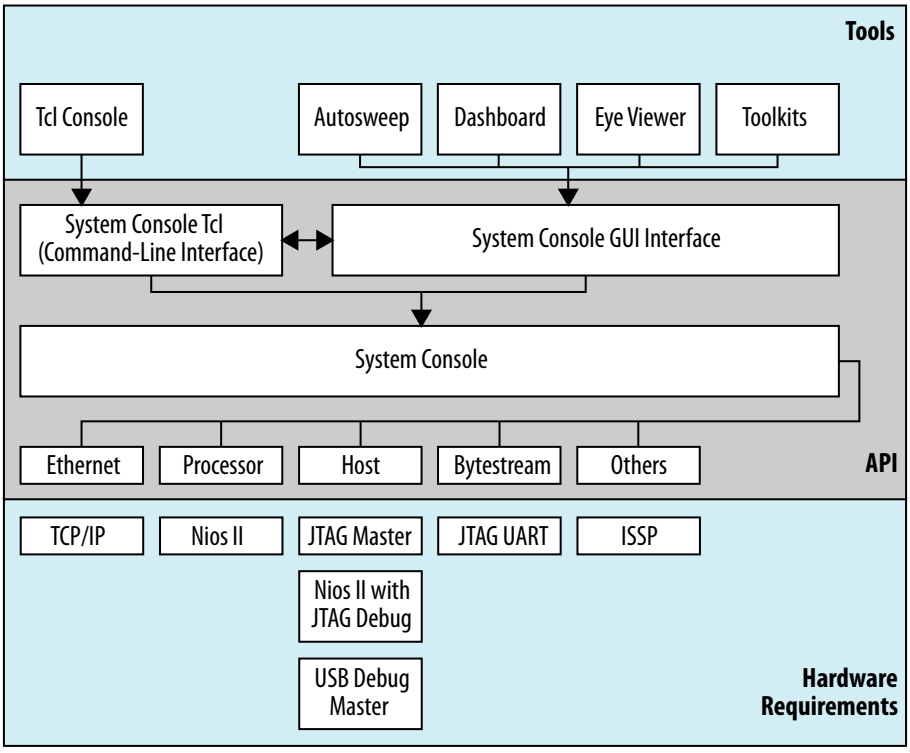
7.1. Introduction to System Console

System Console provides visibility into your design and allows you to perform system-level debug on an FPGA at run-time. System Console performs tests on debug-enabled Intel FPGA IP. A variety of debug services provide read and write access to elements in your design.

- Perform board bring-up with finalized or partially complete designs.
- Automate run-time verification through scripting across multiple devices.
- Debug transceiver links, memory interfaces, and Ethernet interfaces.
- Integrate your debug IP into the debug platform.
- Perform system verification with MATLAB and Simulink.

Figure 113. System Console Tools

The System Console API supports services that access your design in operation.



System Console also provides the hardware debugging infrastructure to support operation and customization of debugging "toolkits." Toolkits are small applications that you can use to perform system-level debug of such elements as external memory interfaces, Ethernet interfaces, PCI Express interfaces, transceiver PHY interfaces, and various other debugging functions. The Intel Quartus Prime software includes the Available System Debugging Toolkits on page 175. For advanced users, System Console also supports Tcl commands that allow you to define and operate your own custom toolkits, as *System Console and Toolkit Tcl Command Reference Manual* describes.

Related Information

- System Console Online Training
- System Console and Toolkit Tcl Command Reference Manual

7.1.1. IP Cores that Interact with System Console

System Console runs on your host computer and communicates with your running design through debug agents. Debug agents are the soft-logic embedded in some IP cores that enable debug communication with the host computer.

You can instantiate debug IP cores using the Intel Quartus Prime software IP Catalog and IP parameter editor. Some IP cores are enabled for debug by default, while you must enable debug for other IP cores through options in the parameter editor. Some debug agents have multiple purposes.

When you include debug-enabled IP cores in your design, you can access large portions of the design running on hardware for debugging purposes. Debug agents allow you to read and write to memory and alter peripheral registers from the tool.

Services associated with debug agents in the running design can open and close as needed. System Console determines the communication protocol with the debug agent. The communication protocol determines the best board connection to use for command and data transmission.

The Programmable SRAM Object File (`.sof`) that the Intel Quartus Prime Assembler generates for device programming provides the System Console with channel communication information. When you open System Console from the Intel Quartus Prime software GUI, with a project open that includes a `.sof`, System Console automatically finds and links to the device(s) it detects. When you open System Console without an open project, or with an unrelated project open, you can manually load the `.sof` file that you want, and then the design linking occurs automatically if the device(s) match.

Related Information

- Available System Debugging Toolkits on page 175
- WP-01170 System-Level Debugging and Monitoring of FPGA Designs

7.1.2. Services Provided through Debug Agents

By adding the appropriate debug agent to your design, System Console services can use the associated capabilities of the debug agent.

Table 36. Common Services for System Console

Service	Function	Debug Agent Providing Service
host	Access a memory-mapped agent connected to the host interface.	• Nios II with debug • JTAG to Avalon Master Bridge • USB Debug Master
agent	Allows a host component to access a single agent without needing to know the location of the agent in the host's memory map. Any agent that is accessible to a System Console host can provide this service.	• Nios II with debug • JTAG to Avalon Master Bridge • USB Debug Master If an SRAM Object File (`.sof`) is loaded, then agents accessed by a debug host provide the agent service.
issp	The In-System Sources and Probes (ISSP) service provides scriptable access to the In-System Sources and Probes Intel FPGA IP for generating stimuli and soliciting responses from your logic design.	In-System Sources and Probes Intel FPGA IP
processor	• Start, stop, or step the processor. • Read and write processor registers.	Nios II with debug
JTAG UART	The JTAG UART is an Avalon memory mapped agent that you can use in conjunction with System Console to send and receive byte streams.	JTAG UART

Note: The following debug agent IP cores in the IP Catalog do not support VHDL simulation generation in the current version of the Intel Quartus Prime software:

- JTAG Debug Link
- JTAG Hub Controller System
- USB Debug Link

7.1.3. System Console Debugging Flow

The System Console debugging flow includes the following steps:

1. Add debug-enabled Intel FPGA IP to your design.
2. Compile the design.
3. Connect to a board and program the FPGA.
4. Start System Console.
5. Locate and open a System Console service.
6. Perform debug operations with the service.
7. Close the service.

Related Information

- Starting System Console on page 162
- Launching a Toolkit in System Console on page 174
- Using System Console Services on page 178
- Running System Console in Command-Line Mode on page 193

7.2. Starting System Console

You can use any of the following methods to start System Console:

- To start System Console from the Intel Quartus Prime software GUI:

  Click **Tools ➤ System Debugging Tools ➤ System Console**.

  Or

  Click **Tools ➤ System Debugging Tools ➤ System Debugging Toolkits**.
- To start System Console from Platform Designer:

  Click **Tools ➤ System Console**
- To start Stand-Alone System Console:
  1. Navigate to the **Download Center** page, click **Additional Software**, and download and install Intel Quartus Prime Pro Edition Programmer and Tools.
  2. On the Windows Start menu, click **All Programs ➤ Intel FPGA <*version*> ➤ Programmer and Tools ➤ System Console**.
- To start System Console from a Nios II Command Shell:
  1. On the Windows Start menu, click **All Programs ➤ Intel ➤ Nios II EDS <*version*> ➤ Nios II<*version*> ➤ Command Shell**.
  2. Type `system-console --project_dir=<project directory>` and specify a directory that contains `.qsf` or `.sof` files.

     Note: Type `--help` for System Console help.

7.2.1. Customizing System Console Startup

You can customize your System Console startup environment, as follows:

- Add commands to the `system_console_rc` configuration file located at:
  - `<$HOME>/system_console/system_console_rc.tcl`

  The file in this location is the user configuration file, which only affects the owner of the home directory.
- Specify your own design startup configuration file with the command-line argument `--rc_script=<path_to_script>`, when you launch System Console from the Nios II command shell.
- Use the `system_console_rc.tcl` file in combination with your custom `rc_script.tcl` file. In this case, the `system_console_rc.tcl` file performs System Console actions, and the `rc_script.tcl` file performs your debugging actions.

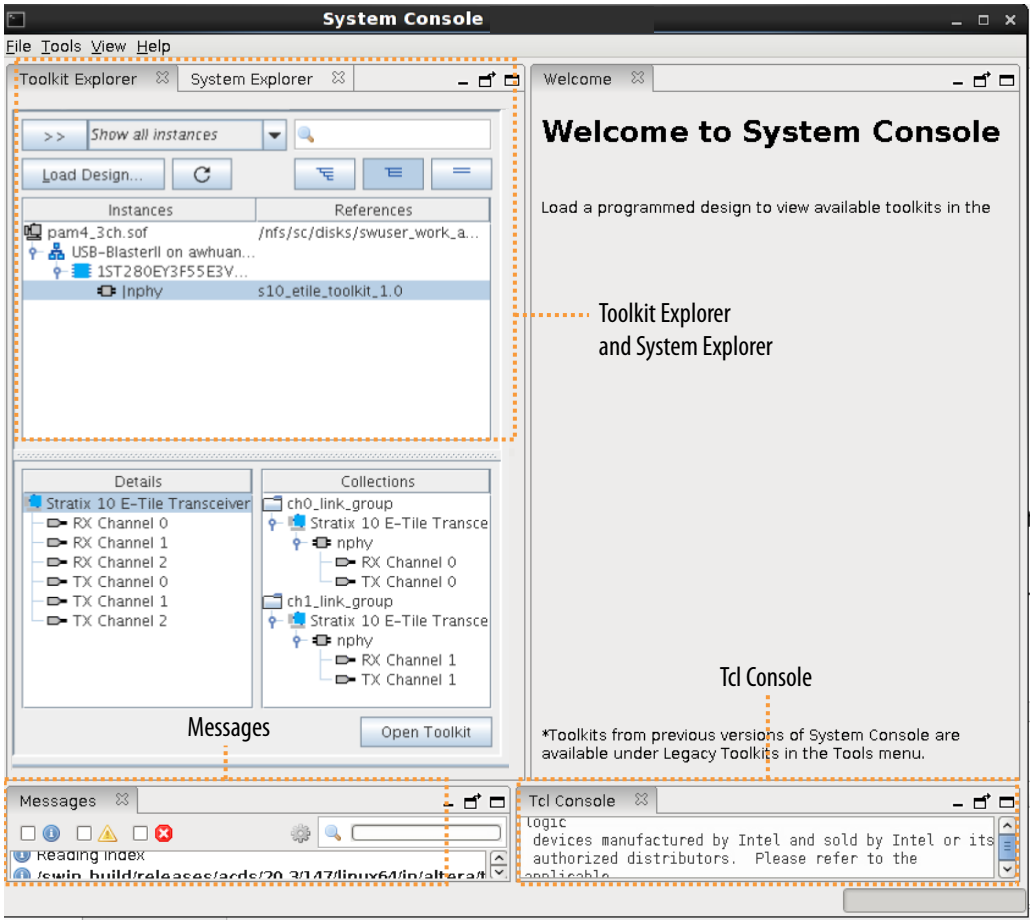
On startup, System Console automatically runs the Tcl commands in these files. The commands in the `system_console_rc.tcl` file run first, followed by the commands in the `rc_script.tcl` file.

7.3. System Console GUI

The System Console GUI consists of a main window with the following panes that allow you to interact with the design currently running on the FPGA device:

- **Toolkit Explorer**—displays all available toolkits and launches tools that use the System Console framework.
- **System Explorer**—displays a list of interactive instances in your design, including board connections, devices, designs, servers, and scripts.
- **Main View**—initially displays the welcome screen. All toolkits that you launch display in this view.
- **Tcl Console**—allows you to interact with your design through individual Tcl commands or by sourcing Tcl scripts, writing procedures, and using System Console APIs.
- **Messages**—displays status, warning, and error messages related to connections and debug actions.

Figure 114. System Console GUI



System Console GUI also provides the **Autosweep**, **Dashboard**, and **Eye Viewer** panes, that display as tabs in the **Main View**.

System Console Views on page 164

Toolkit Explorer Pane on page 171

System Explorer Pane on page 171

Customizing, Saving, and Resetting the System Console Layout on page 173

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.3.1. System Console Views

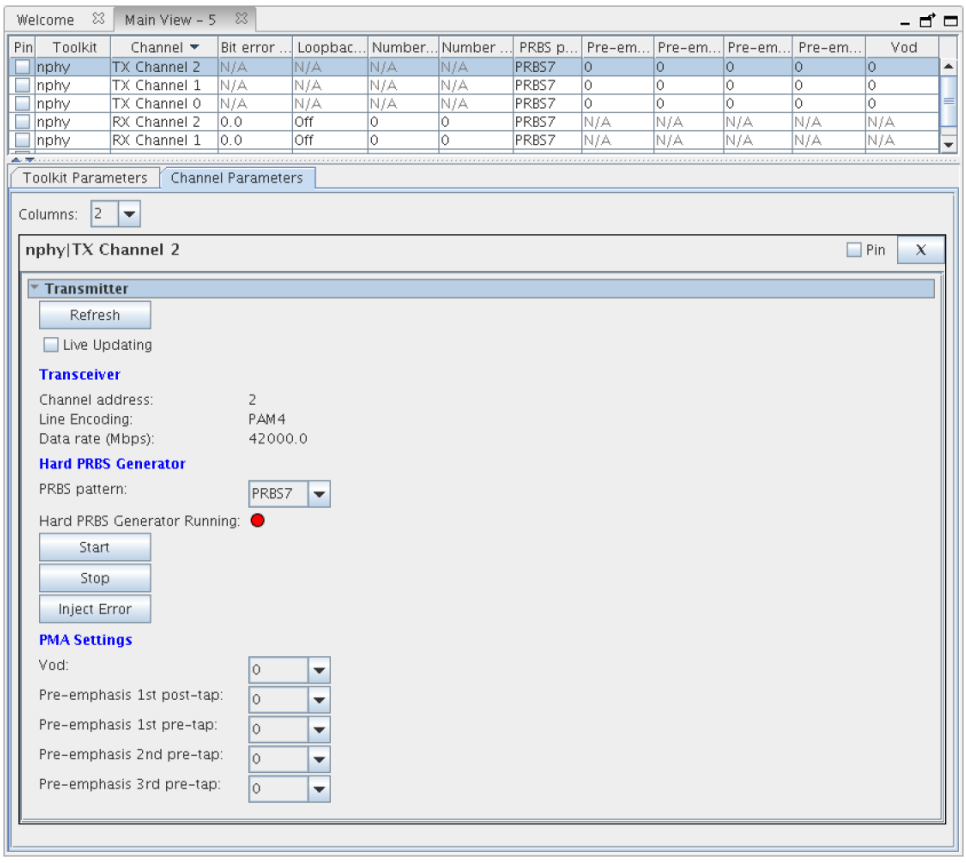
System Console provides the Main, Autosweep, Dashboard, and Eye Viewer views.

7.3.1.1. Main View

The **Main View** in System Console allows you to visualize certain parameter values of the IP that the toolkit targets. These parameter values can be static values from compile-time parameterization, or dynamic values read from the hardware (like reading from CSR registers) at run-time.

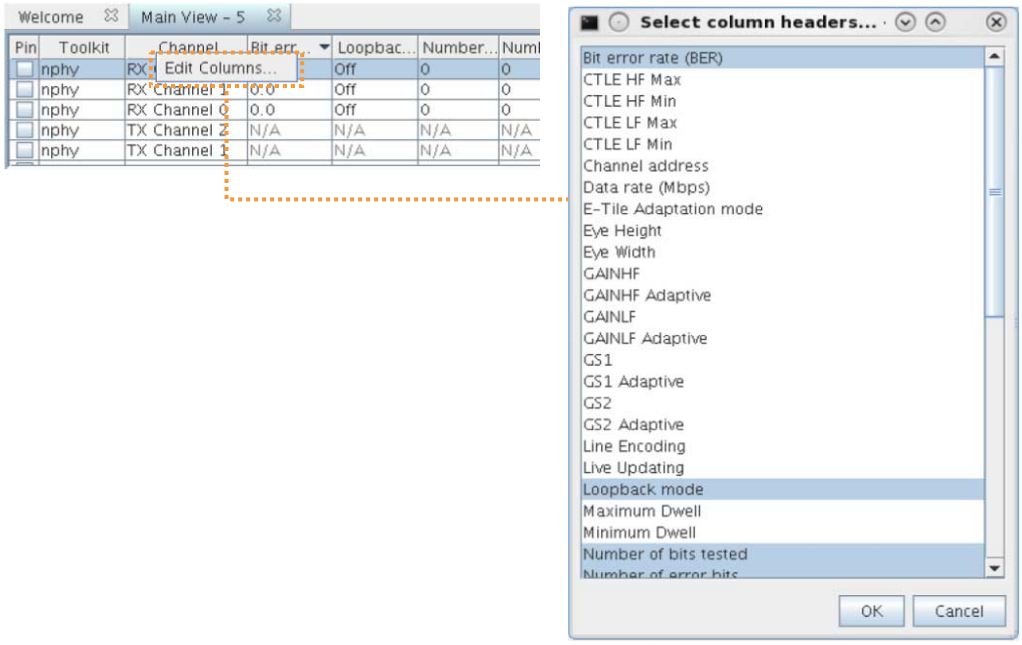
The **Main View** GUI controls allow you to control or configure the IP on the hardware.

Figure 115. Main View of the System Console



If you are using a toolkit, you can add or remove columns from the table in the **Main View**. Right-click on the table header and select **Edit Columns** in the right-click menu. The **Select column headers** dialog box is displayed where you can choose to include more columns, as shown in the following image:

Figure 116. Column Selection in the Main View Table



Parameters Pane

The **Main View** provides the **Parameters** pane that has two tabs, one for global parameters (those not associated with a given channel) and another for channel parameters (those associated with channels). The **Channel Parameters** tab is filled with per-channel parameter editors based on channel row selection in the **Status Table**, as System Console Toolkit Explorer shows.

Status Table Pane

The **Status Table** does not appear for toolkits that do not define channels. The **Status Table** allows you to view status information across all channels of a collection or a toolkit instance, as well as execute actions across multiple channels, as System Console Toolkit Explorer shows. You can execute bulk actions spanning multiple channels by selecting desired channels, and right-clicking and exploring the **Actions** sub-menu.

You can also use the **Status Table** to select which channel to display in the Parameters Pane. The channels you select in the Status Table are shown in the **Parameters Pane**. You can use the **Pin** setting for a channel to display the channel, regardless of the current selection in the **Status Table**.

If you develop your own toolkit, you can design the layout and GUI elements in the **Main View** using the Toolkit Tcl API. You can also define how each GUI element interacts with the hardware.

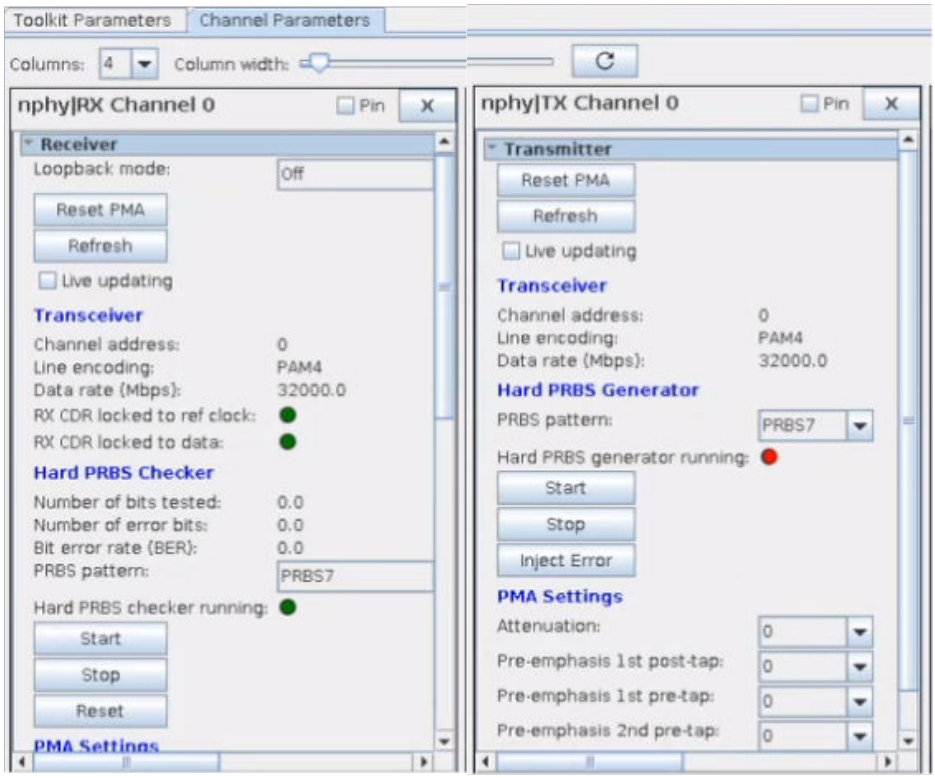
7.3.1.1.1. Link Pair View

Displaying Links With the Main View

You can use the **Main View** to simultaneously display and control associated TX and RX pairs:

1. Select both RX and TX channels in the **Status Table**.
2. Right-click to view the context-sensitive menu.
3. Navigate to the **Actions** menu.

Figure 117. Displaying Links in the Main View



Custom Groups with Links

In the **Status Table**, links are displayed like any other channel, with the exception that their parameter lists encompass all parameters from the associated TX and RX channels. If you create a group with a link and its associated TX and RX instance channels, the link row in the **Status Table** populates in all columns. Whereas, for the independent TX and RX channel rows, only parameters associated with that channel populate the **Status Table**.

Configuring a Link

You have the option to configure links in the following ways:

- Through the provided status table in the **Main View**.
- Through configuration options provided for the associated TX and RX channels.

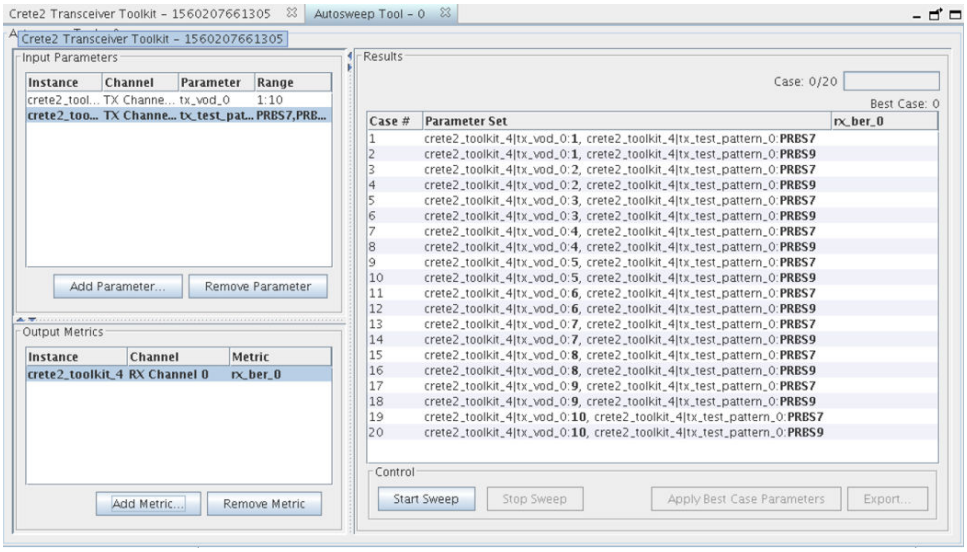
Option provided for the TX and RX channels allow you to individually manipulate each associated channel. You can manipulate multiple parameters simultaneously by selecting one or more items, and then right-clicking parameters in the status table.

7.3.1.2. Autosweep View

The **Autosweep** view allows you to sweep over IP parameters to find the best combination and define your own quality metrics for a given Autosweep run.

The System Console **Autosweep** view allows you to define your own quality metrics for a given Autosweep run.

Figure 118. Autosweep View of the System Console



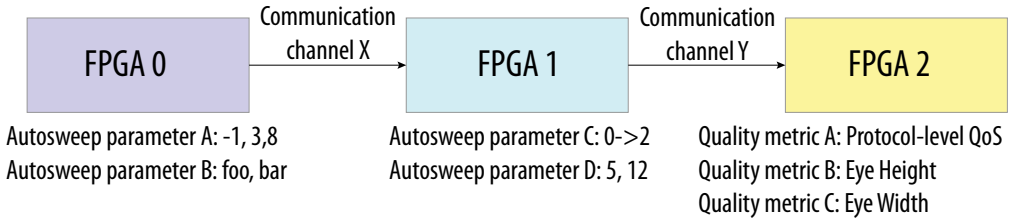
By default, the **Autosweep** view launches without any connection to a toolkit instance(s) or channel pair(s). You can add parameters by clicking **Add Parameter** and selecting parameters from specific toolkit instance(s) in the **Select Parameter** dialog box. You can remove parameters by selecting them and clicking **Remove Parameter**. Alternatively, you can add your own parameters and create as many **Autosweep** views as you want, to allow sweeping over different parameters on different channels of the same instance, or different instances entirely.

To save a parameter set for future use, select the parameter set, and then click **Export Settings**. To load a collection, click **Import Settings**.

Important: Any channel of a particular instance that has parameters currently being swept over in one **Autosweep** view cannot have other (or the same) parameters swept over in a different **Autosweep** view. For example, if one **Autosweep** view is currently sweeping over parameters from `InstA | Channel 0`, and another **Autosweep** view has parameters from `InstA | Channel 0`, an error is displayed if you attempt to start the second sweep before the first has completed. This prevents you from changing more things than are expected from a given run of autosweep.

Consider the following example complex system with parameters spread across multiple devices:

Figure 119. An Example of the Autosweep System



The **Autosweep** view allows you to sweep such a complex system when the multiple devices are visible to System Console. You can select the quality metrics from instances different from those you sweep. You can even span levels of the hardware stack from the PMA-level up to protocol-level signaling.

Results

The **Results** table is populated with one row per autosweep iteration. For every output quality metric added in the **Output Metrics** section, a column for that metric is added to the **Results** table, with new row entries added to the bottom. This format allows sorting of the results by quality metric of the system under test, across many combinations of parameters, to determine which parameter settings achieve best real-world results.

The **Results** table allows visualizing or copying the parameter settings associated with a given case, and sorting by quality metrics. Sort the rows of the table by clicking on the column headers.

Control

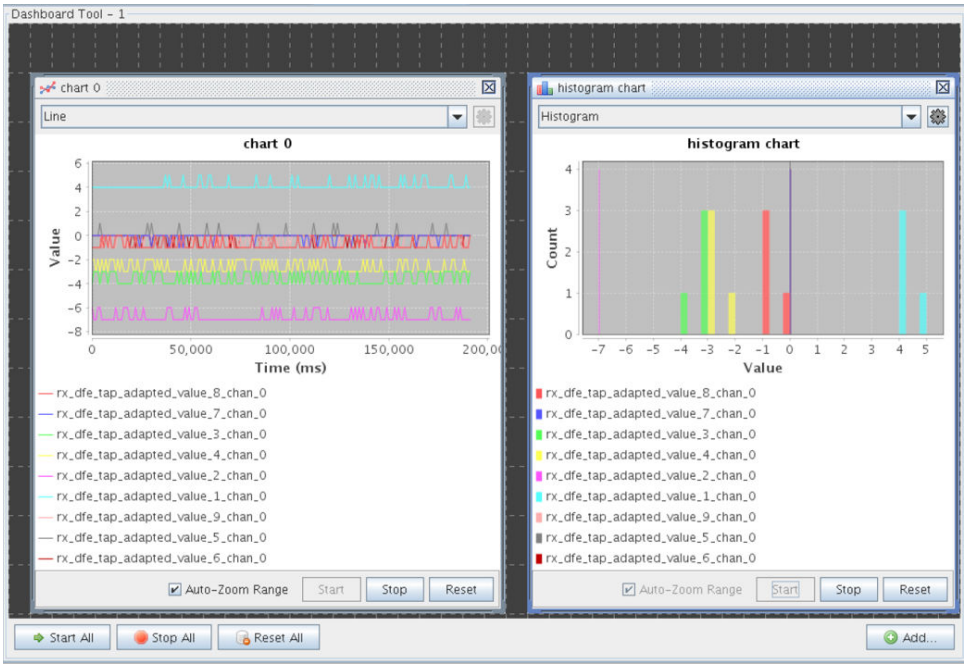
The **Control** pane of the **Autosweep** view allows starting an autosweep run, once you define at least one input parameter and one quality metric. Starting a run, allowing all combinations to complete, and then pressing the **Start** button re-runs the same test case. Pressing the **Stop** button cancels a currently running autosweep.

7.3.1.3. Dashboard View

The **Dashboard** view allows you to visualize the changes to toolkit parameters over time.

The **Dashboard** provides options to view a line chart, histogram, pie chart or bar graph, and a data history. There is no limit imposed on the number of instances of the **Dashboard** view open at once. However, a performance penalty occurs if you update a high number of parameters at a high frequency simultaneously.

Figure 120. Dashboard View of the System Console



The **Add Parameter** dialog box opens when you click the **Add** button. Only those parameters that declare the `allows_charting` parameter property are available for selection in the **Add Parameter** dialog box.

7.3.1.4. Eye Viewer

The System Console **Eye Viewer** allows you to configure, run, and render eye scans. The **Eye Viewer** allows independent control of the eye for each transceiver instance. System Console allows you to open only one **Eye Viewer** per-instance channel pair at any given time. Therefore, there is a one-to-one mapping of a given **Eye Viewer** GUI to a given instance of the eye capture hardware on the FPGA. Click **Tools ➤ Eye Viewer** to launch the **Eye Viewer**.

Eye Viewer Controls

The **Eye Viewer** controls allow you to configure toolkit-specific settings for the current **Eye Viewer** scan. The parameters in the **Eye Viewer** affect the behavior and details of the eye scan run.

Start / Stop Controls

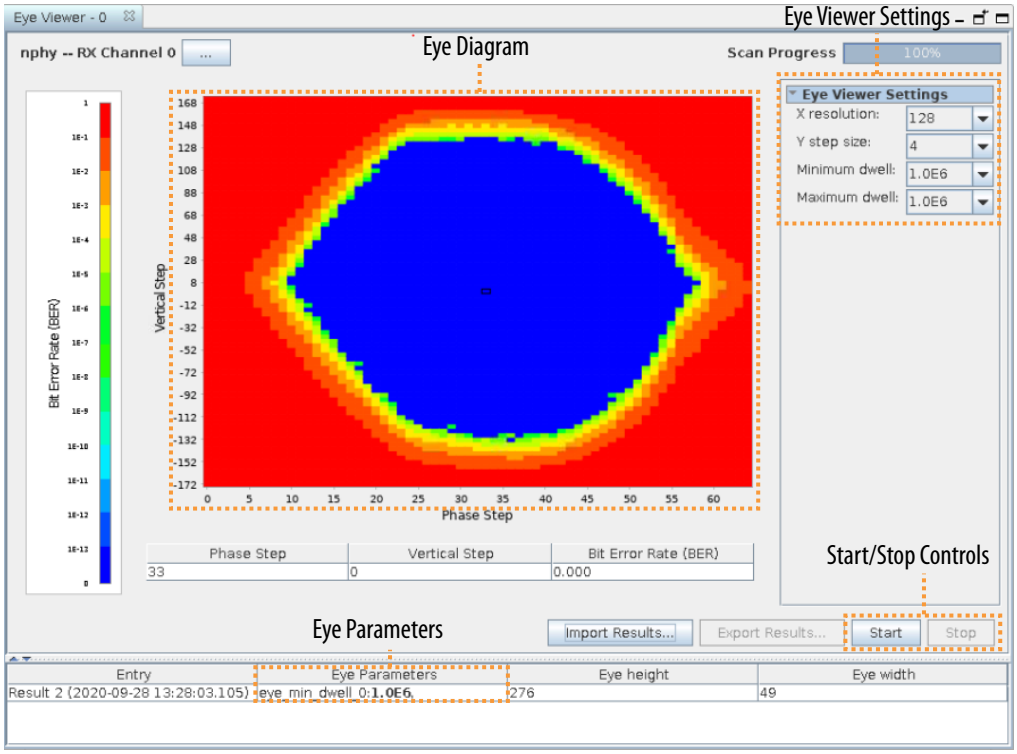
The **Eye Viewer** provides **Start** and **Stop** controls. The **Start** button starts the eye scanning process while the **Stop** button cancels an incomplete scan.

Note: The actual scanning controls, configurations, and metrics shown with the **Eye Viewer** vary by toolkit.

Eye Diagram Visualization

The eye diagram displays the transceiver eye captured from on-die instrumentation (ODI) with a color gradient.

Figure 121. Eye Viewer (E-Tile Transceiver Native PHY Toolkit Example)



Results Table

The **Results** table displays results and statistics of all eye scans. While an eye scan is running, you cannot view any partial results. However, there is a progress bar showing the current progress of the eye scan underway.

When an eye scan successfully completes, a new entry appears in the **Results** table, and that entry automatically gains focus. When you select a given entry in the **Results** table, the eye diagram renders the associated eye data. You can right-click in the **Results** table to do the following:

- Apply the test case parameters to the device
- Delete an entry

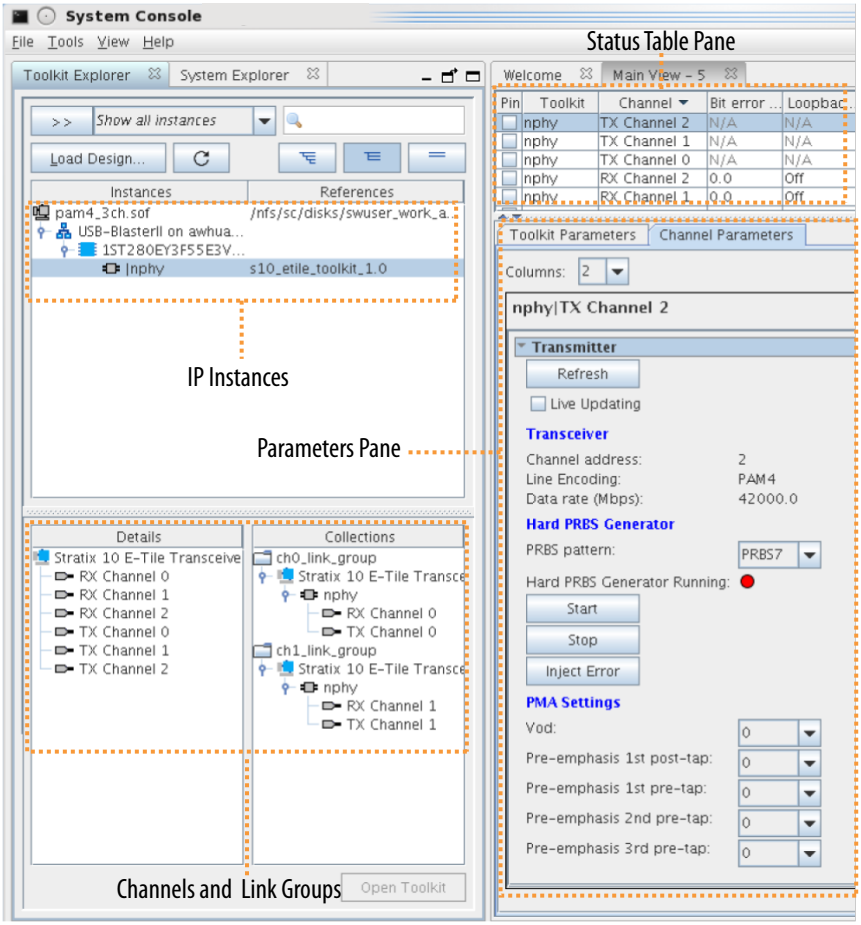
If developing your own toolkit that includes the **Eye Viewer**, the BER gradient is configurable, and the eye diagram GUI supports the following features:

- A BER tool-tip for each cell
- Ability to export the map as PNG
- Zoom

7.3.2. Toolkit Explorer Pane

The **Toolkit Explorer** pane displays all available toolkits and launches tools that use the System Console framework. When you load a design that contains debug-enabled IP, the **Toolkit Explorer** displays the design instances, along with a list of channels and channel collections for debugging. To interact with a channel or a toolkit, double-click on it to launch the **Main View** tabbed window, as shown in the following image:

Figure 122. System Console Toolkit Explorer



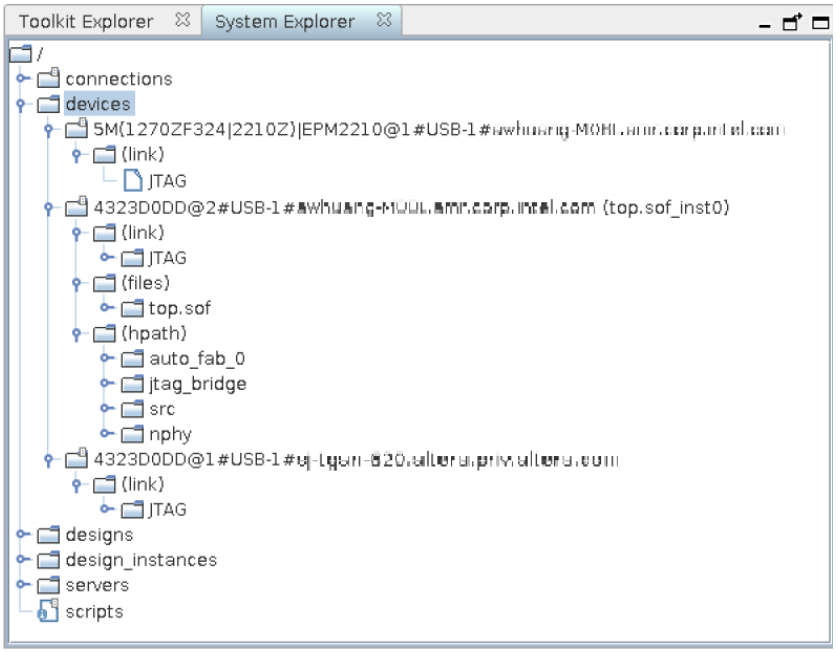
Note: If you close **Toolkit Explorer**, you can reopen it by clicking **View ➤ Toolkit Explorer**.

7.3.3. System Explorer Pane

The **System Explorer** pane displays a list of interactive instances from the design loaded on a connected device. This includes the following items:

- IP instances with debug toolkit capabilities
- IP instances with a debug endpoint

Figure 123. System Explorer Pane



Additionally, the **System Explorer** also displays custom toolkit groups and links that you create. **System Explorer** organizes the interactive instances according to the available device connections. The **System Explorer** contains a **Links** instance, and may contain a **Files** instance. The **Links** instance shows debug agents (and other hardware) that System Console can access. The **Files** instance contains information about the programming files loaded from the Intel Quartus Prime project for the device.

The **System Explorer** provides the following information:

- **Devices**—displays information about all devices connected to the System Console.
- **Scripts**—stores scripts for easy execution.
- **Connections**—displays information about the board connections visible to System Console, such as the Intel FPGA Download Cable. Multiple connections are possible.
- **Designs**—displays information about Intel Quartus Prime designs connected to System Console. Each design represents a loaded `.sof` file.
- Right-click on some of the instances to execute related commands.
- Instances that include a message display a message icon. Click on the instance to view the messages in the **Messages** pane.

7.3.4. Customizing, Saving, and Resetting the System Console Layout

The System Console GUI includes the default **System Console** layout (for general system-level debug) and **Toolkit Explorer** layout (for debugging with toolkits) layouts. These layouts are tailored for system level debug and running debug toolkits, respectively.

You can customize these layouts for your preferences in the System Console workspace, save and reload your custom layout, and reset the layout to default. When you re-load a design, System Console restores the last saved workspace associated with that design.

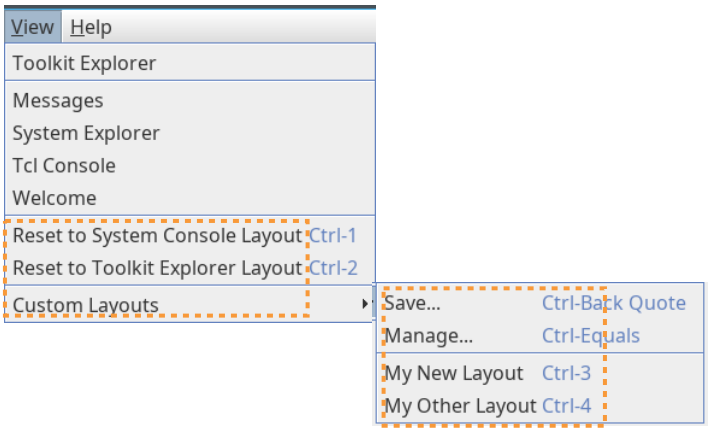
To customize the layout to suit your task flow and preferences:

Follow these steps to customize and save the Platform Designer layout:

1. Click tabbed items on the **View** or **Tools** menus to display and then optionally arrange, remove, or group the items to suit your preferences:
   - Drag, drop, and group the items in the workspace to match your task flow and preferences.
   - Close or dock items that you are not using.
   - Dock items in the main frame as a group, or individually by clicking the tab control in the upper-right corner of the main frame.
   - Tool tips on the upper-right corner of the tabbed item suggest potential workspace arrangements, for example, restoring or disconnecting a tab from the workspace.

     Note: You cannot include the **Autosweep**, **Dashboard**, **Eye Viewer**, or **Legacy Toolkits** tabs from **Tools** menu in custom layouts. You must first close any of these items before saving a layout.
2. To save the current workspace configuration as a custom layout, click **View ➤ Custom Layouts ➤ Save** and specify a layout **Name**. System Console saves the custom layouts and order in the `layouts.ini` file in project directory, and adds the layout to the **Custom Layouts** list.

Figure 124. View Menu Layouts

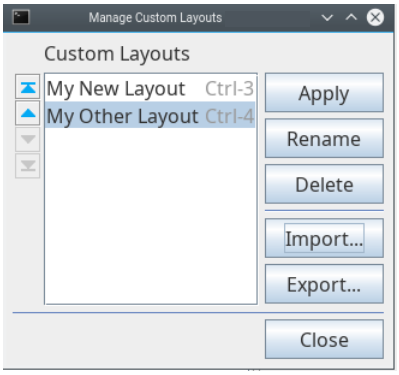


3. Use any of the following methods to change to another layout:

- To revert the layout back to a default layout, click **View ➤ Reset to System Console Layout** or **View ➤ Reset to Toolkit Explorer Layout**.
- To set your workspace to a previously saved layout, click **View ➤ Custom Layouts**, and then select the custom layout.

4. To import, export, delete or rename custom layouts, click **View ➤ Custom Layouts ➤ Manage**. The **Manage Custom Layouts** dialog box opens and allows you to apply a variety of functions that facilitate custom layout management.

Figure 125. Manage Custom Layouts



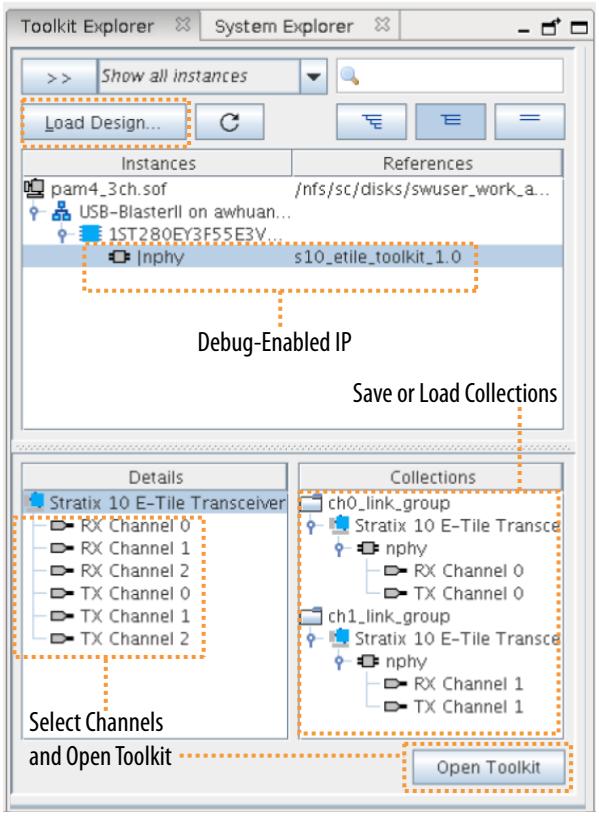
7.4. Launching a Toolkit in System Console

System Console provides the hardware debugging infrastructure to run the Available System Debugging Toolkits on page 175. When you load a design in the **Toolkit Explorer** that includes debug-enabled Intel FPGA IP, the **Toolkit Explorer** automatically lists the toolkits that are available for the IP in the design.

To launch a toolkit in System Console, follow these steps:

1. Create an Intel Quartus Prime project that includes debug-enabled Intel FPGA IP. Refer to IP Cores that Interact with System Console on page 160.
2. On the Compilation Dashboard, double-click **Assembler** to generate a `.sof` programming file for the design. Any prerequisite Compiler stages run automatically before the Assembler starts.
3. Launch System Console, as Starting System Console on page 162 describes.
4. In the **Toolkit Explorer**, click **Load Design**, and then select the `.sof` file that you create in step 2. When you load the design, **Toolkit Explorer** displays the debug-enabled IP instances.
5. Select a debug-enabled IP instance. The **Details** pane displays the channels that can launch toolkits.
6. To launch a toolkit, select the toolkit under **Details**. For toolkits with channels, you can also multi-select one or more channels from the **Details** pane. Then, click **Open Toolkit**. The toolkit opens in the **Main View**, and the **Collections** pane displays a collection of any channels that you select.
7. To save a collection for future use, right-click the collection, and then click **Export Collection.** To load a collection, right-click in the **Collections** pane, and then click **Import Collection**. By default, System Console creates a collection when you launch a toolkit.

Figure 126. Launching a Toolkit in System Console



Related Information

System Console and Toolkit Tcl Command Reference Manual

7.4.1. Available System Debugging Toolkits

The following toolkits are available to launch from the System Console **Toolkit Explorer** in the current version of the Intel Quartus Prime software.

Table 37. Toolkits Available in System Console Toolkit Explorer

Toolkit	Description	Toolkit Documentation
EMIF Calibration Debug Toolkit	Helps you to debug external memory interfaces by accessing calibration data obtained during memory calibration. The analysis tools can evaluate the stability of the calibrated interface and assess hardware conditions.	• *External Memory Interfaces Intel Agilex 7 FPGA IP User Guide* • *External Memory Interfaces Intel Stratix 10 FPGA IP User Guide*
EMIF Traffic Generator Configuration Toolkit	Helps you to debug external memory interfaces by sending sample traffic through the external memory interface to the memory device. The generated EMIF design example includes a traffic generator block with control and status registers.	

continued...

Toolkit	Description	Toolkit Documentation
EMIF Efficiency Monitor Toolkit	Helps you to debug external memory interfaces by measuring efficiency on the Avalon interface in real time. The generated EMIF design example can include the Efficiency Monitor block.	• *External Memory Interfaces Intel Agilex 7 FPGA IP User Guide* • *External Memory Interfaces Intel Stratix 10 FPGA IP User Guide*
Ethernet Toolkit	Helps you to interact with and debug an Ethernet Intel FPGA IP interface in real time. You can verify the status of the Ethernet link, assert and deassert IP resets, verifies the IP error correction capability,	*Ethernet Toolkit User Guide*
Intel Stratix 10 FPGA P-Tile Toolkit (for PCIe)	Helps you to optimize the performance of large-size data transfers with real-time control, monitoring, and debugging of the PCI Express* links at the Physical, Data Link, and Transaction layers.	*Intel FPGA P-Tile Avalon Memory Mapped IP for PCI Express* User Guide*
Serial Lite IV IP Toolkit	An inspection tool that monitors the status of a Serial Lite IV IP link and provides a step-by-step guide for the IP link initialization sequences.	• *Serial Lite IV Intel Agilex 7 FPGA IP Design Example User Guide* • *Serial Lite IV Intel Stratix 10 FPGA IP Design Example User Guide*
Intel Arria 10 and Intel Cyclone 10 GX Transceiver Native PHY Toolkit	Helps you to optimize high-speed serial links in your board design by providing real-time control, monitoring, and debugging of the transceiver links running on your board.	
L-Tile and H-Tile Transceiver Native PHY Toolkit		
E-Tile Transceiver Native PHY Toolkit		

The following legacy toolkits remain available by clicking **Tools ➤ Legacy Toolkits** in System Console. The legacy toolkits support earlier device families and may be subject to end of life and removal of support in a coming software release.

Table 38. Legacy Toolkits Available in System Console

Legacy Toolkit	Description	Legacy Toolkit Documentation
Ethernet Link Inspector - Link Monitor Toolkit	The Ethernet Link Inspector is an inspection tool that can continuously monitor an Ethernet link that contains an Ethernet IP. The Link Monitor toolkit performs real-time status monitoring of an Ethernet IP link. The link monitor continuously reads and displays all of the required status registers related to the Ethernet IP link, and displays the Ethernet IP link status at various stages are valid.	*Ethernet Link Inspector User Guide for Intel Stratix 10 Devices*
Ethernet Link Inspector - Link Analysis Toolkit	The Link Analysis toolkit displays a sequence of events on an Ethernet IP link, which occur in a finite duration of time. The Link Analysis requires the Signal Tap logic analyzer to first capture and store a database (`.csv`) of all required signals. The Link Analysis toolkit then performs an analysis on the database to extract all the required information and displays them in a user-friendly graphical user interface (GUI).	
S10 SDM Debug Toolkit	Provides access to current status of the Intel Stratix 10 device. To use these commands, you must have a valid design loaded that includes the module that you intend to access.	*Intel Stratix 10 Configuration User Guide*

Note: Refer to the toolkit documentation for individual toolkit launch, setup, and use information. The Transceiver Toolkit previously available in the Intel Quartus Prime software Tools menu is replaced by the Intel Arria 10 and Intel Cyclone 10 GX Transceiver Native PHY Toolkit.

Related Information

- External Memory Interfaces Intel Agilex 7 FPGA IP User Guide
- External Memory Interfaces Intel Stratix 10 FPGA IP User Guide
- Ethernet Toolkit User Guide
- Intel FPGA P-Tile Avalon Memory Mapped IP for PCI Express* User Guide
- Intel FPGA P-Tile Avalon Streaming IP for PCI Express* User Guide
- Ethernet Link Inspector User Guide for Intel Stratix 10 Devices
- Intel Stratix 10 Configuration User Guide
- Serial Lite IV Intel Agilex 7 FPGA IP Design Example User Guide
- Serial Lite IV Intel Stratix 10 FPGA IP Design Example User Guide

7.4.2. Creating Collections from the Toolkit Explorer

You can create custom collections to view and configure members from different instances in a single Main View.

Perform these steps to group instances:

1. Select multiple items in the instances tree.
2. Right click to view the context-sensitive menu.
3. Select **Add to Collection ➤ New Collection**. The **Add to Collection** dialog box appears with members you select.

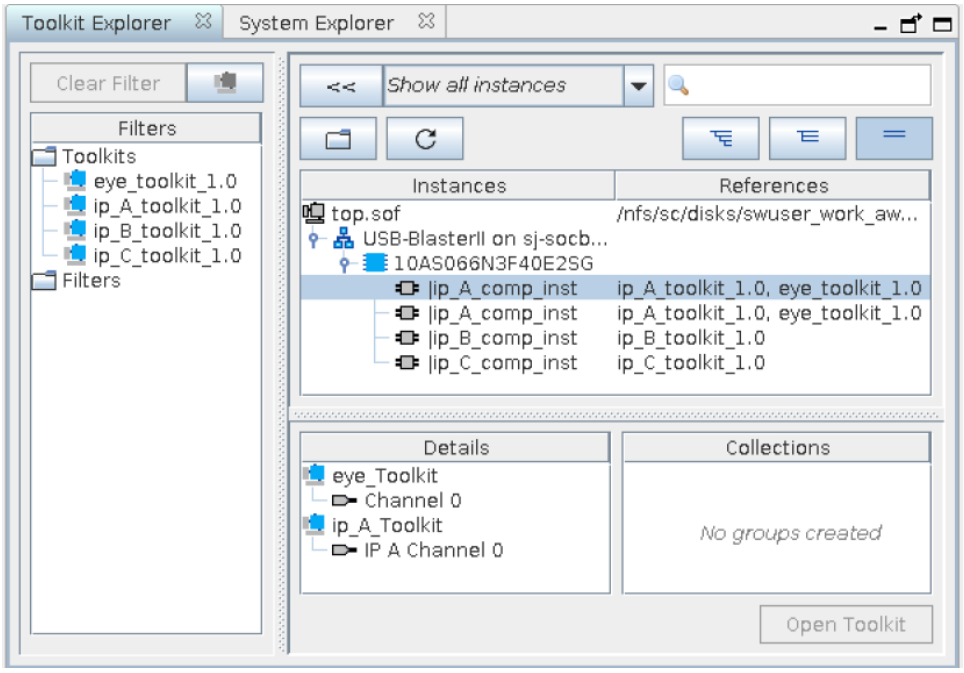
System Console adds the collections that you create to the **Collections** pane of the **Toolkit Explorer**. You can perform one of the following actions:

- Double-click on a custom-created collection to launch the Main view containing all of the group's members.
- Right-click on an existing collection member and select **Remove from Collection** to remove the member.

7.4.3. Filtering and Searching Interactive Instances

By default, the **Toolkits** list shows all toolkit instances and their respective channels linking to the System Console. This view is useful in simple cases, but can become very dense in a complex system having many debug-enabled IPs, and having potentially multiple FPGAs connected to System Console.

Figure 127. Toolkit Explorer with Filters



To limit the information display, the **Filter** list allows filtering the display by toolkit types currently available in the System Console. You can also create custom filters using groups, for example, "Inst A, Inst F, and Inst Z", or "E-Tile and L/H-Tile Transceivers only".

To refine the list of toolkits, use the search field in the **Toolkit Explorer** to filter the list further.

7.5. Using System Console Services

System Console services provide access to hardware modules that you instantiate in your FPGA. Services vary in the type of debug access they provide.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.5.1. Locating Available Services

System Console uses a virtual file system to organize the available services, which is similar to the `/dev location` on Linux systems. Board connection, device type, and IP names are all part of a service path. Instances of services are referred to by their unique service path in the file system. To retrieve service paths for a particular service, use the command `get_service_paths` <*service-type*>.

Example 11. Locating a Service Path

```
#We are interested in master services.
set service_type "master"

#Get all the paths as a list.
```

```
set master_service_paths [get_service_paths $service_type]

#We are interested in the first service in the list.
set master_index 0

#The path of the first master.
set master_path [lindex $master_service_paths $master_index]

#Or condense the above statements into one statement:
set master_path [lindex [get_service_paths master] 0]
```

System Console commands require service paths to identify the service instance you want to access. The paths and indexes for different components can change between runs of System Console and between versions. Use the `get_service_paths` command to obtain service paths.

The string values of service paths change with different releases of the tool. Use the `marker_node_info` command to get information from the path.

System Console automatically discovers most services at startup. System Console automatically scans for all JTAG and USB-based service instances and retrieves their service paths. System Console does not automatically discover some services, such as TCP/IP. Use `add_service` to inform System Console about those services.

Example 12. Marker_node_info

Use the `marker_node_info` command to get information about SLD nodes associated with the specified service.

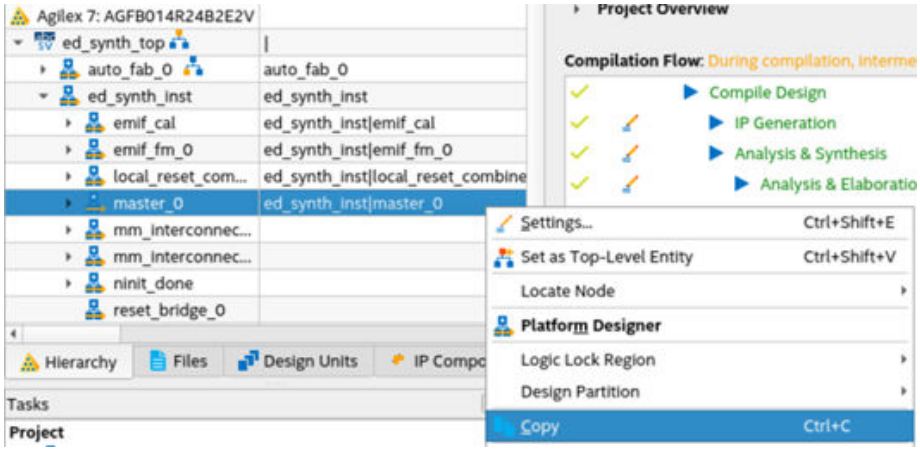
```
set slave_path [get_service_paths -type altera_avalon_uart.slave slave]
array set uart_info [marker_node_info $slave_path]
echo $uart_info(full_hpath)
```

Example 13. Locating the Correct Service Path

If the design contains multiple instances of the same service, it is necessary to use the full hierarchy path to correctly identify the service path.

In the following example, to identify the correct service path for a debug endpoint called `master_0`:

1. Locate the full hierarchy name of the endpoint in your design.



2. Right-click `master_0` and select **Copy** in the context-sensitive menu. This copies the full hierarchical path, (`ed_synth_top|master_0`).
3. Save this hierarchy path to a Tcl variable for easy reference later, as follows:

```
set desired_hpath ed_synth_inst|master_0
```

4. Use the `get_service_paths` command with the `–hpath` flag and the hierarchy path to obtain specific service path of the desired endpoint. The command returns a list with a service path.

```
#Return a list of one element with the desired element
set service_paths [get_service_paths -hpath $desired_hpath master]
```

5. Save the service path as a variable for easier access later.

```
#Extract and store the element in a variable
set master_0_spath [lindex $service_paths 0]
```

7.5.2. Opening and Closing Services

After you have a service path to a particular service instance, you can access the service for use.

The `claim_service` command directs System Console to start using a particular service instance, and with no additional arguments, claims a service instance for exclusive use.

Example 14. Opening a Service

```
set service_type "master"
set claim_path [claim_service $service_type $master_path mylib];#Claims service.
```

You can pass additional arguments to the `claim_service` command to direct System Console to start accessing a particular portion of a service instance. For example, if you use the master service to access memory, then use `claim_service` to only access the address space between `0x0` and `0x1000`. System Console then allows other users to access other memory ranges, and denies access to the claimed memory range. The `claim_service` command returns a newly created service path that you can use to access your claimed resources.

You can access a service after you open it. When you finish accessing a service instance, use the `close_service` command to direct System Console to make this resource available to other users.

Example 15. Closing a Service

```
close_service master $claim_path; #Closes the service.
```

7.5.3. Using the SLD Service

The SLD Service shifts values into the instruction and data registers of SLD nodes and captures the previous value. When interacting with a SLD node, start by acquiring exclusive access to the node on an opened service.

Example 16. SLD Service

```
set timeout_in_ms 1000
set lock_failed [sld_lock $sld_service_path $timeout_in_ms]
```

This code attempts to lock the selected SLD node. If it is already locked, `sld_lock` waits for the specified timeout. Confirm the procedure returns non-zero before proceeding. Set the instruction register and capture the previous one:

```
if {$lock_failed} {
    return
}
set instr 7
set delay_us 1000
set capture [sld_access_ir $sld_service_path $instr $delay_us]
```

The 1000 microsecond delay guarantees that the following SLD command executes at least 1000 microseconds later. Data register access works the same way.

```
set data_bit_length 32
set delay_us 1000
set data_bytes [list 0xEF 0xBE 0xAD 0xDE]
set capture [sld_access_dr $sld_service_path $data_bit_length $delay_us \
$data_bytes]
```

Shift count is specified in bits, but the data content is specified as a list of bytes. The capture return value is also a list of bytes. Always unlock the SLD node once finished with the SLD service.

```
sld_unlock $sld_service_path
```

Related Information

Virtual JTAG IP Core User Guide

7.5.3.1. SLD Commands

Table 39. SLD Commands

Command	Arguments	Function
`sld_access_ir`	*<claim-path>* *<ir-value>* *<delay>* (in µs)	Shifts the instruction value into the instruction register of the specified node. Returns the previous value of the instruction.

continued...

Command	Arguments	Function
		If the <*delay*> parameter is non-zero, then the JTAG clock is paused for this length of time after the access.
`sld_access_dr`	<*service-path*> <*size_in_bits*> <*delay-in-µs*>, <*list_of_byte_values*>	Shifts the byte values into the data register of the SLD node up to the size in bits specified. If the <*delay*> parameter is non-zero, then the JTAG clock is paused for at least this length of time after the access. Returns the previous contents of the data register.
`sld_lock`	<*service-path*> <*timeout-in-milliseconds*>	Locks the SLD chain to guarantee exclusive access. Returns 0 if successful. If the SLD chain is already locked by another user, tries for <*timeout*>ms before returning a Tcl error. You can use the catch command if you want to handle the error.
`sld_unlock`	<*service-path*>	Unlocks the SLD chain.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.5.4. Using the In-System Sources and Probes Service

The In-System Sources and Probes (ISSP) service provides scriptable access to the In-System Sources and Probes Intel FPGA IP in a similar manner to using the **In-System Sources and Probes Editor** in the Intel Quartus Prime software.

Example 17. ISSP Service

Before you use the ISSP service, ensure your design works in the **In-System Sources and Probes Editor**. In System Console, open the service for an ISSP instance:

```
set issp_index 0
set issp [lindex [get_service_paths issp] 0]
set claimed_issp [claim_service issp $issp mylib]
```

View information about this particular ISSP instance:

```
array set instance_info [issp_get_instance_info $claimed_issp]
set source_width $instance_info(source_width)
set probe_width $instance_info(probe_width)
```

The Intel Quartus Prime software reads probe data as a single bitstring of length equal to the probe width:

```
set all_probe_data [issp_read_probe_data $claimed_issp]
```

As an example, you can define the following procedure to extract an individual probe line's data:

```
proc get_probe_line_data {all_probe_data index} {
    set line_data [expr { ($all_probe_data >> $index) & 1 }]
    return $line_data
}
set initial_all_probe_data [issp_read_probe_data $claim_issp]
set initial_line_0 [get_probe_line_data $initial_all_probe_data 0]
set initial_line_5 [get_probe_line_data $initial_all_probe_data 5]
# ...
set final_all_probe_data [issp_read_probe_data $claimed_issp]
set final_line_0 [get_probe_line_data $final_all_probe_data 0]
```

Similarly, the Intel Quartus Prime software writes source data as a single bitstring of length equal to the source width:

```
set source_data 0xDEADBEEF
issp_write_source_data $claimed_issp $source_data
```

You can also retrieve the currently set source data:

```
set current_source_data [issp_read_source_data $claimed_issp]
```

As an example, you can invert the data for a 32-bit wide source by doing the following:

```
set current_source_data [issp_read_source_data $claimed_issp]
set inverted_source_data [expr { $current_source_data ^ 0xFFFFFFFF }]
issp_write_source_data $claimed_issp $inverted_source_data
```

7.5.4.1. In-System Sources and Probes Commands

Note: The valid values for ISSP claims include `read_only`, `normal`, and `exclusive`.

Table 40. In-System Sources and Probes Commands

Command	Arguments	Function
`issp_get_instance_info`	*<service-path>*	Returns a list of the configurations of the In-System Sources and Probes instance, including: `instance_index` `instance_name` `source_width` `probe_width`
`issp_read_probe_data`	*<service-path>*	Retrieves the current value of the probe input. A hex string is returned representing the probe port value.
`issp_read_source_data`	*<service-path>*	Retrieves the current value of the source output port. A hex string is returned representing the source port value.
`issp_write_source_data`	*<service-path>* *<source-value>*	Sets values for the source output port. The value can be either a hex string or a decimal value supported by the System Console Tcl interpreter.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.5.5. Using the Monitor Service

The monitor service builds on top of the host service to allow reads of Avalon memory-mapped interface agents at a regular interval. The service is fully software-based. The monitor service requires no extra soft-logic. This service streamlines the logic to do interval reads, and it offers better performance than exercising the host service manually for the reads.

Example 18. Monitor Service

1. Determine the host and the memory address range that you want to poll:

```
set master_index     0
set master [lindex [get_service_paths master] $master_index]
set address          0x2000
set bytes_to_read    100
set read_interval_ms 100
```

   With the first host, read 100 bytes starting at address 0x2000 every 100 milliseconds.

2. Open the monitor service:

```
set monitor [lindex [get_service_paths monitor] 0]
set claimed_monitor [claim_service monitor $monitor mylib]
```

   The monitor service opens the host service automatically.

3. With the monitor service, register the address range and time interval:

```
monitor_add_range $claimed_monitor $master $address $bytes_to_read
monitor_set_interval $claimed_monitor $read_interval_ms
```

4. Add more ranges, defining the result at each interval:

```
global monitor_data_buffer
set monitor_data_buffer [list]
```

5. Gather the data and append it with a global variable:

```
proc store_data {monitor master address bytes_to_read} {\
  global monitor_data_buffer
# monitor_read_data returns the range of data polled from the running \
  design as a list
#(in this example, a 100-element list).
  set data [monitor_read_data $claimed_monitor $master $address \
  $bytes_to_read]
# Append the list as a single element in the monitor_data_buffer \
  global list.
  lappend monitor_data_buffer $data
}
```

   Note: If this procedure takes longer than the interval period, the monitor service may have to skip the next one or more calls to the procedure. In this case, `monitor_read_data` returns the latest polled data.

6. Register this callback with the opened monitor service:

```
set callback [list store_data $claimed_monitor $master $address
$bytes_to_read]
monitor_set_callback $claimed_monitor $callback
```

7. Use the callback variable to call when the monitor finishes an interval. Start monitoring:

```
monitor_set_enabled $claimed_monitor 1
```

   Immediately, the monitor reads the specified ranges from the device and invokes the callback at the specified interval. Check the contents of `monitor_data_buffer` to verify this. To turn off the monitor, use 0 instead of 1 in the above command.

7.5.5.1. Monitor Commands

You can use the Monitor commands to read many Avalon memory-mapped interface agent memory locations at a regular interval.

Under normal load, the monitor service reads the data after each interval and then calls the callback. If the value you read is timing sensitive, you can use the `monitor_get_read_interval` command to read the exact time between the intervals at which the data was read.

Under heavy load, or with a callback that takes a long time to execute, the monitor service skips some callbacks. If the registers you read do not have side effects (for example, they read the total number of events since reset), skipping callbacks has no effect on your code. The `monitor_read_data` command and `monitor_get_read_interval` command are adequate for this scenario.

If the registers you read have side effects (for example, they return the number of events since the last read), you must have access to the data that was read, but for which the callback was skipped. The `monitor_read_all_data` and `monitor_get_all_read_intervals` commands provide access to this data.

Table 41. Monitoring Commands

Command	Arguments	Function
`monitor_add_range`	*<service-path>* *<target-path>* *<address>* *<size>*	Adds a contiguous memory address into the monitored memory list. *<service path>* is the value returned when you opened the service. *<target-path>* argument is the name of a host service to read. The address is within the address space of this service. *<target-path>* is returned from `[lindex [get_service_paths master] n]` where *n* is the number of the host service. *<address>* and *<size>* are relative to the host service.
`monitor_get_all_read_intervals`	*<service-path>* *<target-path>* *<address>* *<size>*	Returns a list of intervals in milliseconds between two reads within the data returned by `monitor_read_all_data`.
`monitor_get_interval`	*<service-path>*	Returns the current interval set which specifies the frequency of the polling action.
`monitor_get_missing_event_count`	*<service-path>*	Returns the number of callback events missed during the evaluation of last Tcl callback expression.
`monitor_get_read_interval`	*<service-path>* *<target-path>* *<address>* *<size>*	Returns the milliseconds elapsed between last two data reads returned by `monitor_read_data`.
`monitor_read_all_data`	*<service-path>* *<target-path>* *<address>* *<size>*	Returns a list of 8-bit values read from all recent values read from device since last Tcl callback. You must specify a memory range within the range in `monitor_add_range`.

continued...

Command	Arguments	Function
`monitor_read_data`	*<service-path>* *<target-path>* *<address>* *<size>*	Returns a list of 8-bit values read from the most recent values read from device. You must specify a memory range within the range in `monitor_add_range`.
`monitor_set_callback`	*<service-path>* *<Tcl-expression>*	Specifies a Tcl expression that the System Console must evaluate after reading all the memories that this service monitors. Typically, you specify this expression as a single string Tcl procedure call with necessary argument passed in.
`monitor_set_enabled`	*<service-path>* *<enable(1)/disable(0)>*	Enables and disables monitoring. Memory read starts after this command, and Tcl callback evaluates after data is read.
`monitor_set_interval`	*<service-path>* *<interval>*	Defines the target frequency of the polling action by specifying the interval between two memory reads. The actual polling frequency varies depending on the system activity.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.5.6. Using the Device Service

The device service supports device-level actions.

Example 19. Programming

You can use the device service with Tcl scripting to perform device programming:

```
set device_index 0 ; #Device index for target
set device [lindex [get_service_paths device] $device_index]
set sof_path [file join project_path output_files project_name.sof]
device_download_sof $device $sof_path
```

To program, all you need are the device service path and the file system path to a `.sof`. Ensure that no other service (e.g. host service) is open on the target device or else the command fails. Afterwards, you may do the following to check that the design linked to the device is the same one programmed:

```
device_get_design $device
```

7.5.6.1. Device Commands

The device commands provide access to programmable logic devices on your board. Before you use these commands, identify the path to the programmable logic device on your board using the `get_service_paths`.

Table 42. Device Commands

Command	Arguments	Function
`device_download_sof`	*<service_path>* *<sof-file-path>*	Loads the specified `.sof` to the device specified by the path.
`device_get_connections`	*<service_path>*	Returns all connections which go to the device at the specified path.
`device_get_design`	*<device_path>*	Returns the design this device is currently linked to.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.5.7. Using the Design Service

You can use design service commands to work with Intel Quartus Prime design information.

Example 20. Load

When you open System Console from the Intel Quartus Prime software, the current project's debug information is sourced automatically if the `.sof` file is present. In other situations, you can load the `.sof` manually.

```
set sof_path [file join project_dir output_files project_name.sof]
set design [design_load $sof_path]
```

System Console is now aware of the `.sof` loading.

Example 21. Linking

Once a `.sof` loads, System Console automatically links design information to the connected device. The link persists and you can unlink or reuse the link on an equivalent device with the same `.sof`.

You can perform manual linking as follows:

```
set device_index 0; # Device index for our target
set device [lindex [get_service_paths device] $device_index]
design_link $design $device
```

Manual linking fails if the target device does not match the design service.

Linking fails even if the `.sof` programmed to the target is not the same as the design `.sof`.

7.5.7.1. Design Service Commands

Design service commands load and work with your design at a system level.

Table 43. Design Service Commands

Command	Arguments	Function
`design_load`	*<quartus-project-path>*, *<sof-file-path>*, or *<qpf-file-path>*	Loads a model of an Intel Quartus Prime design into System Console. Returns the design path. For example, if your Intel Quartus Prime Project File (`.qpf`) is in `c:/projects/loopback`, type the following command: `design_load {c:\projects\loopback\}`
`design_link`	*<design-path>* *<device-service-path>*	Links an Intel Quartus Prime logical design with a physical device. For example, you can link an Intel Quartus Prime design called **2c35_quartus_design** to a 2c35 device. After you create this link, System Console creates the appropriate correspondences between the logical and physical submodules of the Intel Quartus Prime project.
`design_extract_debug_files`	*<design-path>* *<zip-file-name>*	Extracts debug files from a `.sof` to a zip file which can be emailed to *Intel FPGA Support* for analysis.

continued...

Command	Arguments	Function
		You can specify a design path of {} to unlink a device and to disable auto linking for that device.
design_get_warnings	*<design-path>*	Gets the list of warnings for this design. If the design loads correctly, then an empty list returns.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.5.8. Using the Bytestream Service

The bytestream service provides access to modules that produce or consume a stream of bytes. Use the bytestream service to communicate directly to the IP core that provides bytestream interfaces, such as the JTAG UART or the Avalon Streaming JTAG interface Intel FPGA IP.

Example 22. Bytestream Service

The following code finds the bytestream service for your interface and opens it:

```
set bytestream_index 0
set bytestream [lindex [get_service_paths bytestream] $bytestream_index]
set claimed_bytestream [claim_service bytestream $bytestream mylib]
```

To specify the outgoing data as a list of bytes and send it through the opened service:

```
set payload [list 1 2 3 4 5 6 7 8]
bytestream_send $claimed_bytestream $payload
```

Incoming data also comes as a list of bytes:

```
set incoming_data [list]
while {[llength $incoming_data] ==0} {
    set incoming_data [bytestream_receive $claimed_bytestream 8]
}
```

Close the service when done:

```
close_service bytestream $claimed_bytestream
```

7.5.8.1. Bytestream Commands

Table 44. Bytestream Commands

Command	Arguments	Function
bytestream_send	*<service-path>* *<values>*	Sends the list of bytes to the specified bytestream service. Values argument is the list of bytes to send.
bytestream_receive	*<service-path>* *<length>*	Returns a list of bytes currently available in the specified services receive queue, up to the specified limit. Length argument is the maximum number of bytes to receive.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.5.9. Using the JTAG Debug Service

The JTAG Debug service allows you to check the state of clocks and resets within your design.

The following is a JTAG Debug design flow example.

1. To identify available JTAG Debug paths:

```
get_service_paths jtag_debug
```

2. To select a JTAG Debug path:

```
set jtag_debug_path [lindex [get_service_paths jtag_debug] 0]
```

3. To claim a JTAG Debug service path:

```
 set claim_jtag_path [claim_service jtag_debug$jtag_debug_path mylib]
```

4. Running the JTAG Debug service:

```
jtag_debug_reset_system $claim_jtag_path
jtag_debug_loop $claim_jtag_path [list 1 2 3 4 5]
```

7.5.9.1. JTAG Debug Commands

JTAG Debug commands help debug the JTAG Chain connected to a device.

Table 45. JTAG Debug Commands

Command	Argument	Function
`jtag_debug_loop`	*<service-path> <list_of_byte_values>*	Loops the specified list of bytes through a loopback of `tdi` and `tdo` of a system-level debug (SLD) node. Returns the list of byte values in the order that they were received. This command blocks until all bytes are received. Byte values have the `0x` (hexadecimal) prefix and are delineated by spaces.
`jtag_debug_sample_clock`	*<service-path>*	Returns the clock signal of the system clock that drives the module's system interface. The clock value is sampled asynchronously; consequently, you must sample the clock several times to guarantee that it is switching.
`jtag_debug_sample_reset`	*<service-path>*	Returns the value of the `reset_n` signal of the Avalon-ST JTAG Interface core. If `reset_n` is low (asserted), the value is 0 and if `reset_n` is high (deasserted), the value is 1.
`jtag_debug_sense_clock`	*<service-path>*	Returns a sticky bit that monitors system clock activity. If the clock switched at least once since the last execution of this command, returns 1. Otherwise, returns 0.. The sticky bit is reset to 0 on read.
`jtag_debug_reset_system`	*<service-path>*	Issues a reset request to the specified service. Connectivity within your device determines which part of the system is reset.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.6. On-Board Intel FPGA Download Cable II Support

System Console supports an On-Board Intel FPGA Download Cable II circuit via the USB Debug Master IP component. This IP core supports the master service.

7.7. MATLAB and Simulink* in a System Verification Flow

You can test system development in System Console using MATLAB and Simulink*, and set up a system verification flow using the Intel FPGA Hardware in the Loop (HIL) tools. In this approach, you deploy the design hardware to run in real time, and simulate the system's surrounding components in a software environment. The HIL approach allows you to use the flexibility of software tools with the real-world accuracy and speed of hardware. You can gradually introduce more hardware components to the system verification testbench. This technique gives you more control over the integration process as you tune and validate the system. When the full system is integrated, the HIL approach allows you to provide stimuli via software to test the system under a variety of scenarios.

Advantages of HIL Approach

- Avoid long computational delays for algorithms with high processing rates
- API helps to control, debug, visualize, and verify FPGA designs all within the MATLAB environment
- FPGA results are read back by the MATLAB software for further analysis and display

Required Tools and Components

- MATLAB software
- DSP Builder for Intel FPGAs software
- Intel Quartus Prime software
- Intel FPGA

Note: The DSP Builder for Intel FPGAs installation bundle includes the System Console MATLAB API.

Figure 128. Hardware in the Loop Host-Target Setup

Related Information

Hardware in the Loop from the MATLAB Simulink Environment white paper

7.7.1. Supported MATLAB API Commands

You can perform the work from the MATLAB environment, and read and write to hosts and agents through the System Console. The supported MATLAB API commands do not require launching the System Console GUI. The supported commands are:

- `SystemConsole.refreshMasters;`
- `M = SystemConsole.openMaster(1);`
- `M.write (type, byte address, data);`
- `M.read (type, byte address, number of words);`
- `M.close`

Example 23. MATLAB API Script Example

```
SystemConsole.refreshMasters; %Investigate available targets
M = SystemConsole.openMaster(1); %Creates connection with FPGA target
%%%%%%%% User Application %%%%%%%%%%%%
....
M.write('uint32',write_address,data); %Send data to FPGA target
....
data = M.read('uint32',read_address,size); %Read data from FPGA target
....
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M.close; %Terminates connection to FPGA target
```

7.7.2. High Level Flow

1. Install the DSP Builder for Intel FPGAs software, so you have the necessary libraries to enable this flow
2. Build the design using Simulink and the DSP Builder for Intel FPGAs libraries.

   DSP Builder for Intel FPGAs helps to convert the Simulink design to HDL
3. Include Avalon memory mapped components in the design (DSP Builder for Intel FPGAs can port non-Avalon memory mapped components)
4. Include Signals and Control blocks in the design
5. Separate synthesizable and non-synthesizable logic with boundary blocks.
6. Integrate the DSP system in Platform Designer
7. Program the Intel FPGA
8. Interact with the Intel FPGA through the supported MATLAB API commands.

7.8. System Console Examples and Tutorials

Intel provides examples for performing board bring-up, creating a simple toolkit, and programming a Nios II processor. The `System_Console.zip` file contains design files for the board bring-up example. The Nios II Ethernet Standard `.zip` files contain the design files for the Nios II processor example.

Note: The instructions for these examples assume that you are familiar with the Intel Quartus Prime software, Tcl commands, and Platform Designer.

Related Information

On-Chip Debugging Design Examples Website
Contains the design files for the example designs that you can download.

7.8.1. Nios II Processor Example

This example programs the Nios II processor on your board to run the count binary software example included in the Nios II installation. This is a simple program that uses an 8-bit variable to repeatedly count from 0x00 to 0xFF. The output of this variable is displayed on the LEDs on your board. After programming the Nios II processor, you use System Console processor commands to start and stop the processor.

To run this example, perform the following steps:

1. Download the Nios II Ethernet Standard Design Example for your board from the Intel website.
2. Create a folder to extract the design. For this example, use `C:\Count_binary`.
3. Unzip the Nios II Ethernet Standard Design Example into `C:\Count_binary`.
4. In a Nios II command shell, change to the directory of your new project.
5. Program your board. In a Nios II command shell, type the following:

```
nios2-configure-sof niosii_ethernet_standard_<board_version>.sof
```

6. Using Nios II Software Build Tools for Eclipse, create a new Nios II Application and BSP from Template using the **Count Binary** template and targeting the Nios II Ethernet Standard Design Example.
7. To build the executable and linkable format (ELF) file (`.elf`) for this application, right-click the **Count Binary** project and select **Build Project**.
8. Download the `.elf` file to your board by right-clicking **Count Binary** project and selecting **Run As ➤ Nios II Hardware**.
   - The LEDs on your board provide a new light show.
9. Type the following:

```
system-console; #Start System Console.

#Set the processor service path to the Nios II processor.
set niosii_proc [lindex [get_service_paths processor] 0]

set claimed_proc [claim_service processor $niosii_proc mylib]; #Open the
service.

processor_stop $claimed_proc; #Stop the processor.
#The LEDs on your board freeze.

processor_run $claimed_proc; #Start the processor.
#The LEDs on your board resume their previous activity.

processor_stop $claimed_proc; #Stop the processor.

close_service processor $claimed_proc; #Close the service.
```

- The `processor_step`, `processor_set_register`, and `processor_get_register` commands provide additional control over the Nios II processor.

Related Information

- Nios II Ethernet Standard Design Example
- Nios II Gen2 Software Developer's Handbook

7.8.1.1. Processor Commands

Table 46. Processor Commands

Command [2]	Arguments	Function
`processor_download_elf`	*<service-path>* *<elf-file-path>*	Downloads the given Executable and Linking Format File (`.elf`) to memory using the master service associated with the processor. Sets the processor's program counter to the `.elf` entry point.
`processor_in_debug_mode`	*<service-path>*	Returns a non-zero value if the processor is in debug mode.
`processor_reset`	*<service-path>*	Resets the processor and places it in debug mode.
`processor_run`	*<service-path>*	Puts the processor into run mode.
`processor_stop`	*<service-path>*	Puts the processor into debug mode.
`processor_step`	*<service-path>*	Executes one assembly instruction.
`processor_get_register_names`	*<service-path>*	Returns a list with the names of all of the processor's accessible registers.
`processor_get_register`	*<service-path>* *<register_name>*	Returns the value of the specified register.
`processor_set_register`	*<service-path>* *<register_name>* *<value>*	Sets the value of the specified register.

Related Information

Nios II Processor Example on page 192

7.9. Running System Console in Command-Line Mode

You can run System Console in command line mode and either work interactively or run a Tcl script. System Console prints the output in the console window.

(2) If your system includes a Nios II/f core with a data cache, it may complicate the debugging process. If you suspect the Nios II/f core writes to memory from the data cache at nondeterministic intervals; thereby, overwriting data written by the System Console, you can disable the cache of the Nios II/f core while debugging.

- `--cli`—Runs System Console in command-line mode.
- `--project_dir=<project dir>`—Directs System Console to the location of your hardware project. Also works in GUI mode.
- `--script=<your script>.tcl`—Directs System Console to run your Tcl script.
- `--help`— Lists all available commands. Typing `--help` <*command name*> provides the syntax and arguments of the command.

System Console provides command completion if you type the beginning letters of a command and then press the **Tab** key.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.10. Using System Console Commands

You can use System Console commands to control hardware debug and testing with the command-line or scripting. Use System Console commands to identify a System Console service by its path, to open and close a connection, add a service, and a variety of other System Console controls.

Note: For a complete reference of currently supported System Console and toolkit commands, refer to the *System Console and Toolkit Tcl Command Reference Manual*.

The following steps show initiation of a simple service connection:

1. Identify a service by specifying its path with the `get_service_paths` command.
2. Open a connection to the service with the `claim_service` command.
3. Use Tcl and System Console commands to test the connected device.
4. Close a connection to the service with the `close_service` command

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.11. Using Toolkit Tcl Commands

For advanced users, System Console also supports Tcl commands that allow you to define and operate your own custom toolkits.

You can use the Toolkit Tcl commands to add and set the toolkit requirements and properties, and to retrieve accessible toolkit modules, systems, and services at the command-line or with scripting.

Note: For a complete reference of currently supported System Console and toolkit commands, refer to the *System Console and Toolkit Tcl Command Reference Manual*.

Related Information

System Console and Toolkit Tcl Command Reference Manual

7.12. Analyzing and Debugging Designs with the System Console Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2023.10.02	23.3	• Updated product family name to "Intel Agilex 7." • Revised the existing information and added a new example on how to locate the correct service path using full hierarchy name in *Locating Available Services* topic.
2022.12.12	22.4	• Added new *Customizing, Saving, and Resetting the System Console Layout* topic for new layout manager feature.
2022.07.08	22.1	• Fixed broken links in the following topics: — "Introduction to System Console" — "MATLAB and Simulink* in a System Verification Flow" — "System Console Example" — "Nios II Gen 2 Processor Example"
2021.06.21	21.2	• Moved System Console and Toolkit Tcl command descriptions to *System Console and Toolkit Tcl Command Reference Manual* and provided links to this new comprehensive document. • Replaced non-inclusive terms with "host" and "agent" inclusive terms for Avalon memory mapped interface descriptions and related GUI elements. • Added toolkit definition to *Introduction to System Console* topic. • Revised *System Console Tools* figure. • Revised wording of *Autosweep View* topic for clarity. • Added details to explanation of legacy toolkits in *Available System Debugging Toolkits* • Added ISSP service to *Common Services for System Console* table. • Added link to download center.
2020.09.28	20.3	• Revised "Introduction to System Console" wording and block diagram. • Revised "Starting System Console" to consolidate all methods. • Revised "Toolkit Explorer Pane" to refer to launching toolkits. • Revised "Autosweep View" to account for use with or without toolkit and export and import of settings. • Added new "Launching a Toolkit in System Console" topic. • Added new "Available System Debugging Toolkits" topic. • Added new Toolkit Tcl Commands section. • Reordered some topics and updated outdated screenshots.
2019.09.30	19.3	Made the following updates in the *Analyzing and Debugging Designs with System Console* chapter:

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Updated *System Console GUI* and *System Explorer Pane* topics to describe the new framework. • Added the following new topics to describe various panes and views added to the System Console: — *System Console Default Panes* — *Toolkit Explorer Pane* — *Filtering and Searching Interactive Instances* — *Creating Collections from the Toolkit Explorer* — *System Console Views* — *Main View* — *Link Pair View* — *Autosweep View* — *Dashboard View* — *Eye View* • Removed *Working with Toolkit* section completely since it was now outdated due to the implementation of new System Console framework.
2018.05.07	18.0.0	Removed obsolete section: *Board Bring-Up with System Console Tutorial*.
2017.05.08	17.0.0	• Created topic *Convert your Dashboard Scripts to Toolkit API*. • Removed *Registering the Service* Example from *Toolkit API Script Examples*, and added corresponding code snippet to *Registering a Toolkit*. • Moved *.toolkit Description File Example* under *Creating a Toolkit Description File*. • Renamed *Toolkit API GUI Example .toolkit File* to *.toolkit Description File Example*. • Updated examples on Toolkit API to reflect current supported syntax.
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	• Edits to Toolkit API content and command format. • Added Toolkit API design example. • Added graphic to *Introduction to System Console*. • Deprecated Dashboard. • Changed instances of *Quartus II* to *Intel Quartus Prime*.
October 2015	15.1.0	• Added content for Toolkit API — Required .toolkit and Tcl files — Registering and launching the toolkit — Toolkit discovery and matching toolkits to IP — Toolkit API commands table
May 2015	15.0.0	Added information about how to download and start System Console stand-alone.
December 2014	14.1.0	• Added overview and procedures for using ADC Toolkit on MAX 10 devices. • Added overview for using MATLAB/Simulink Environment with System Console for system verification.
June 2014	14.0.0	Updated design examples for the following: board bring-up, dashboard service, Nios II processor, design service, device service, monitor service, bytestream service, SLD service, and ISSP service.
November 2013	13.1.0	Re-organization of sections. Added high-level information with block diagram, workflow, SLD overview, use cases, and example Tcl scripts.
June 2013	13.0.0	Updated Tcl command tables. Added board bring-up design example. Removed SOPC Builder content.

continued...

Document Version	Intel Quartus Prime Version	Changes
November 2012	12.1.0	Re-organization of content.
August 2012	12.0.1	Moved Transceiver Toolkit commands to Transceiver Toolkit chapter.
June 2012	12.0.0	Maintenance release. This chapter adds new System Console features.
November 2011	11.1.0	Maintenance release. This chapter adds new System Console features.
May 2011	11.0.0	Maintenance release. This chapter adds new System Console features.
December 2010	10.1.0	Maintenance release. This chapter adds new commands and references for Qsys.
July 2010	10.0.0	Initial release. Previously released as the System Console User Guide, which is being obsoleted. This new chapter adds new commands.

8. Intel Quartus Prime Pro Edition User Guide Debug Tools Archives

For the latest and previous versions of this user guide, refer to *Intel Quartus Prime Pro Edition User Guide: Design Compilation*. If a software version is not listed, the guide for the previous software version applies.

A. Intel Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Intel Quartus Prime Pro Edition FPGA design flow.

Related Information

- Intel Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Pro Edition software, including managing Intel Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Intel Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Intel Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Intel Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Intel Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Intel Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Intel Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Intel Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Intel Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Intel Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Intel Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Timing Analyzer

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

1. Timing Analysis Introduction

Comprehensive timing analysis of your design allows you to validate circuit performance, identify timing violations, and drive the Fitter's placement of logic to meet your timing goals. The Quartus® Prime Timing Analyzer uses industry-standard constraint and analysis methodology to report on all data required times, data arrival times, and clock arrival times for all register-to-register, I/O, and asynchronous reset paths in your design.

The Timing Analyzer verifies that your design meets all required timing relationships to correctly function, and confirms actual signal arrival times against the constraints that you specify. This user guide provides an introduction to basic timing analysis concepts, along with step-by-step instructions for using the Quartus Prime Timing Analyzer.

1.1. What's New In This Version

- Updated throughout to reflect recent added support for DNI-related SDC-on-RTL constraints and post-synthesis Early Timing Analysis.
- Some timing reports now support opening the reported source file in a text editor, as Report Timing By Source Files describes.
- For change details, refer to the chapter revision histories in this document.

1.2. Timing Analysis Basic Concepts

This user guide introduces the following concepts to describe timing analysis:

Table 1. Timing Analyzer Terminology

Term	Definition
Arrival time	The Timing Analyzer calculates the data and clock arrival time versus the required time at register pins.
Cell	Device resource that contains look-up tables (LUT), registers, digital signal processing (DSP) blocks, memory blocks, or I/O elements. In Stratix® series devices, the LUTs and registers are contained in logic elements (LE) modeled as cells.
Clock	Named signal representing clock domains inside or outside of your design.
Clock-as-data analysis	More accurate timing analysis for complex paths that includes any phase shift associated with a PLL for the clock path, and considers any related phase shift for the data path.
Clock hold time	Minimum time interval that a signal must be stable on the input pin that feeds a data input or clock enable, after an active transition on the clock input.

continued...

Term	Definition
Clock launch and latch edge	The launch edge is the clock edge that sends data out of a register or other sequential element, and acts as a source for the data transfer. The latch edge is the active clock edge that captures data at the data port of a register or other sequential element, acting as a destination for the data transfer.
Clock pessimism	Clock pessimism refers to use of the maximum (rather than minimum) delay variation associated with common clock paths during static timing analysis.
Clock setup time	Minimum time interval between the assertion of a signal at a data input, and the assertion of a low-to-high transition on the clock input.
Maximum or minimum delay constraint	A constraint that specifies timing path analysis with a non-default setup or hold relationship.
Net	A collection of two or more interconnected components.
Node	Represents a wire carrying a signal that travels between different logical components in the design. Most basic timing netlist unit. Used to represent ports, pins, and registers.
Pin	Inputs or outputs of cells.
Port	Top-level module inputs or outputs; for example, a device pin.
Metastability	Metastability problems can occur when a signal transfers between circuitry in unrelated or asynchronous clock domains. The Timing Analyzer analyzes the potential for metastability in your design and can calculate the MTBF for synchronization register chains.
Multicorner analysis	Timing analysis of slow and fast timing corners to verify your design under a variety of voltage, process, and temperature operating conditions.
Multicycle paths	A data path that requires a non-default number of clock cycles for proper analysis.
Recovery and removal time	Recovery time is the minimum length of time for the deassertion of an asynchronous control signal relative to the next clock edge. Removal time is the minimum length of time the deassertion of an asynchronous control signal must be stable after the active clock edge.
Timing netlist	A Compiler-generated list of your design's synthesized nodes and connections. The Timing Analyzer requires this netlist to perform timing analysis.
Timing path	The wire connection (net) between any two sequential design nodes, such as the output of a register to the input of another register.

1.2.1. Timing Path and Clock Analysis

The Timing Analyzer measures the timing performance for all timing paths identified in your design. Prior to running full timing analysis, you can run post-synthesis early timing analysis to obtain an early view of the design core timing. For post-fit timing analysis, the Timing Analyzer requires a timing netlist that describes your design's nodes and connections for analysis and uses routing delays between core blocks represented by average interconnect delays. The post-synthesis timing delays reflect the delays of each type of connected core block.

Post-fit timing analysis also determines clock relationships for all register-to-register transfers in your design by analyzing the clock setup and hold relationship between the launch edge and latch edge of the clock.

1.2.1.1. The Timing Netlist

The Timing Analyzer uses the timing netlist data to determine the data and clock arrival time versus required time for all timing paths in the design. Post-synthesis timing analysis utilizes a timing netlist that incorporates core blocks (including their

internal logic) and peripheral blocks (excluding their internal details). The Timing Analyzer estimates pre-synthesis routing delays by using an average interconnect model that the Analysis & Elaboration compilation stage generates. You can generate the post-fit timing netlist in the Timing Analyzer any time after running the Fitter.

The following figures illustrate division of a simple design schematic into timing netlist delays.

Figure 1. Simple Design Schematic

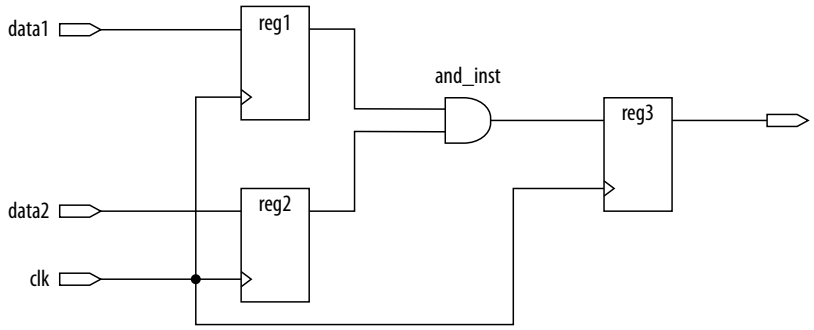
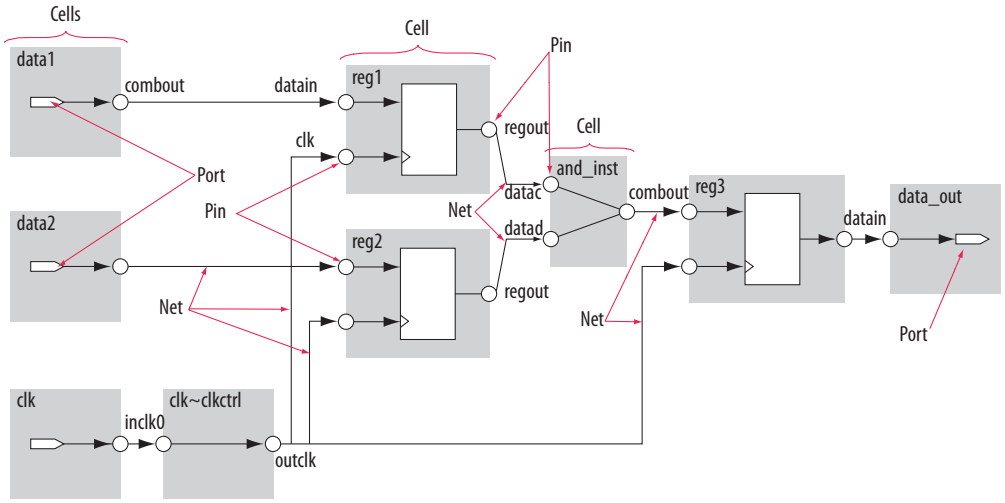


Figure 2. Division of Elements into Timing Netlist Delays



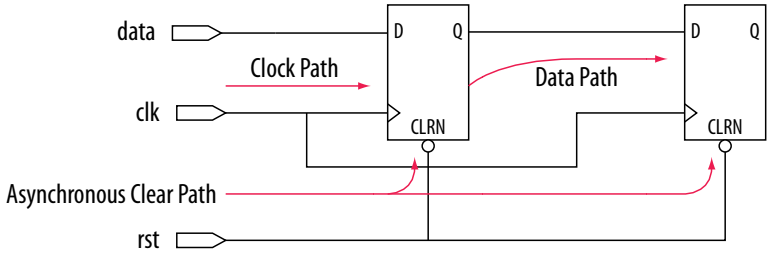
1.2.1.2. Timing Paths

Timing paths connect two design nodes, such as the output of a register to the input of another register.

Understanding the types of timing paths is important to timing closure and optimization. The Timing Analyzer recognizes and analyzes the following timing paths:

- **Edge paths**—connections from ports-to-pins, from pins-to-pins, and from pins-to-ports.
- **Clock paths**—connections from device ports or internally generated clock pins to the clock pin of a register.
- **Data paths**—connections from a port or the data output pin of a sequential element to a port or the data input pin of another sequential element.
- **Asynchronous paths**—connections from a port or asynchronous pins of another sequential element such as an asynchronous reset or asynchronous clear.

Figure 3. Path Types Commonly Analyzed by the Timing Analyzer



In addition to identifying various paths in a design, the Timing Analyzer analyzes clock characteristics to compute the worst-case requirement between any two registers in a single register-to-register path. You must constrain all clocks in your design before analyzing clock characteristics.

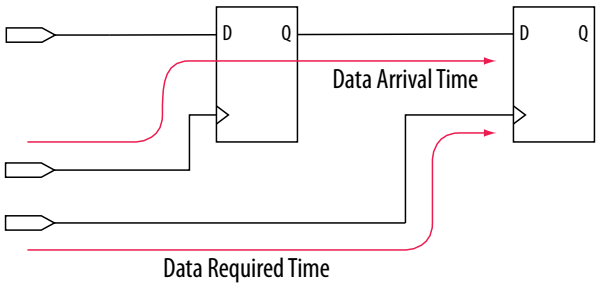
1.2.1.3. Data and Clock Arrival Times

After the Timing Analyzer identifies the path type, the Timing Analyzer can report data and clock arrival times at register pins.

The Timing Analyzer calculates data arrival time by adding the launch edge time to the delay from the clock source to the clock pin of the source register, the micro clock-to-output delay (μt_{CO}) of the source register, and the delay from the source register's data output (Q) to the destination register's data input (D).

The Timing Analyzer calculates data required time by adding the latch edge time to the sum of all delays between the clock port and the clock pin of the destination register. It includes any clock port buffer delays and subtracts the micro setup time (μt_{SU}) (or adds the micro hold time) of the destination register. Where the μt_{SU} is the intrinsic setup time of an internal register in the FPGA.

Figure 4. Data Arrival and Data Required Times



The basic calculations for data arrival and data required times including the launch and latch edges.

Figure 5. Data Arrival and Data Required Time Equations

$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Source Clock Delay} + \mu t_{CO} + \text{Register-to-Register Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Destination Clock Delay} - \mu t_{SU}$$

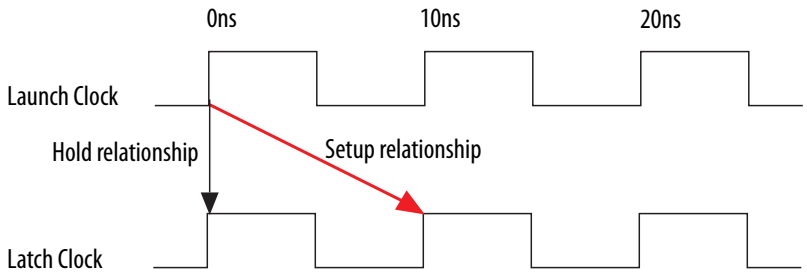
1.2.1.4. Launch and Latch Edges

All timing analysis requires the presence of one or more clock signals. The Timing Analyzer determines clock relationships for all register-to-register transfers in your design by analyzing the clock setup and hold relationship between the launch edge and latch edge of the clock.

The launch edge of the clock signal is the clock edge that sends data out of a register or other sequential element, and acts as a source for the data transfer. The latch edge is the active clock edge that captures data at the data port of a register or other sequential element, acting as a destination for the data transfer.

Figure 6. Setup and Hold Relationship for Launch and Latch Edges 10ns Apart

In this example, the launch edge sends the data from register `reg1` at 0 ns, and the register `reg2` captures the data when triggered by the latch edge at 10 ns. The data arrives at the destination register before the next latch edge.



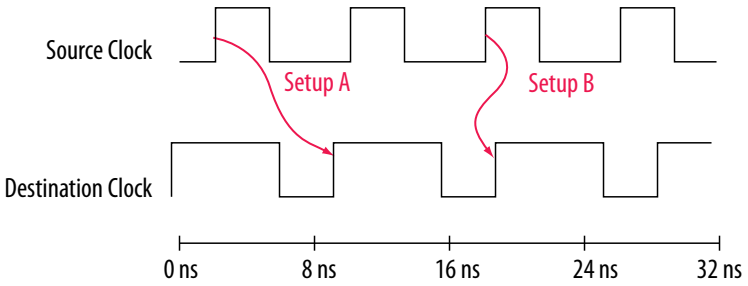
You must define all clocks in your design by assigning a clock constraint to each clock source node. These clock constraints provide the structure required for repeatable data relationships. If you do not constrain the clocks in your design, the Quartus Prime software analyzes all clocks as 1 GHz clocks to maximize timing based Fitter effort. To ensure realistic slack values, you must constrain all clocks in your design with real values.

1.2.2. Clock Setup Analysis

To perform a clock setup check, the Timing Analyzer determines a setup relationship by analyzing each launch and latch edge for each register-to-register path.

For each latch edge at the destination register, the Timing Analyzer uses the closest previous clock edge at the source register as the launch edge. The following figure shows two setup relationships, setup A and setup B. For the latch edge at 10 ns, the closest clock that acts as a launch edge is at 3 ns and has the setup A label. For the latch edge at 20 ns, the closest clock that acts as a launch edge is 19 ns and has the setup B label. The Timing Analyzer analyzes the most restrictive setup relationship, in this case setup B; if that relationship meets the design requirement, then setup A meets the requirement by default.

Figure 7. Setup Check



The Timing Analyzer reports the result of clock setup checks as slack values. Slack is the margin by which a circuit meets or does not meet the timing requirement. Positive slack indicates the margin by the circuit meets the requirement. Negative slack indicates the margin by which the circuit does not meet the requirement.

Figure 8. Clock Setup Slack for Internal Register-to-Register Paths

$$\begin{aligned}\text{Clock Setup Slack} &= \text{Data Required Time} - \text{Data Arrival Time}\\ \text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Register Delay}\\ \text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU} - \text{Setup Uncertainty}\end{aligned}$$

The Timing Analyzer performs setup checks using the maximum delay when calculating data arrival time, and minimum delay when calculating data required time. Some of the spread between maximum arrival path delays and minimum required path delays may be recoverable with path pessimism removal, as Timing Pessimism on page 16 describes.

Figure 9. Clock Setup Slack from Input Port to Internal Register

$$\begin{aligned}\text{Clock Setup Slack} &= \text{Data Required Time} - \text{Data Arrival Time}\\ \text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay} + \text{Input Maximum Delay} + \text{Port-to-Register Delay}\\ \text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU} - \text{Setup Uncertainty}\end{aligned}$$

Figure 10. Clock Setup Slack from Internal Register to Output Port

$$\begin{aligned}\text{Clock Setup Slack} &= \text{Data Required Time} - \text{Data Arrival Time}\\ \text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Output Port} - \text{Output Maximum Delay}\\ \text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Port Delay}\end{aligned}$$

1.2.3. Clock Hold Analysis

To perform a clock hold check, the Timing Analyzer determines a hold relationship for each possible setup relationship that exists for all source and destination register pairs. The Timing Analyzer checks all adjacent clock edges from all setup relationships to determine the hold relationships.

The Timing Analyzer performs two hold checks for each setup relationship. The first hold check determines that the data launched by the current launch edge is not captured by the previous latch edge. The second hold check determines that the data launched by the next launch edge is not captured by the current latch edge. From the possible hold relationships, the Timing Analyzer selects the hold relationship that is the most restrictive. The most restrictive hold relationship is the hold relationship with the smallest difference between the latch and launch edges and determines the minimum allowable delay for the register-to-register path. In the following example, the Timing Analyzer selects hold check A2 as the most restrictive hold relationship of two setup relationships, setup A and setup B, and their respective hold checks.

Figure 11. Setup and Hold Check Relationships

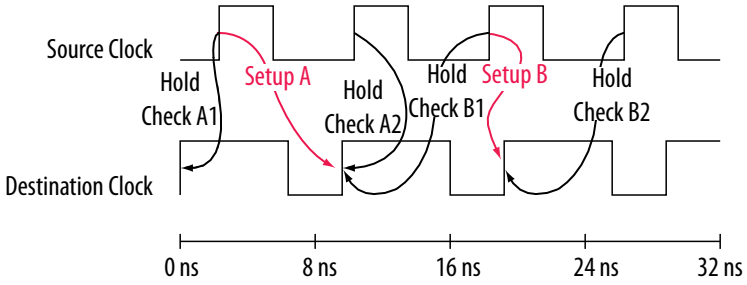


Figure 12. Clock Hold Slack for Internal Register-to-Register Paths

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Register Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} + \mu t_{H} + \text{Hold Uncertainty}$$

The Timing Analyzer performs hold checks using the minimum delay when calculating data arrival time, and maximum delay when calculating data required time.

Figure 13. Clock Hold Slack Calculation from Input Port to Internal Register

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay} + \text{Input Minimum Delay} + \text{Pin-to-Register Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay to Destination Register} + \mu t_{H}$$

Figure 14. Clock Hold Slack Calculation from Internal Register to Output Port

$$\text{Clock Hold Slack} = \text{Data Arrival Time} - \text{Data Required Time}$$
$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Pin Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Clock Network Delay} - \text{Output Minimum Delay}$$

1.2.4. Recovery and Removal Analysis

Recovery time is the minimum length of time for the deassertion of an asynchronous control signal relative to the next clock edge.

For example, signals such as `clear` and `preset` must be stable before the next active clock edge. The recovery slack calculation is similar to the clock setup slack calculation, but the calculation applies to asynchronous control signals.

Figure 15. Recovery Slack Calculation if the Asynchronous Control Signal is Registered

$$\begin{aligned}\text{Recovery Slack Time} &= \text{Data Required Time} - \text{Data Arrival Time}\\ \text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU}\\ \text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} + \text{Register-to-Register Delay}\end{aligned}$$

Figure 16. Recovery Slack Calculation if the Asynchronous Control Signal is not Registered

$$\begin{aligned}\text{Recovery Slack Time} &= \text{Data Required Time} - \text{Data Arrival Time}\\ \text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Destination Register} - \mu t_{SU}\\ \text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay} + \text{Input Maximum Delay} + \text{Port-to-Register Delay}\end{aligned}$$

Note: If the asynchronous reset signal is from a device I/O port, you must create an input delay constraint for the asynchronous reset port for the Timing Analyzer to perform recovery analysis on the path.

Removal time is the minimum length of time the deassertion of an asynchronous control signal must be stable after the active clock edge. The Timing Analyzer removal slack calculation is similar to the clock hold slack calculation, but the calculation applies to asynchronous control signals.

Figure 17. Removal Slack Calculation if the Asynchronous Control Signal is Registered

$$\begin{aligned}\text{Removal Slack Time} &= \text{Data Arrival Time} - \text{Data Required Time}\\ \text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay to Source Register} + \mu t_{CO} \text{ of Source Register} + \text{Register-to-Register Delay}\\ \text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Destination Register} + \mu t_{H}\end{aligned}$$

Figure 18. Removal Slack Calculation if the Asynchronous Control Signal is not Registered

$$\begin{aligned}\text{Removal Slack Time} &= \text{Data Arrival Time} - \text{Data Required Time}\\ \text{Data Arrival Time} &= \text{Launch Edge} + \text{Clock Network Delay} + \text{Input Minimum Delay of Pin} + \text{Minimum Pin-to-Register Delay}\\ \text{Data Required Time} &= \text{Latch Edge} + \text{Clock Network Delay to Destination Register} + \mu t_{H}\end{aligned}$$

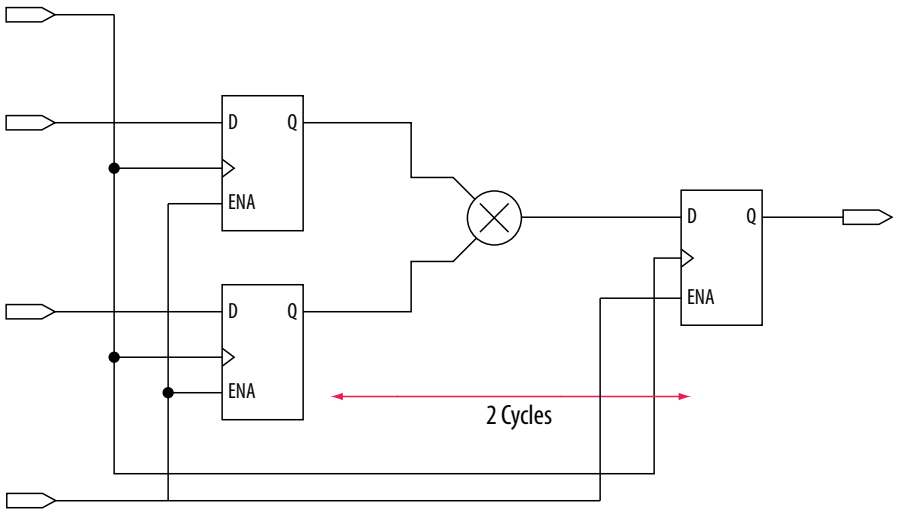
If the asynchronous reset signal is from a device pin, you must create an input delay constraint to the asynchronous reset pin for the Timing Analyzer to perform removal analysis on the path.

1.2.5. Multicycle Path Analysis

Multicycle paths are data paths that require an exception to the default setup or hold relationship, for proper analysis. For example, a register that requires data capture on every second or third rising clock edge (multicycle exception), rather than requiring capture on every clock edge (default analysis).

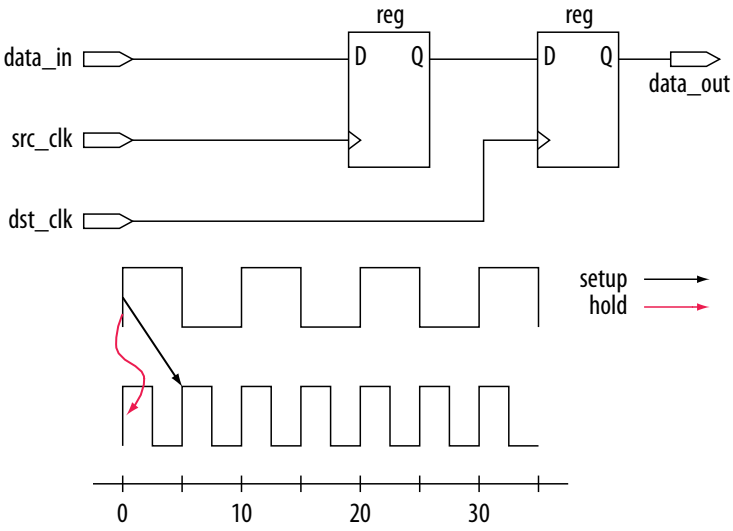
A multicycle path occurs between the input registers of a multiplier, and an output register with a destination that latches data on every other clock edge.

Figure 19. Multicycle Path



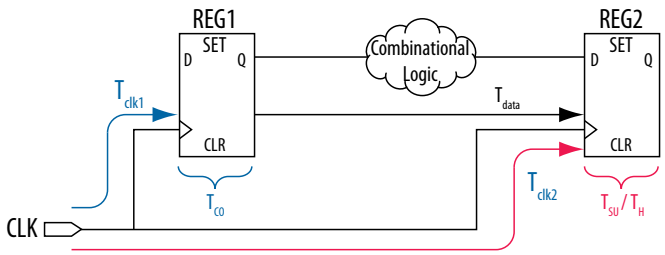
A register-to-register path is for the default setup and hold relationship. Also, for the respective timing diagrams for the source and destination clocks and the default setup and hold relationships, when the source clock, `src_clk`, has a period of 10 ns and the destination clock, `dst_clk`, has a period of 5 ns. The default setup relationship is 5 ns; the default hold relationship is 0 ns.

Figure 20. Register-to-Register Path and Default Setup and Hold Timing Diagram



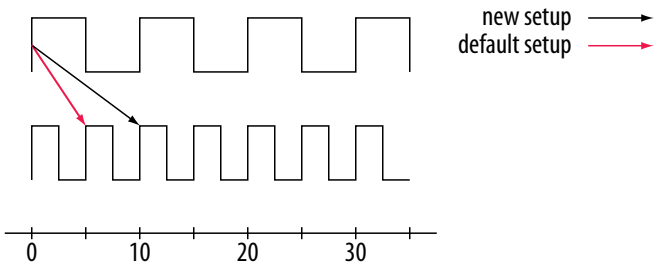
To accommodate the system requirements, you can modify the default setup and hold relationships by specifying a multicycle timing constraint to a register-to-register path.

Figure 21. Register-to-Register Path



The exception has a multicycle setup assignment of two to use the second occurring latch edge; in this example, to 10 ns from the default value of 5 ns.

Figure 22. Modified Setup Diagram



1.2.5.1. Multicycle Clock Hold

The number of clock periods between the clock launch edge and the latch edge defines the setup relationship.

By default, the Timing Analyzer performs a single-cycle path analysis. When analyzing a path, the Timing Analyzer performs two hold checks. The first hold check determines that the data that launches from the current launch edge is not captured by the previous latch edge. The second hold check determines that the data that launches from the next launch edge is not captured by the current latch edge. The Timing Analyzer reports only the most restrictive hold check. The Timing Analyzer calculates the hold check by comparing launch and latch edges.

Figure 23. Hold Check

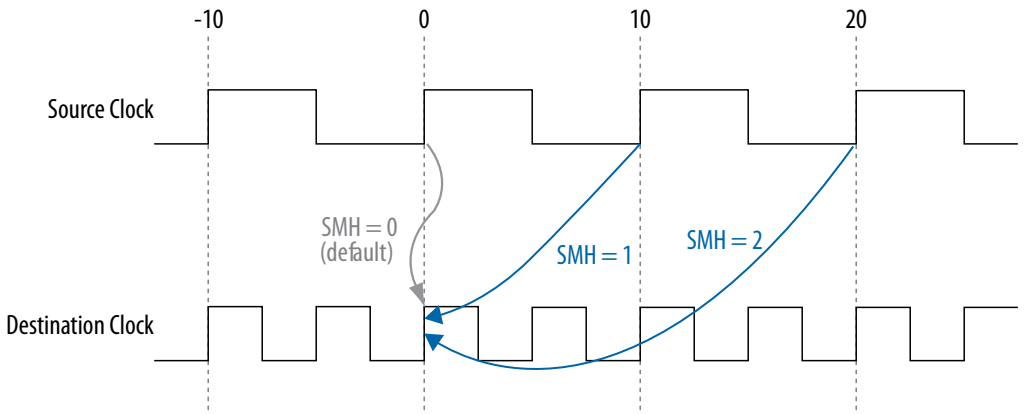
The Timing Analyzer uses the following calculation to determine the hold check.

$$\text{hold check 1} = \text{current launch edge} - \text{previous latch edge}$$
$$\text{hold check 2} = \text{next launch edge} - \text{current latch edge}$$

Tip: If a hold check overlaps a setup check, the hold check is ignored.

A start multicycle hold assignment modifies the launch edge of the source clock by moving the launch edge the number of clock periods you specify to the right of the default launch edge. The following figure shows various values of the start multicycle hold (SMH) assignment and the resulting launch edge.

Figure 24. Start Multicycle Hold Values



An end multicycle hold assignment modifies the latch edge of the destination clock by moving the latch edge the specific number of clock periods to the left of the default latch edge. The following figure shows various values of the end multicycle hold (EMH) assignment and the resulting latch edge.

Figure 25. End Multicycle Hold Values

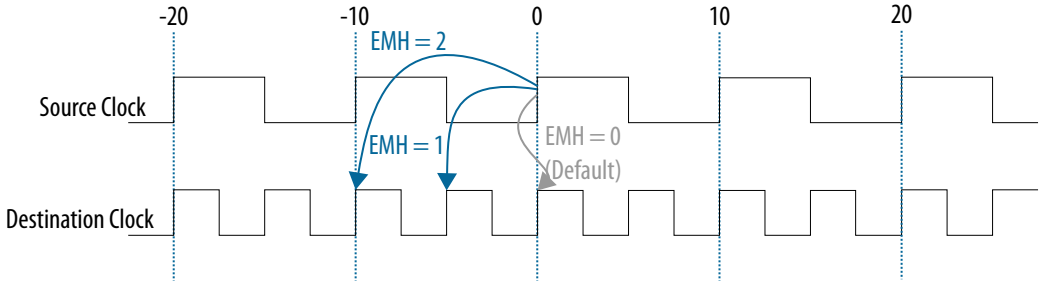
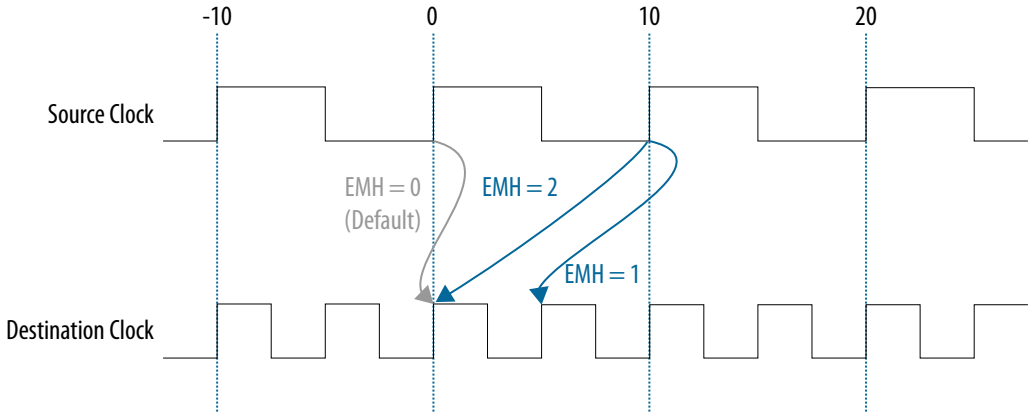


Figure 26. End Multicycle Hold Values the Timing Analyzer Reports

The following shows the hold relationship the Timing Analyzer reports for the negative hold relationship:

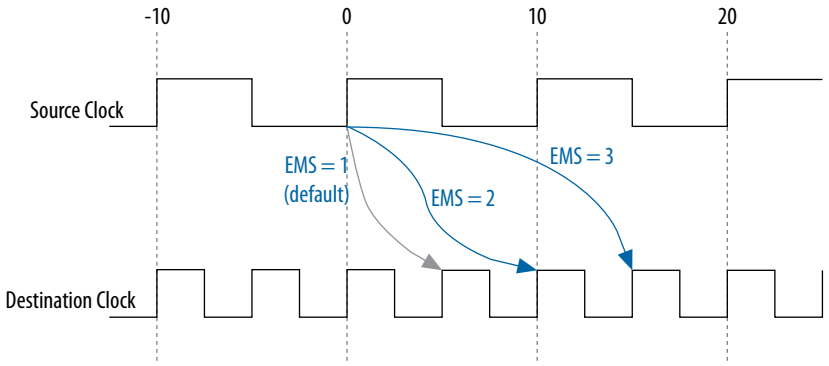


1.2.5.2. Multicycle Clock Setup

The setup relationship is the number of clock periods between the latch edge and the launch edge. By default, the Timing Analyzer performs a single-cycle path analysis, which results in the setup relationship being equal to one clock period (latch edge – launch edge). Applying a multicycle setup assignment, adjusts the setup relationship by the multicycle setup value. The adjustment value may be negative.

An end multicycle setup assignment modifies the latch edge of the destination clock by moving the latch edge the specified number of clock periods to the right of the determined default latch edge. The following figure shows various values of the end multicycle setup (EMS) assignment and the resulting latch edge.

Figure 27. End Multicycle Setup Values



A start multicycle setup assignment modifies the launch edge of the source clock by moving the launch edge the specified number of clock periods to the left of the determined default launch edge. A start multicycle setup (SMS) assignment with various values can result in a specific launch edge.

Figure 28. Start Multicycle Setup Values

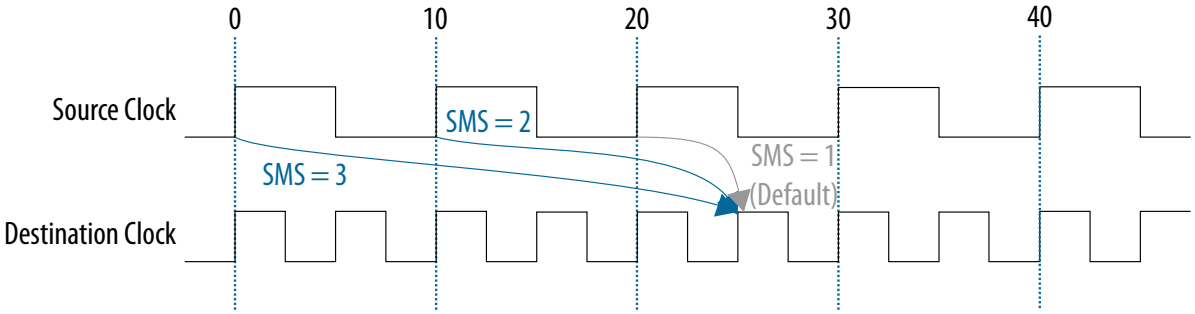
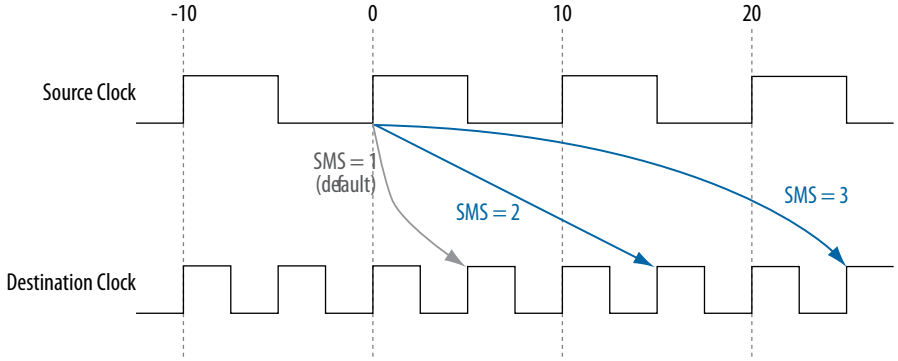


Figure 29. Start Multicycle Setup Values Reported by the Timing Analyzer

The following shows the negative setup relationship reported by the Timing Analyzer.



1.2.6. Metastability Analysis

Metastability problems can occur when a signal transfers between circuitry in unrelated or asynchronous clock domains because the signal does not meet setup and hold time requirements.

To minimize the failures due to metastability, circuit designers typically use a sequence of registers, also known as a synchronization register chain, or synchronizer, in the destination clock domain to resynchronize the data signals to the new clock domain.

The mean time between failures (MTBF) is an estimate of the average time between instances of failure due to metastability.

The Timing Analyzer analyzes the potential for metastability in your design and can calculate the MTBF for synchronization register chains. The Timing Analyzer then estimates the MTBF of the entire design from the synchronization chains the design contains.

In addition to reporting synchronization register chains found in the design, the Quartus Prime software also protects these registers from optimizations that might negatively impact MTBF, such as register duplication and logic retiming. The Quartus Prime software can also optimize the MTBF of your design if the MTBF is too low.

Related Information

- Report Metastability on page 138
- Step 1: Specify General Timing Analyzer Settings on page 29
- report_metastability

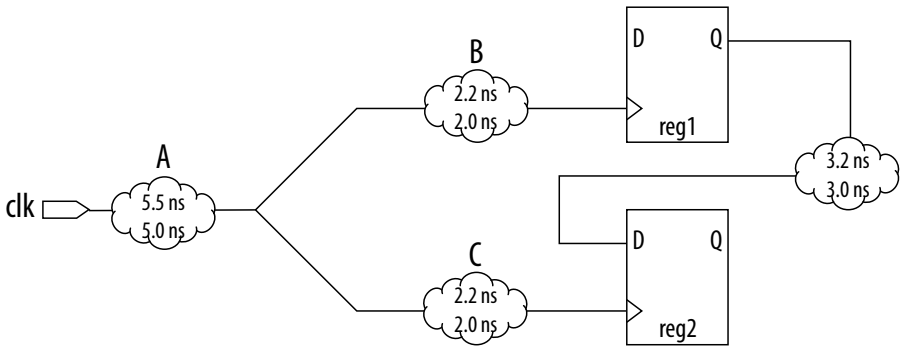
1.2.7. Timing Pessimism

Common clock path pessimism removal accounts for the minimum and maximum delay variation associated with common clock paths during static timing analysis by adding the difference between the maximum and minimum delay value of the common clock path to the appropriate slack equation.

Minimum and maximum delay variation can occur when timing analysis uses two different delay values for the same clock path. For example, in a simple setup analysis, the maximum clock path delay to the source register determines the data

arrival time. The minimum clock path delay to the destination register determines the data required time. However, if the clock path to the source register and to the destination register share a common clock path, both the maximum delay and the minimum delay model the common clock path during timing analysis. The use of both the minimum delay and maximum delay results in an overly pessimistic analysis since two different delay values, the maximum and minimum delays, cannot be used to model the same clock path.

Figure 30. Typical Register to Register Path

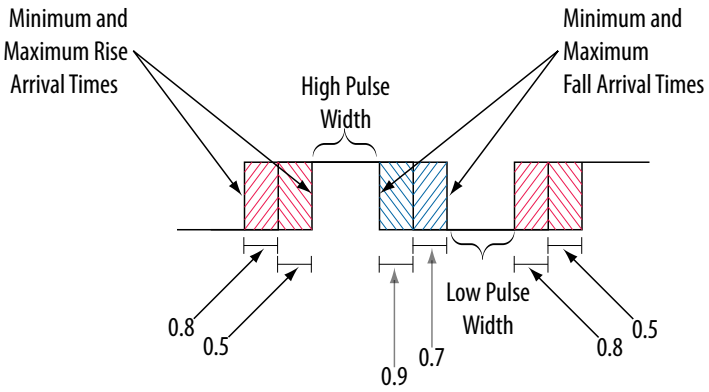


Segment A is the common clock path between `reg1` and `reg2`. The minimum delay is 5.0 ns; the maximum delay is 5.5 ns. The difference between the maximum and minimum delay value equals the common clock path pessimism removal value; in this case, the common clock path pessimism is 0.5 ns. The Timing Analyzer adds the common clock path pessimism removal value to the appropriate slack equation to determine overall slack. Therefore, if the setup slack for the register-to-register path in the example equals 0.7 ns without common clock path pessimism removal, the slack is 1.2 ns with common clock path pessimism removal.

You can also use common clock path pessimism removal to determine the minimum pulse width of a register. A clock signal must meet a register's minimum pulse width requirement for recognition by the register. A minimum high time defines the minimum pulse width for a positive-edge triggered register. A minimum low time defines the minimum pulse width for a negative-edge triggered register.

Clock pulses that violate the minimum pulse width of a register prevent data from latching at the data pin of the register. To calculate the slack of the minimum pulse width, the Timing Analyzer subtracts the required minimum pulse width time from the actual minimum pulse width time. The Timing Analyzer determines the actual minimum pulse width time by the clock requirement you specify for the clock that feeds the clock port of the register. The Timing Analyzer determines the required minimum pulse width time by the maximum rise, minimum rise, maximum fall, and minimum fall times.

Figure 31. Required Minimum Pulse Width time for the High and Low Pulse



With common clock path pessimism, the minimum pulse width slack can increase by the smallest value of either the maximum rise time minus the minimum rise time, or the maximum fall time minus the minimum fall time. In the example, the slack value can increase by 0.2 ns, which is the smallest value between 0.3 ns (0.8 ns – 0.5 ns) and 0.2 ns (0.9 ns – 0.7 ns).

1.2.8. Clock-As-Data Analysis

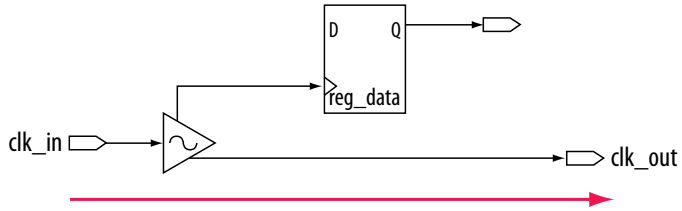
The majority of FPGA designs contain simple connections between any two nodes, known as either a data path or a clock path.

A data path is a connection between the output of a synchronous element to the input of another synchronous element.

A clock is a connection to the clock pin of a synchronous element. However, for more complex FPGA designs, such as designs that use source-synchronous interfaces, this simplified view is no longer sufficient. The Timing Analyzer performs clock-as-data analysis in circuits with elements such as clock dividers and DDR source-synchronous outputs.

You can treat the connection between the input clock port and output clock port as a clock path or a data path. Simplified Source Synchronous Output shows a design where the path from port `clk_in` to port `clk_out` is both a clock and a data path. The clock path is from the port `clk_in` to the register `reg_data` clock pin. The data path is from port `clk_in` to the port `clk_out`.

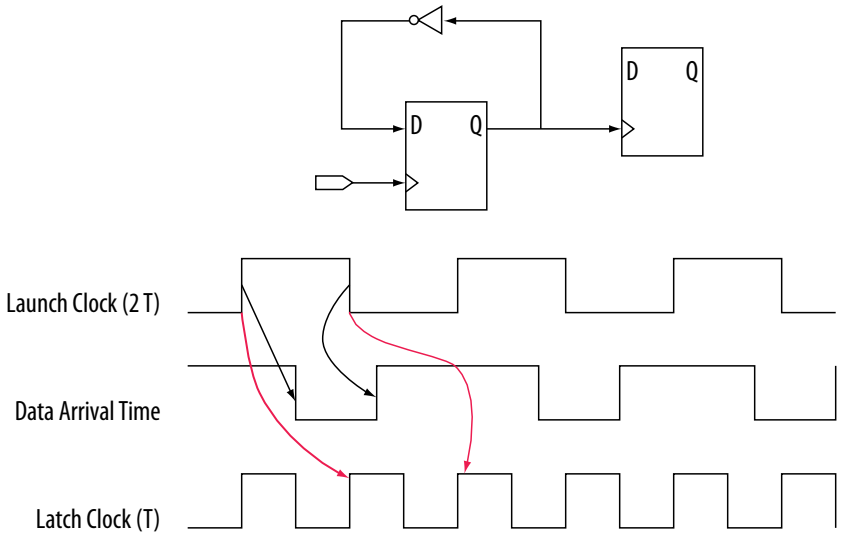
Figure 32. Simplified Source Synchronous Output



With clock-as-data analysis, the Timing Analyzer provides a more accurate analysis of the path based on user constraints. In the clock path analysis, the Timing Analyzer includes any phase shift associated with the phase-locked loop (PLL). For the data path analysis, the Timing Analyzer includes any phase shift associated with the PLL, rather than ignoring the phase shift.

The clock-as-data analysis also applies to internally generated clock dividers. In the following figure, the waveforms are for the inverter feedback path, analyzed during timing analysis. The output of the divider register determines the launch time, and the clock port of the register determines the latch time.

Figure 33. Clock Divider



1.2.9. Multicorner Timing Analysis

You can direct the Timing Analyzer to perform multicorner timing analysis to verify your design under different voltage, process, and temperature operating conditions.

To ensure that no violations occur under different timing conditions (models) during device operation, you must perform static timing analysis under all available operating conditions.

You specify the operating conditions in the Timing Analyzer prior to running analysis.

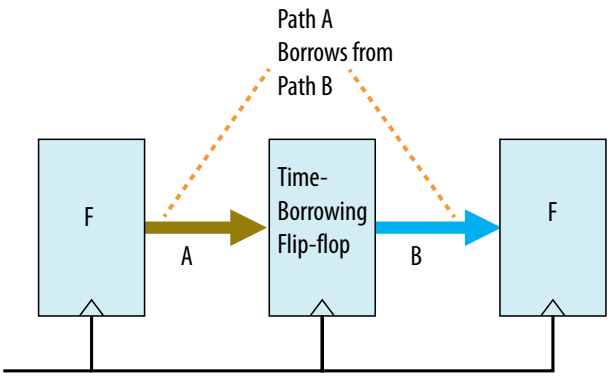
Related Information

Setting the Operating Conditions for Timing Analysis on page 38

1.2.10. Time Borrowing

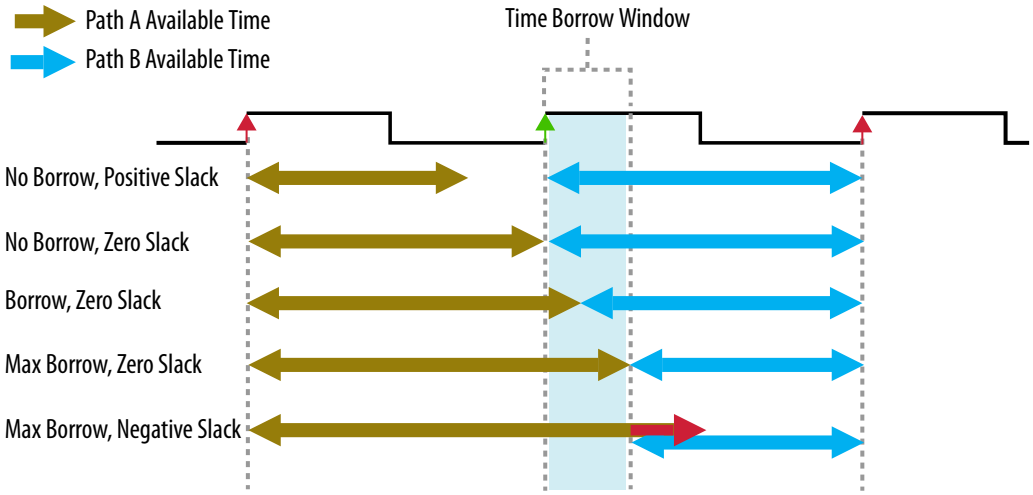
Time borrowing can improve performance by enabling the path ending at a time-borrowing flip-flop or latch to "borrow" time from the next path in the register pipeline. The borrowed time subtracts from the next path, resulting in the same cumulative timing. In this way, time borrowing can shift slack to more critical parts of the design. Without time borrowing, the Timing Analyzer analyzes each path independently and normally assumes exactly one clock cycle for each transfer.

Figure 34. Time Borrowing Example



Some of the flip-flops in Stratix 10 and Arria® 10 devices allow time borrowing. The exact size of the available time borrow window depends on hardware settings. The Fitter (Finalize) stage automatically configures the appropriate borrow window for each time-borrowing flip-flop, based on hardware restrictions and the available hold slack.(1)

Figure 35. Time Borrowing with Various Slack Conditions and Borrow Values



Intel FPGA devices generally support only a few borrow window sizes. For example, Stratix 10 devices support narrow, medium, and wide. Typically, groups of several flip-flops must share the same setting. The actual borrowed amount is completely flexible within a given borrow window. The Timing Analyzer calculates the borrowed amount separately for each operating condition, clock, and signal rise and fall edge. Selecting a wider borrow window reduces hold slack. The Compiler only selects wider settings if hold slack allows. Furthermore, if the Compiler determines that a narrower window is sufficient for a given group of registers (based on the optimal time borrowing solution), the Compiler uses the narrower window, even if there is sufficient hold slack for a wider window.

(1) In Partial Reconfiguration designs, additional restrictions may apply to time-borrowing window size.

For a given borrow window size, the exact size of the borrow window may depend on the register input (for example, `d` or `sclr`), the edge of the incoming signal (rising or falling), the device speed grade, and operating conditions.

You can enable automatic implementation of time borrowing without making any RTL changes. Once enabled, the Fitter automatically configures the window size. The Fitter also determines the optimal time borrow amount within the available borrow window for any design registers that the Fitter places in time-borrowing flip-flops.

Proper timing analysis of designs that contain level-sensitive latches typically requires time borrowing. However, the automatic Fitter time borrowing optimizations do not apply for level-sensitive latches, as Time Borrowing with Latches describes in detail.

Related Information

- Enabling Time Borrowing Optimization on page 23
- Report Time Borrowing Data on page 158

1.2.10.1. Time Borrowing Limitations

Time borrowing optimization, which occurs in the Fitter (Finalize) stage, cannot occur for the following registers. For these registers, the time borrowing optimization is zero, and the maximum operating frequency in the **Fmax Summary** reports include zero time borrowing:

- Any register that is the source of a cross-clock transfer
- Any register that is the source of a `set_max_skew` or `set_max_delay` assignment
- Any register in a clock domain with one or more level-sensitive latch

Furthermore, registers that are destinations of cross-clock transfers, `set_max_skew` or `set_max_delay` constraints do not have borrowing values that are optimized for such transfers (but may still have non-zero borrowing from other transfers).

If any such registers are on the critical timing path, you can possibly report better performance by enabling Dynamic time borrowing mode, which reports time borrowing for all borrow-capable registers. Dynamic time borrowing mode can then provide a more accurate (less pessimistic) analysis, but only at a specific set of clock frequencies that you specify in the `.sdc`.

To view time borrowing results for such registers:

1. Set the clock frequency in the `.sdc` file to a value higher than the clock frequency that **Fmax Summary** reports.
2. Reset your design and read the SDC file again in the Timing Analyzer.
3. Run the `update_timing_netlist -dynamic_borrow` command.
4. View the results in **Slack Summary** reports (**Reports ➤ Slack** to determine if timing passes. The **Fmax Summary** report does not reflect any gains from dynamic borrowing.

Note: Time borrowing is similar to the beneficial clock skew technique, whereby you delay the clock to a given register, giving more time to the incoming paths at the expense of the outgoing paths. However, time borrowing and beneficial clock skew have the following important differences:

- Time borrowing offers more flexibility than clock skew in distributing setup slack. Skewing is typically limited to fixed increments. You can use borrowing to shift any amount of slack (however small) from one side of the register to the other, as long as the amounts fit within the available borrow window. Furthermore, borrow amounts calculate separately for each operating condition, making it possible to shift the optimum amount of slack for each operating condition. This type of shifting is not possible with skewing.
- You can use beneficial clock skew to increase Hold slack on outgoing paths from the register where the clock is skewed. Time borrowing does not offer this benefit.

1.2.10.2. Time Borrowing with Latches

The Quartus Prime Timing Analyzer treats level-sensitive latches similar to registers. The Timing Analyzer treats the latch enable pin as a clock pin, while modifying the clock relationship appropriately.

You can run **Reports ➤ Constraint Diagnostics ➤ Check Timing** in the Timing Analyzer **Tasks** pane to view a list of the level-sensitive latches in your design.

Implementation of latch time borrowing requires that you enable Dynamic borrowing mode (`update_timing_netlist -dynamic_borrow`). Otherwise, the Timing Analyzer calculates zero time borrowing for latches. In Dynamic mode, the Timing Analyzer simply reports the amount of time borrowing that would physically happen in the circuit, given the clock frequencies you specify in SDC constraints, and does not actually optimize borrowing in any way.

For latches, the setup relationship is to the opening edge of the latch, which allows time borrowing. The hold relationship is to the closing edge of the latch. For example, a path from a positive register to another positive register has a default setup clock relationship of one clock period. A path from a positive register to a positive (open-high), level-sensitive latch has a default setup clock relationship of zero clock periods, plus any time borrow value.

The Timing Analyzer treats paths to and from a latch as two separate paths. For example, in a positive register--> positive latch--> negative register transfer, the Timing Analyzer does not analyze the overall register-->register transfer, even though you expect the latch to be transparent for the entire duration of the transfer. The Timing Analyzer analyzes and reports the paths to and from the latch separately.

The Timing Analyzer automatically computes the maximum amount of time borrowing available for each latch. Typically, the maximum amount of time borrowing available is roughly equivalent to half the clock period. The exact amount of time borrowing available is based on:

- The timing of opening and closing latch edges
- Physical latch implementation (closing-edge μt_{SU} of the latch)
- Clock uncertainty and other effects

The time borrowing never exceeds the maximum borrow value. However, you can specify a smaller maximum borrow time with the `set_max_time_borrow` SDC constraint. For example:

```
#Borrow at most 3ns at all "lat*" latches:
set_max_time_borrow 3 [get_registers lat*]
```

Specifying a clock with a negative borrow window can result in negative maximum borrowable time, which is equivalent to a minimum pulse width violation. For example, this condition can occur if half the clock period is smaller than the closing-edge μt_{SU} of the latch. If such a violation occurs, a warning indicates that the design cannot pass timing.

Note: Whether you use time borrowing or not, do not rely on the timing analysis **Fmax Summary** report for any clock domains with latches. The **Fmax Summary** values for such clock domains include no borrowing, and are therefore significantly pessimistic.

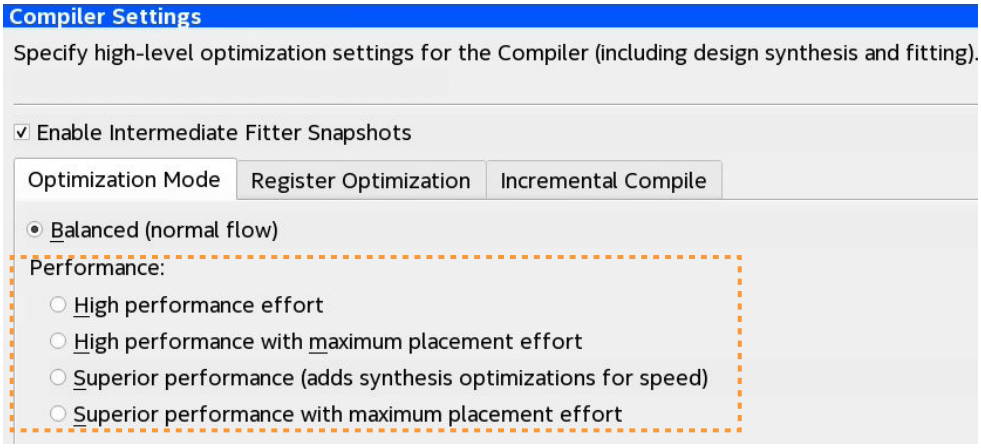
1.2.10.3. Enabling Time Borrowing Optimization

During any High or Superior **Performance** compilation, the Compiler automatically computes and stores **Optimal** time borrow values for Stratix 10 and Arria 10 designs during the Finalize stage. By default, the subsequent timing analysis results reflect the **Optimal** borrow values from the Finalize stage.

Follow these steps to enable time borrowing for supported devices:

1. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Optimization Mode**. Select any high or superior **Performance** setting.
2. Run the Fitter and Timing Analyzer, as Step 3: Run the Timing Analyzer on page 34 describes.
3. To generate reports showing time borrowing data, click **Reports ➤ Timing Slack ➤ Report Timing**. Time borrowing data appears on the critical path for a given clock domain, as Report Time Borrowing Data describes.

Figure 36. Performance Compiler Optimization Mode Settings



- To specify time borrowing optimization without changing the Compiler **Optimization Mode**, specify the following assignment in the project `.qsf`:

```
set_global_assignment -name ENABLE_TIME_BORROWING_OPTIMIZATION <ON|OFF>
```

- To manually specify the time borrow mode during timing analysis, run one of the following `update_timing_netlist` command options:

Table 2. Time Borrowing Modes

Time Borrowing Mode	Command Option	Default Mode For
Optimal—timing analysis includes optimal time borrow values from the Finalize stage. You can optionally add the `recompute_borrow` option to `update_timing_netlist` to recompute the borrow amounts, but not the borrow window sizes.	`update_timing_netlist`	High and Superior performance compilations for Stratix 10 and Arria 10 designs.
Dynamic—timing analysis reports the time borrowing that would physically occur on the device, with respect to your SDC constraints, without any optimization. That is, timing analysis applies as much borrowing as necessary to fix all negative slack. Timing analysis assumes maximum possible borrowing for any timing path where the maximum amount of time borrowing is insufficient to eliminate all negative slack. Only mode that allows borrowing for level-sensitive latches.	`update_timing_netlist -dynamic_borrow` `update_timing_netlist -loop_aware_dynamic_borrow`	None
Zero—timing analysis uses zero time borrowing.	`update_timing_netlist –no_borrow`	Unsupported devices, or any Compiler Optimization mode other than a **Performance** mode.

Note: Dynamic mode cannot yield the optimal results with overconstrained clocks, as overconstrained clocks result in excessive negative slack on almost every path. This condition causes use of maximum time borrowing everywhere, which is unlikely to be optimal. When using Partial Reconfiguration, if you compile the base design with time borrowing enabled, compile the implementation design(s) with time borrowing enabled. Otherwise, time borrowing amounts in the base design reset to zero, and the design may not pass timing. If this condition occurs, you can use the `update_timing_netlist –recompute_borrow` command to restore time borrowing amounts throughout the design.

Related Information

Time Borrowing on page 19

1.3. Timing Analysis Overview Document Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Updated *What's New In This Version* topic for DNI, SDC-on-RTL, and locating from report to source file.
2023.04.03	23.1	• Updated *What's New In This Version* topic for changes to entity-bound SDCs.
2022.03.28	22.1	• Added Top FAQ navigation to cover page. • Added new *What's New In This Version* topic for changes to Report Register Statistics report.
2020.09.28	20.3	• Revised setup arrow direction in multiple timing diagrams for consistency.
2020.04.13	20.1	• Added "Time Borrowing" section.
2019.09.30	19.3	• Updated "MultiCorner Timing Analysis" code example and stated limitation for operating conditions.
2019.07.15	19.2	• Updated "MultiCorner Analysis" for SmartVID timing models.
2018.09.24	18.1	Minor text enhancements for clarity and style.

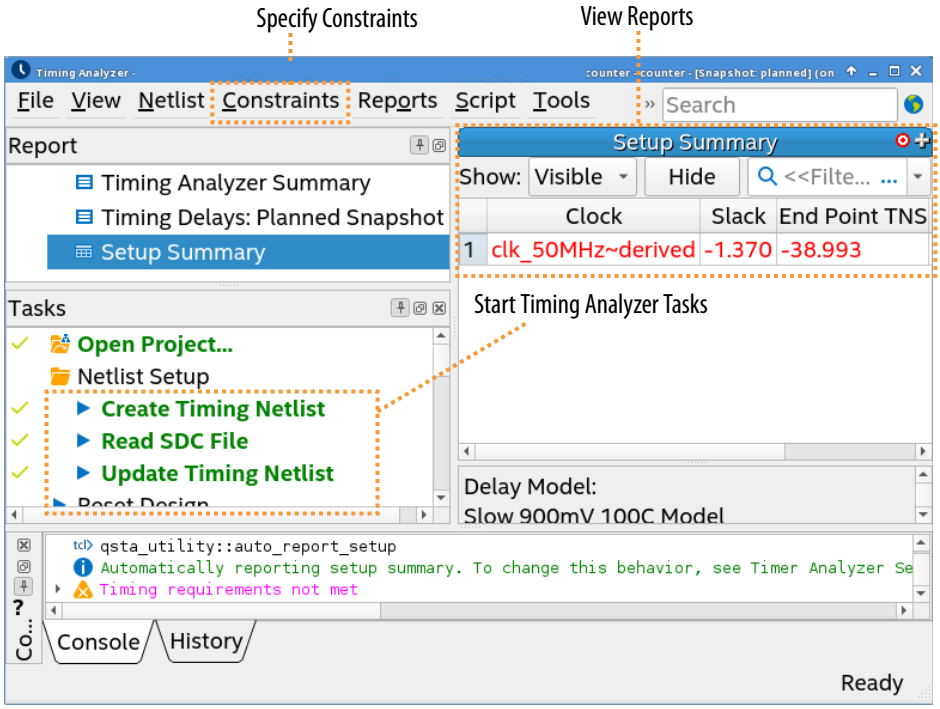
Table 3. Document Revision History

Date	Version	Changes
2016.10.31	16.1	• Implemented Intel rebranding.
2016.05.02	16.0	Corrected typo in Fig 6-14: Clock Hold Slack Calculation from Internal Register to Output Port
2015.11.02	15.1	Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1	Moved Multicycle Clock Setup Check and Hold Check Analysis section from the Timing Analyzer chapter.
June 2014	14.0	Updated format
June 2012	12.0	Added social networking icons, minor text updates
November 2011	11.1	Initial release.

2. Using the Quartus Prime Timing Analyzer

The Quartus Prime Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology. Use the Timing Analyzer GUI or command-line interface to constrain, analyze, and report results for all timing paths in your design.

Figure 37. Quartus Prime Timing Analyzer GUI



Related Information

- Timing Analyzer Quick-Start Tutorial: Quartus Prime Pro Edition
- Intel FPGA Technical Training

2.1. Using Timing Constraints throughout the Design Flow

To ensure accurate timing analysis, it is essential to define proper timing constraints that specify your design's clock frequency requirements, timing exceptions, and I/O timing specifications for comparison with actual conditions.

You specify the timing constraints in various Synopsys Design Constraint (SDC) files that you add to the project. You can define SDC files and run timing analysis at two key stages of the design compilation flow:

- After running **Analysis & Synthesis**—you can run Early Timing Analysis based on the synthesized design and initial SDC-on-RTL constraints you specify. If you're starting a new design, you can use SDC-on-RTL constraint methodology to learn benefits of timing analysis after synthesis. If you've already partially completed an existing design, it is best to use conventional SDC constraints.
- After running the **Fitter**—you can run post-fit timing analysis that accounts for actual path delays and the conventional SDC constraints you specify.

Post-Synthesis Early Timing Analysis Constraints

After running **Analysis & Synthesis**, you can run post-synthesis Early Timing Analysis based on the synthesized design and initial constraints that you define with SDC-on-RTL (`.rtlsdc`) or a synthesis-only conventional (`.sdc`). SDC-on-RTL allows you to define constraints using the same names in your design RTL, ensuring that your timing constraints names align closely with the RTL node names in the elaborated netlist.

Early timing analysis uses the initial SDC-on-RTL constraints that you specify to perform post-synthesis static timing analysis without needing to run the Fitter. The Compiler reads the constraints during Analysis & Elaboration, and then applies the SDC-on-RTL constraints for all downstream stages of compilation. For more details, refer to Specifying SDC-on-RTL Timing Constraints for step by step instructions.

As an alternative to SDC-on-RTL, you can define a conventional synthesis-only `.sdc` that applies the constraints only for design synthesis, as Specifying Synthesis-Only SDC Timing Constraints describes.

Post-Fit Timing Analysis Constraints

After running the Fitter's Plan, Place Route, Fitter (Finalize) stage, you can run post-fit timing analysis that accounts for actual path delays based on the Planned, Placed, or Routed design with constraints that you define in conventional SDC (`.sdc`) files. This post-fit timing analysis provides the most precise control over timing constraints.

You can define a conventional `.sdc` file directly in the Timing Analyzer GUI, or use the SDC file templates available using **Edit ➤ Insert Template**. You can alternatively define an `.sdc` file in any text editor and then integrate it into your project. Refer to Specifying Conventional SDC Timing Constraints.

The SDC File Types Supported table summarizes the differences between the various SDC file types and when the Quartus Prime software uses them.

Table 4. Supported SDC File Types

	SDC-on-RTL	Synthesis-Only SDC	Conventional SDC
Stage where constraints are read	Analysis & Elaboration	Synthesis	Fitter, Signoff
Stage where constraints are processed	Synthesis through Fitter	Synthesis only	Fitter, Signoff
QSF assignment	`RTL_SDC_FILE` (supports entities)	`SDC_FILE SDC_ENTITY_FILE -read_during_post_syn_and_post_fit_timing_analysis`	`SDC_FILE`
		`SDC_FILE SDC_ENTITY_FIL E -read_during_post_syn_and_not_post_fit_timing_analysis`	
Syntax supported	Tcl with SDC 2.1 commands	Tcl with Quartus Prime SDC commands	Tcl with Quartus Prime SDC commands
SDC 2.1-compliant	Yes	No	No
Target type	RTL	Quartus Prime timing graph	Quartus Prime timing graph
Hierarchical targets	Yes	No	No
Buried timing nodes (used by IP)	No	Core fabric only. Such nodes do not exist for the periphery in post-synthesis timing analysis.	Yes
STA command to load constraints	Executes the `read_sdc` or `import_sdc` command in any snapshot.	Executes the `read_sdc` command only during static timing analysis on the synthesized snapshot.	Executes the `read_sdc` command during static timing analysis on any fitter snapshot (plan, place, route, retime). Not loaded during synthesis.

Related Information

- Applying Timing Constraints on page 49
- Using Entity-Bound SDC Files on page 64
- Using Entity-Based SDC-on-RTL Constraints on page 57
- Creating Constraints in SDC-on-RTL SDC Files, Quartus Prime Pro Edition User Guide: Design Compilation

2.2. Timing Analysis Flow

The following describes high-level steps in the timing analysis flow. This section describes each step in detail:

- Step 1: Specify General Timing Analyzer Settings on page 29
- Step 2: Specify Timing Constraints on page 30
- Step 3: Run the Timing Analyzer on page 34
- Step 4: Analyze Timing Reports on page 40

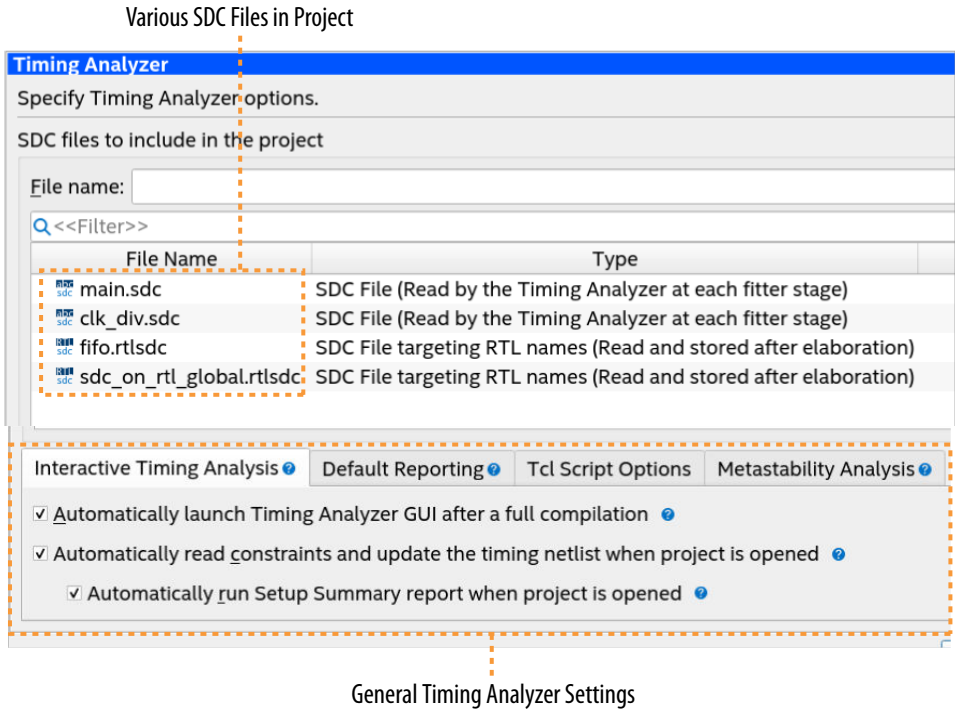
2.2.1. Step 1: Specify General Timing Analyzer Settings

Before running timing analysis, you must open an Quartus Prime project. You can then consider and specify general settings for timing analysis, as well as other project-wide Compiler settings that impact the timing analysis results.

To specify general Timing Analyzer settings, follow these steps:

1. To open an existing project, click **File ➤ Open Project**.
2. Click **Assignments ➤ Settings ➤ Timing Analyzer** to open the **Timing Analyzer** settings.

Figure 38. Timing Analyzer Page of Settings Dialog Box



3. In the **Timing Analyzer** page, specify one or more of the following general Timing Analyzer general settings:

Table 5. Timing Analyzer General Settings

Setting	Description
SDC files to include in the project	Specifies the name and processing order of timing constraint files in the project, such as conventional .sdc files and SDC-on-RTL (`.rtlsdc`) files. For more details, refer to Using Timing Constraints Effectively and Step 2: Specify Timing Constraints.
Interactive Timing Analysis	Specify options for automatically running timing analysis, reading constraints, and generating reports automatically. Turn on or off:

continued...

Setting	Description
	• **Automatically launch Timing Analyzer GUI after a full compilation** (default, on) • **Automatically read constraints and update the timing netlist when project is opened** in Timing Analyzer (default, on) • **Automatically run setup summary report when project is opened** in Timing Analyzer (default, on)
Default Reporting	Specify options to automatically **Report worst-case paths during compilation** (default, on). Specify the **Paths reported per clock domain** (default, 10), and whether to **Show routing** (default, off) in reports.
Tcl Script Options	**Tcl Script File name** specifies the file name for a custom timing analysis script. You can specify whether to **Run default timing analysis before running custom script**.
Metastability Analysis	Specifies how the Timing Analyzer identifies registers as being part of a synchronization register chain for metastability analysis.

4. Consider and specify project-wide Compiler settings that impact timing analysis:

Table 6. Compiler Settings Impacting Timing Analysis

Setting	Description	Location
Enable multicorner support for Timing Analyzer and EDA Netlist Writer(default, on)	Directs the Timing Analyzer to perform multicorner timing analysis by default, which analyzes the design against best-case and worst-case operating conditions.	**Assignments ➤ Settings ➤ Compilation Process Settings**
Optimization Mode (default, Balanced)	Specifies the focus of Compiler optimization efforts during synthesis and fitting. Specify a **Balanced** strategy, or optimize for **Performance**, **Area**, **Power**, **Routability**, or **Compile Time**.	**Assignments ➤ Settings ➤ Compiler Settings**
SDC Constraint Protection (default, off)	Verifies `.sdc` constraints in register merging. This option helps to maintain the validity of `.sdc` constraints through compilation.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**
Synchronization Register Chain Length (default, 3)	Specifies the maximum number of registers in a row that the Compiler considers as a synchronization chain. The Compiler considers these registers for metastability analysis. The Compiler prevents optimizations of these registers, such as retiming. When gate-level retiming is enabled, the Compiler does not remove these registers.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**
Optimize Design for Metastability (default, on)	This setting improves the reliability of the design by increasing its Mean Time Between Failures (MTBF). The Fitter increases the output setup slacks of synchronizer registers in the design. This slack can exponentially increase the design MTBF. This option only applies when using the Timing Analyzer for timing-driven compilation. Use the Timing Analyzer `report_metastability` command to review the synchronizers detected in your design and to produce MTBF estimates.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**

2.2.2. Step 2: Specify Timing Constraints

You can use any combination of the following to enter the timing constraints for your Quartus Prime project throughout the design flow, as Using Timing Constraints Effectively throughout the Design Flow describes.

- Specifying SDC-on-RTL Timing Constraints
- Specifying Conventional SDC Timing Constraints
- Specifying Synthesis-Only SDC Timing Constraints

2.2.2.1. Specifying SDC-on-RTL Timing Constraints

To specify SDC-on-RTL timing constraints for post-synthesis Early Timing Analysis, follow these steps:

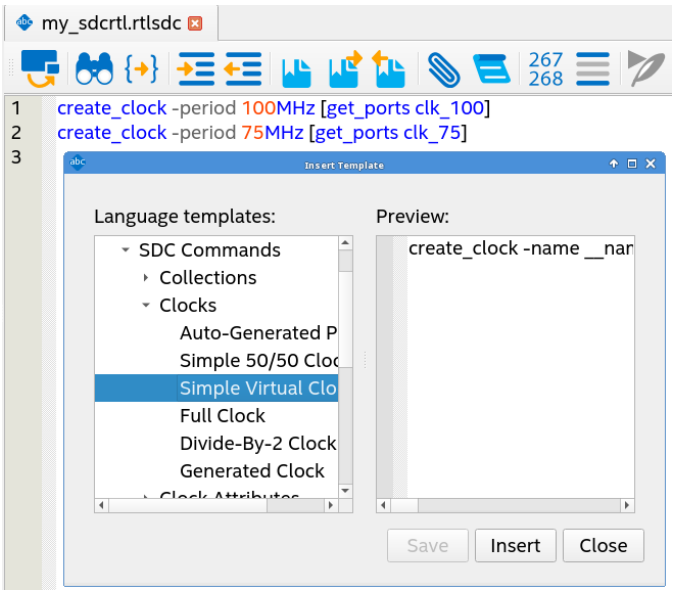
1. In the Quartus Prime software, click **File ➤ New** and then select the **SDC File Targeting RTL Names** (`.rtlsdc`) file type. The new file opens in the Text Editor.

   You can specify any extension for SDC-on-RTL constraints, but this document always uses the file extension `.rtlsdc` to distinguish from conventional SDC files.

2. In the Text Editor, define SDC-on-RTL constraints. You can click **Edit ➤ Insert Template ➤ Timing Analyzer** to insert available SDC templates. Alternatively, use any other text editor to enter the constraints and save as `.rtlsdc` file type.

   Note: SDC-on-RTL constraints support a subset of conventional SDC commands. The syntax and arguments of SDC-on-RTL constraints aligns with the SDC 2.1 standard. The Quartus Prime software may support more than just the SDC 2. 1 commands because of some Quartus Prime software-specific arguements.

Figure 39. Inserting SDC Templates



Note: The `-comment` argument in SDC-on-RTL allows you to add a constraint comment. This comment does not appear in timing analysis reports.

3. Save the `.rtlsdc` file in the Quartus Prime Text Editor, turning on the **Add File to Project** option.

Note: You can add any SDC file to the project at any time by clicking **Assignments ➤ Settings ➤ Timing Analyzer**.

4. Run post-synthesis Early Timing Analysis, as Running Post-Synthesis Early Timing Analysis describes.

   Note: As an alternative to SDC-on-RTL constraints, you can consider initially specifying synthesis-only constraints that apply only to the Analysis & Synthesis stage of compilation, as Specifying Synthesis-Only SDC Timing Constraints describes.

Related Information

- Using Entity-Based SDC-on-RTL Constraints on page 57
- Automatic Scope Example for SDC-on-RTL on page 61
- Manual Scope Example for SDC-on-RTL on page 64
- Using Synopsys* Design Constraint (SDC) on RTL Files

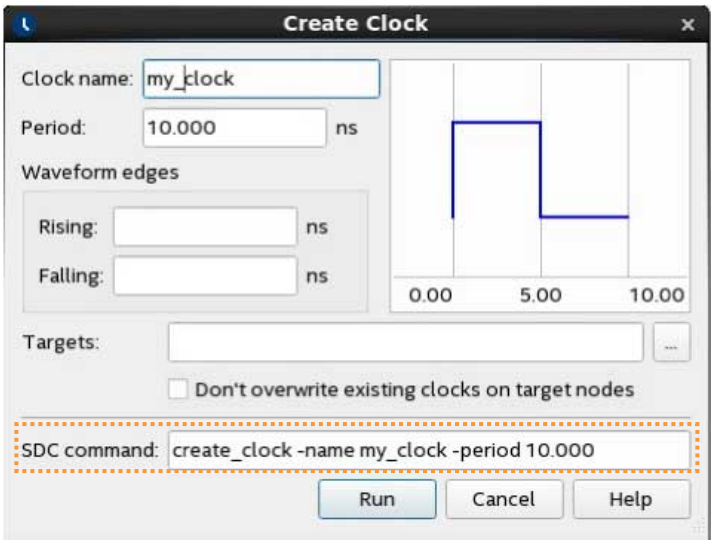
2.2.2.2. Specifying Conventional SDC Timing Constraints

To specify conventional timing constraints for post-fit timing analysis in the Timing Analyzer GUI, follow these steps:

Note: As an alternative to the Timing Analyzer GUI, you can specify conventional SDC constraints directly in a `.sdc` text file and add it to the project.

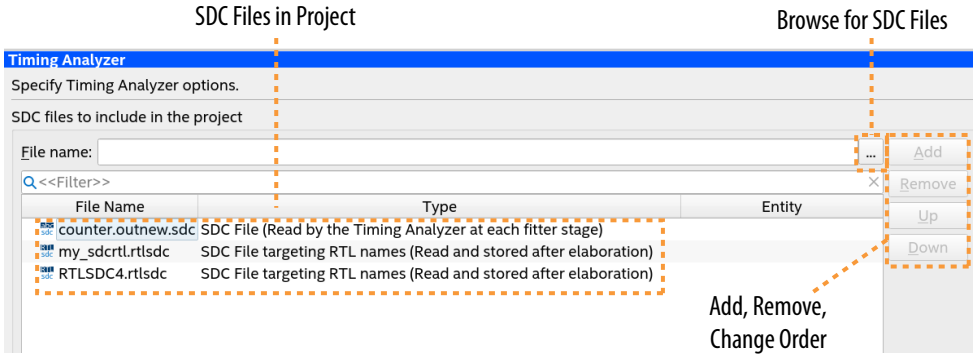
1. In the Timing Analyzer GUI, click the **Constraints** menu and then click the constraint that you want to define. You can start by adding the Recommended Initial Conventional SDC Constraints, and then iteratively modify the constraints and reanalyze the timing results.
2. In the GUI, specify values for the constraint and click **OK**. The **SDC command** field echoes the corresponding SDC command that applies as you enter the constraint value in the GUI.

Figure 40. Create Clock Dialog Defines Clock Constraints



3. When done entering constraints, click **Constraints ➤ Write SDC File** to save the constraints to an .sdc file.
4. To add the conventional .sdc file to your project, click **Assignments ➤ Settings ➤ Timing Analyzer**.

Figure 41. Adding an SDC to the Project



5. Run post-fit timing analysis, as Running Post-Fit Timing Analysis describes.

Related Information

- Using Entity-Bound SDC Files on page 64
- Automatic Scope Entity-bound Constraint Example on page 69
- Manual Scope Entity-bound Constraint Example on page 71

2.2.2.3. Specifying Synthesis-Only SDC Timing Constraints

You use this synthesis-only SDCs if you want to create custom SDC's that apply just for post-synthesis timing analysis, but you do not want the constraints to apply to other downstream stages of the compilation flow. If you want the constraints to persist post-synthesis, you can use SDC-on-RTL constraints, as Specifying SDC-on-RTL Timing Constraints describes.
Follow these steps to define a conventional `.sdc` file that applies only to the Analysis & Synthesis stage of compilation.

1. Create a conventional `.sdc` file that contains only the timing constraints for the Analysis & Synthesis stage of compilation, as Specifying Conventional SDC Timing Constraints describes.
2. To apply the conventional `.sdc` to the project for synthesis-only, add the following assignment to the project:

```
set_global_assignment -name SDC_ENTITY_FILE <file>.sdc /
   -entity <name> -read_during_post_syn_and_not_post_fit_timing_analysis
```

3. To run design synthesis and apply the constraints to the timing netlist, click **Analysis & Synthesis** on the Compilation Dashboard.
4. Click the **Open Timing Analyzer** icon next to **Analysis & Synthesis** on the Compilation Dashboard. The synthesis-only constraints now apply to only the static timing analysis of this synthesized snapshot.
5. Analyze the results of Early Timing Analysis, as Step 4: Analyze Timing Reports on page 40 describes.

2.2.3. Step 3: Run the Timing Analyzer

The Timing Analyzer generates reports that you can review to determine the performance of your design compared against your timing constraints. You can run timing analysis at two key stages of the design compilation flow:

- After running **Analysis & Synthesis**—you can run Early Timing Analysis based on the synthesized design and initial SDC-on-RTL constraints you specify.
- After running the **Fitter**—you can run post-fit timing analysis that accounts for actual path delays and the conventional SDC constraints you specify.

2.2.3.1. Running Post-Synthesis Early Timing Analysis

Running the Early Timing Analysis stage of compilation provides a preliminary view of your design's core timing. Before running Early Timing Analysis, you must setup your design RTL, the Quartus Prime project, specify timing constraints, and run the Compiler through the Analysis & Synthesis stage.

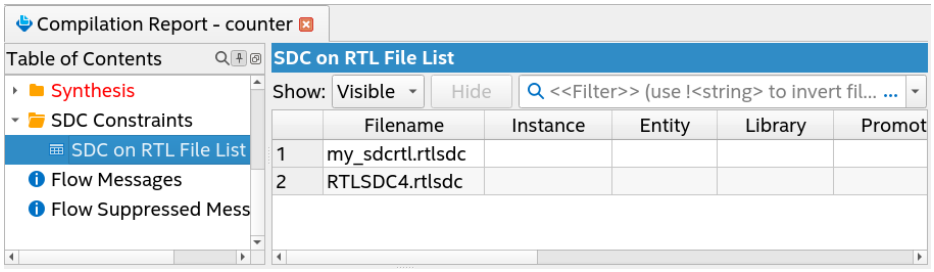
To run post-synthesis Early Timing Analysis, follow these steps:

1. Specify SDC-on-RTL timing constraints for Early Timing Analysis, as Specifying SDC-on-RTL Timing Constraints describes.

Note: As an alternative to SDC-on-RTL, you can define a synthesis-only `.sdc`, as Specifying Synthesis-Only SDC Timing Constraints describes.

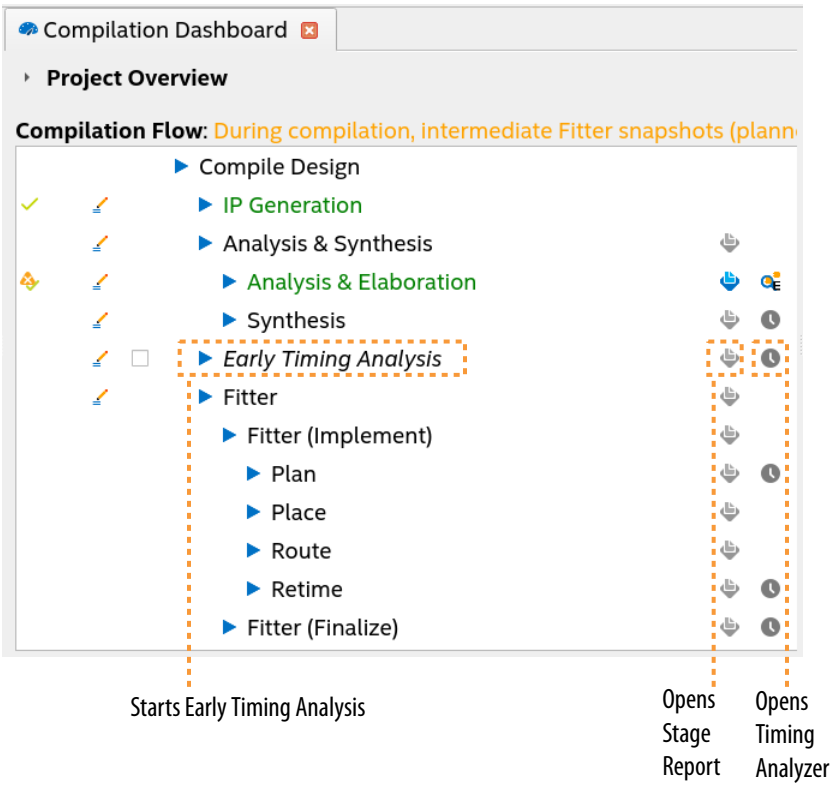
2. On the Compilation Dashboard, click **Analysis & Elaboration**. Analysis & Elaboration processes all SDC-on-RTL, applying the constraints to the design netlist. During Analysis & Elaboration, messages confirm that the Compiler appropriately applies each `.rtlsdc` file according to its assigned module.
3. To review the implementation of constraints when Elaboration & Analysis completes, click the **Open Compilation Reports** icon next to **Elaboration & Analysis** in the Compilation Dashboard.
4. Click the **SDC Constraints ➤ SDC-on-RTL File List** report to view all SDC-on-RTL files in the current project.

Figure 42. SDC-on-RTL File List Report



5. Click **Analysis & Synthesis** on the Compilation Dashboard. Analysis & Synthesis transforms the elaborated netlist into a node netlist for device resource mapping and generates a simplified device delay model that excludes precise Fitter-generated timing delays. This simplified delay model provides an early overview of the design delays based on block types that connect to a net. Analysis & Synthesis propagates SDC-on-RTL constraints to subsequent compilation stages, thereby applying to all subsequent Timing Analyzer runs.
6. To run Early Timing Analysis and view the results, double-click **Early Timing Analysis** on the Compilation Dashboard. The Compiler runs Analysis & Synthesis and then initializes the Timing Analyzer.

Figure 43. Running Early Timing Analysis from Compilation Dashboard



7. When Analysis & Synthesis completes, click the **Open Timing Analyzer** icon next to **Early Timing Analysis** on the Compilation Dashboard. The Timing Analyzer opens with the updated timing netlist loaded automatically.
8. In the Timing Analyzer reports, view the preliminary timing report data measured against your SDC-on-RTL constraints, such as the **Setup Summary**, **Create Generated Clocks**, and **Set False Path** reports. Refer to Step 4: Analyze Timing Reports.

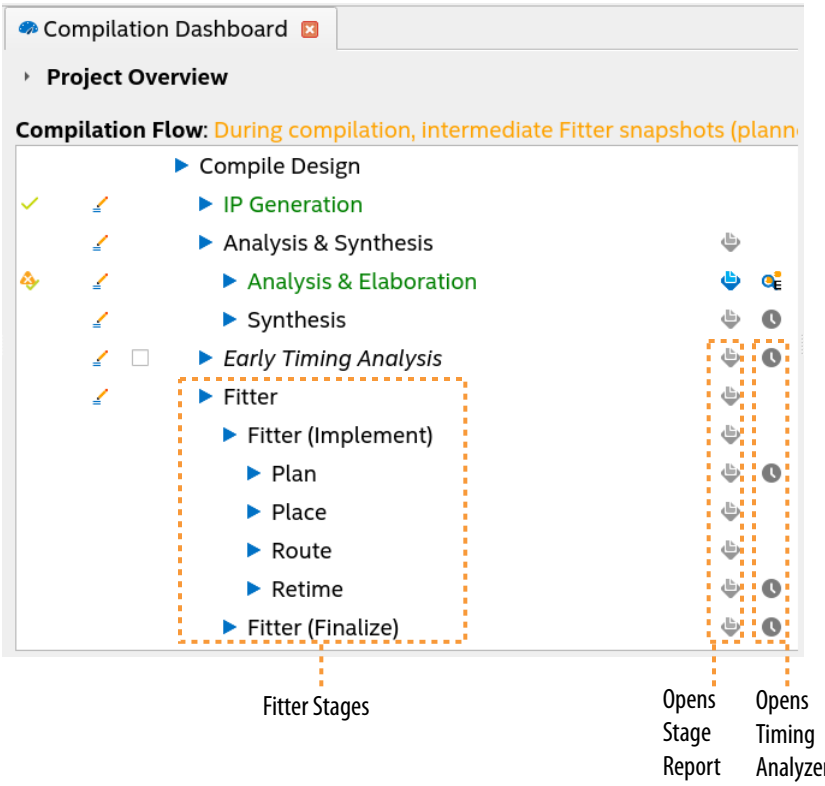
2.2.3.2. Running Post-Fit Timing Analysis

Before running post-fit timing analysis, you must run the Fitter to apply conventional SDC constraints to the post-fit timing netlist. The Fitter then attempts to place the logic of your design to adhere to the timing constraints you specify. After running the Fitter, the Timing Analyzer generates reports detailing the margin (slack) by which your design either meets or fails each constraint. The post-fit timing netlist accounts for actual path delays.

To run post-fit timing analysis, follow these steps:

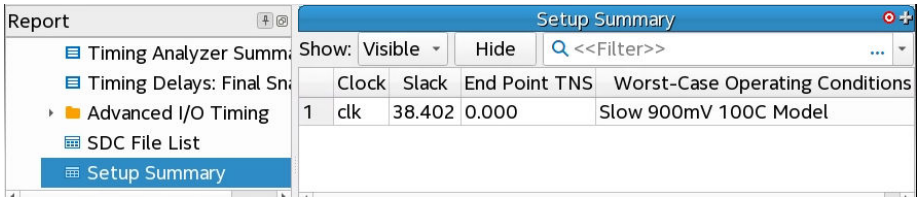
1. Specify conventional SDC timing constraints for post-fit timing analysis, as Specifying Conventional SDC Timing Constraints describes.
2. On the Compilation Dashboard, run any stage of the **Fitter** (**Plan**, **Place**, **Route**, **Retime**, **Fitter (Finalize)**) or run a full compilation.

Figure 44. Fitter Stages in Compilation Dashboard



3. When the Fitter completes, click the Timing Analyzer icon next to the completed stage in the Compilation Dashboard. The Setup Summary report opens by default in the Timing Analyzer.
4. Review the timing reports. To generate additional timing reports for analysis, click the **Reports** menu, and then click one of the submenu items to generate that report, as Step 4: Analyze Timing Reports describes.

Figure 45. Setup Summary Report

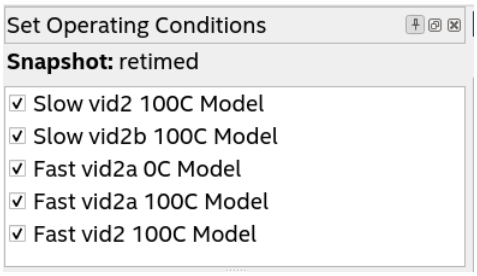


5. To run timing analysis under different operation conditions, click **Set Operating Conditions** on the **Tasks** pane and specify options, as Setting the Operating Conditions for Timing Analysis describes. By default, the Timing Analyzer generates reports for all supported operating conditions.
6. If you specify any settings or constraints that impact timing analysis, click **Update Timing Netlist** on the **Tasks** pane to apply the new constraints to the timing netlist.

2.2.3.2.1. Setting the Operating Conditions for Timing Analysis

Click **View ➤ Timing Corners** in the Timing Analyzer to specify the operating conditions for the timing analysis under different power and temperature ranges. The available operating conditions vary by device and speed grade. The various operating conditions that you can select represent the different "timing corners" in a multi-corner timing analysis.

Figure 46. Set Operating Conditions Panel



The Timing Analyzer displays the selectable operating conditions that are appropriate for your device in the **Set Operating Conditions** panel, according to the following timing model name conventions:

<process in speed grade> <voltage/vid> <temperature>

The following examples illustrate this naming convention:

- **Fast 900mV 40C Model**
  - **Fast**—timing model for the fastest process within a speed grade.
  - **900mV**—nominal 900mV timing model.
  - **40C**—low temperature (40 Celsius) timing model.

- **Slow vid<*n*> 100C Model**
  - **Slow**—timing model for the slowest process within a speed grade.
  - **vid**—timing model for analysis with SmartVID.
  - **<*n*>**—the device speed grade of the timing model.
  - **100C**—high temperature (100 Celsius) timing model.

Select one or more voltage and temperature combinations and double-click **Report Timing...** under **Timing Slack** in the **Tasks** pane to configure the generation of timing analysis reports for that model. The generated report shows the worst-case timing path slack across all operating conditions that you specify. After generating the report for that model, the report shows the worst-case timing path slacks across all operating conditions that you select.

You can use the following context menu options to generate or regenerate reports in the **Report** window:

- **Regenerate**—regenerates the report you select.
- **Regenerate All Out of Date**—regenerates all reports that are outdated because of SDC changes since the last generation.
- **Delete All Out of Date**—removes all outdated report data.

As an alternative to the GUI, you can run the `set_operating_conditions` command with the `-model`, `-speed`, `-temperature`, and `-voltage` options to specify the operating conditions. When running `set_operating_conditions`, you must only specify valid operating conditions for the current device. Run the `get_available_operating_conditions` command to return a list of all valid operating conditions for the current device.

The following example sets the operating condition to the slow timing model, with a voltage of 900 mV, and temperature of 100° C:

```
set_operating_conditions -model slow -temperature 100 -voltage 900
```

Related Information

Multicorner Timing Analysis on page 19

2.2.3.2.2. Promoting Critical Warnings to Errors

You can promote critical warnings to errors so that the Timing Analyzer halts on receiving the critical warnings (as it does for all errors). All critical warnings support promotion to error. However, you can only promote the message IDs of open projects.

1. In the **Message** dialog box, right-click on the critical warning you want to promote to an error.
2. Click **Message Promotion ➤ Promote Critical Message ID to Error**
3. To clear all promotions, click **Message Promotion ➤ Clear All Message Promotions**
4. Alternatively, promote or demote a critical warning in the `.qsf` with the following:

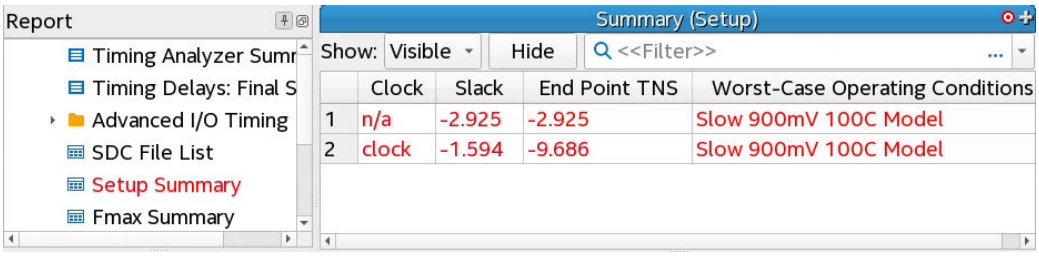
   ```
   set_global_assignment -name PROMOTE_WARNING_TO_ERROR <ID_Number>
   ```

2.2.4. Step 4: Analyze Timing Reports

During analysis, the Timing Analyzer examines the timing paths in the design, calculates the propagation delay along each path, checks for timing constraint violations, and reports timing results as positive slack or negative slack. Negative slack indicates a timing violation. Positive slack indicates that timing requirements are met.

The Timing Analyzer provides very fine-grained reporting and analysis capabilities to identify and correct violations along timing paths. Generate timing reports to view how to best optimize the critical paths in your design. If you modify, remove, or add constraints, re-run timing analysis. This iterative process helps resolve timing violations in your design.

Figure 47. Timing Analyzer Shows Failing Paths in Red



Reports that indicate failing timing performance appear in red text, and reports that pass appear in black text. A gold question mark icon indicates reports that are outdated due to SDC changes since generation. Regenerate these reports to show the latest data.

The following sections describe how to generate various timing reports for analysis.

Related Information

Timing Report Descriptions on page 126

2.2.4.1. Cross-Probing with Design Assistant

The Quartus Prime Design Assistant can automatically report any violations against a standard set of Intel FPGA-recommended design guidelines during stages of compilation. You can specify which rules you want the Design Assistant to check in your design, and customize the severity levels, thus eliminating or waiving rule checks that are not important for your design.

When you run Design Assistant during compilation, Design Assistant utilizes the in-flow (transient) data that generates during compilation to check for rule violations.

When you run Design Assistant in analysis mode from the Timing Analyzer, Design Assistant performs design rule checks using the static compilation snapshot data that you load.

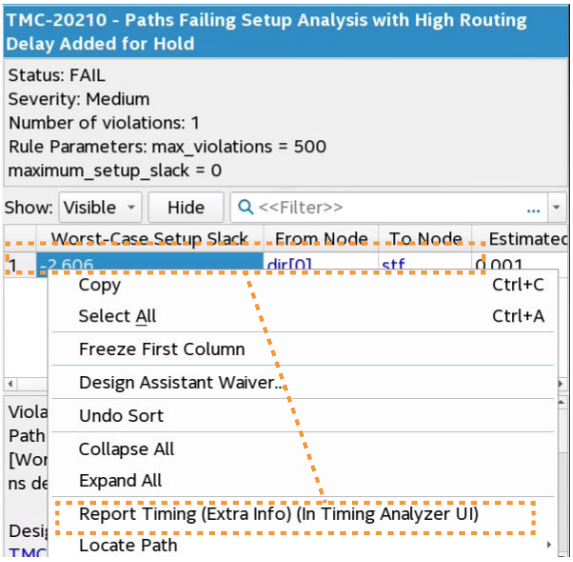
Some Design Assistant rule violations allow cross-probing into the related timing analysis data. Cross-probing can help you to more quickly identify the root cause and resolve any Design Assistant rule violations. For example, for a path with a setup analysis violation, you can cross-probe into the Timing Analyzer to identify the edge that has delay added for hold time.

Note: You must run the Compiler through at least the Plan stage before you can cross-probe to Timing Analyzer.

2.2.4.1.1. Cross-Probing from Design Assistant to Timing Analyzer

Some Design Assistant rule violations allow cross-probing into Timing Analyzer. For example, for a path that Design Assistant flags with a setup analysis violation due to delay added for hold, you can cross-probe into the Timing Analyzer to view more information on the affected path and edge.

Figure 48. Cross Probing from Design Assistant Rule TMC-20210 Violations to Timing Analyzer



Follow these steps to cross-probe from such Design Assistant rule violations to the Timing Analyzer:

1. Compile the design through at least the Compiler's Plan stage.
2. Locate a rule violation in the Design Assistant folder of the Compilation Report.
3. Right-click the rule violation to display any **Report Timing** commands available for the violation.
4. Click the **Report Timing** command. The Timing Analyzer opens and reports the timing data for the violation path. **Report Timing (Extra Info)** includes Estimated Delay Added for Hold and Route Stage Congestion Impact extra data.

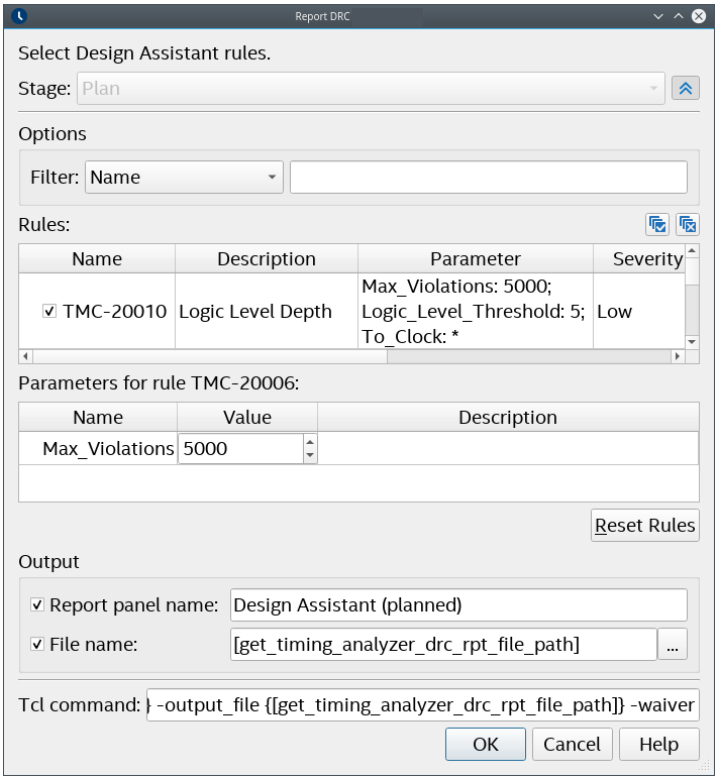
2.2.4.2. Launching Design Assistant from Timing Analyzer

You can run Design Assistant directly from the Timing Analyzer to assist when optimizing timing paths and other timing conditions. When you launch Design Assistant from the Timing Analyzer, Design Assistant only checks rules that relate to timing analysis.

Follow these steps to run the Design Assistant from the Timing Analyzer:

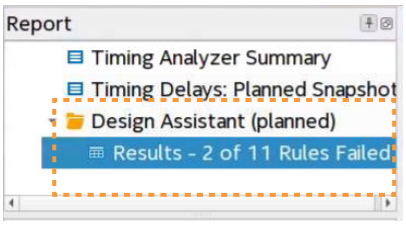
1. Compile the design through at least the Compiler's Plan stage.
2. Open the Timing Analyzer for the Compiler stage from the Compilation Dashboard.
3. In the Timing Analyzer, click **Reports ➤ Design Assistant ➤ Report DRC**. The **Report DRC** (design rule check) dialog box opens.
4. Under **Rules**, disable any rules that are not important to your analysis by removing the check mark.
5. Consider whether to adjust rule parameter values in the **Parameters** field.

Figure 49. Report DRC (Design Rule Check) Dialog Box



6. Confirm the **Report panel name** and optionally specify an output **File name**.
7. Click **Run**. The Results reports generate and appear in the **Report** pane, as well as the main Compilation Report.

Figure 50. Design Assistant Reports in Timing Analyzer Report Pane

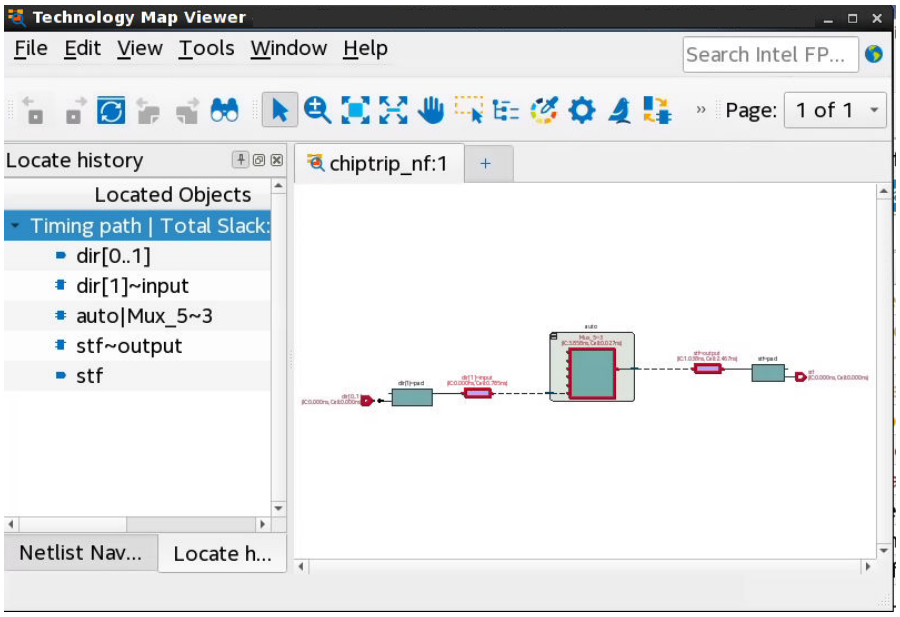


2.2.4.3. Locating Timing Paths in Other Tools

You can locate from paths and elements in the Timing Analyzer to other visualization tools in the Quartus Prime software, such as the Chip Planner, Technology Map Viewer, or Resource Property Viewer.

You can right-click most paths or node names in the Timing Analyzer reports and click the **Locate Node** or **Locate Path** commands. Use these commands in the Timing Analyzer GUI or the `locate` command in the Tcl console to locate to that node in other Quartus Prime tools.

Figure 51. Locate Path from Timing Analyzer to Technology Map Viewer



The following examples show how to locate the ten paths with the worst timing slack from Timing Analyzer to the Technology Map Viewer, and locate all ports matching data* in the Chip Planner.

Example 1. Locating from the Timing Analyzer

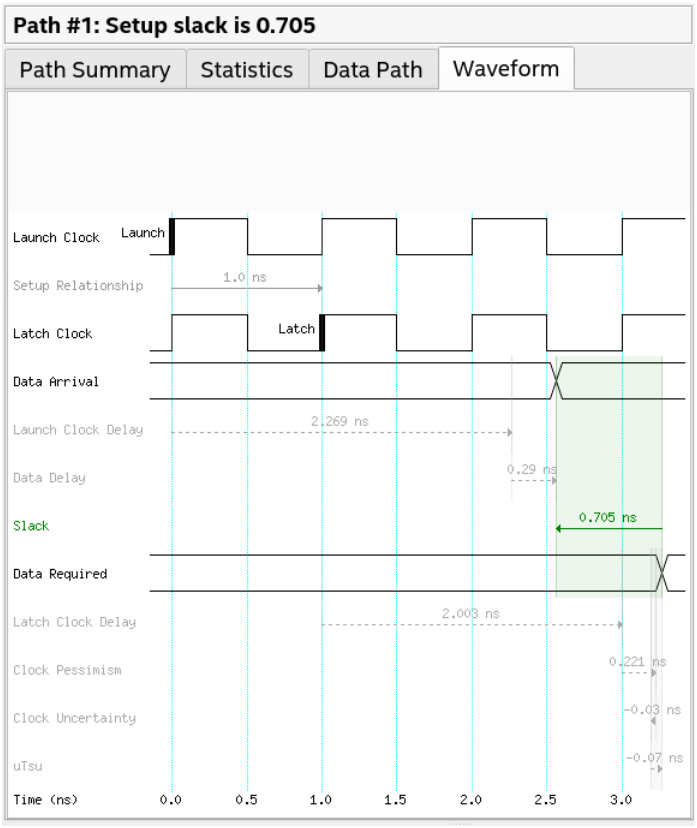
```
# Locate in the Technology Map Viewer the ten paths with the worst slack
locate [get_timing_paths -npaths 10] -tmv
# locate all ports that begin with data in the Chip Planner
locate [get_ports data*] -chip
```

2.2.4.4. Correlating Constraints to the Timing Report

Understanding how timing constraints and violations appear in the timing analysis reports is critical to understanding the results. The following examples show how specific constraints impact the timing analysis reports. Most timing constraints only affect the clock launch and latch edges. Specifically, `create_clock` and `create_generated_clock` create clocks with default relationships. However, the `set_multicycle_path` exception modifies the default setup and hold relationships. The `set_max_delay` and `set_min_delay` constraints are low-level overrides that explicitly indicate the maximum and minimum delays for the launch and latch edges.

The following figures show the results of running **Report Timing** on a particular path. You can view the incremental delays on the **Data Path** and **Waveform** tabs after running **Report Timing**. The **Waveform** tab allows you to visually reference the **Data Path** data, as well as the original `.sdc` constraints. You can use the **Waveform** tab to easily see how and where the constraints apply.

Figure 52. Report Timing (Waveform Tab)



In the following example, the design includes a clock driving the source and destination registers with a period of 10 ns. This results in a setup relationship of 10 ns (launch edge = 0 ns, latch edge = 10ns) and hold relationship of 0 ns (launch edge = 0 ns, latch edge = 0 ns) from the command:

```
create_clock -name clocktwo -period 10.000 [get_ports {clk2}]
```

Figure 53. Setup Relationship 10ns

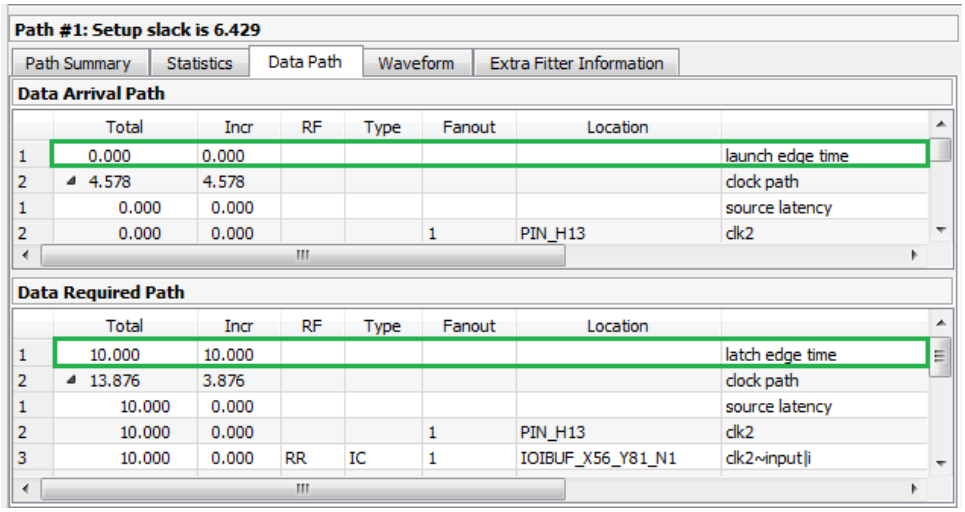
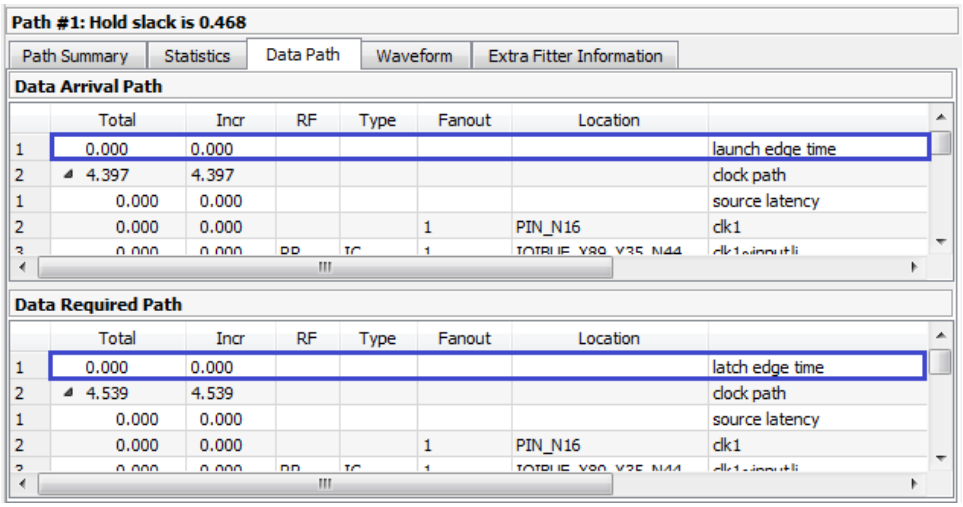


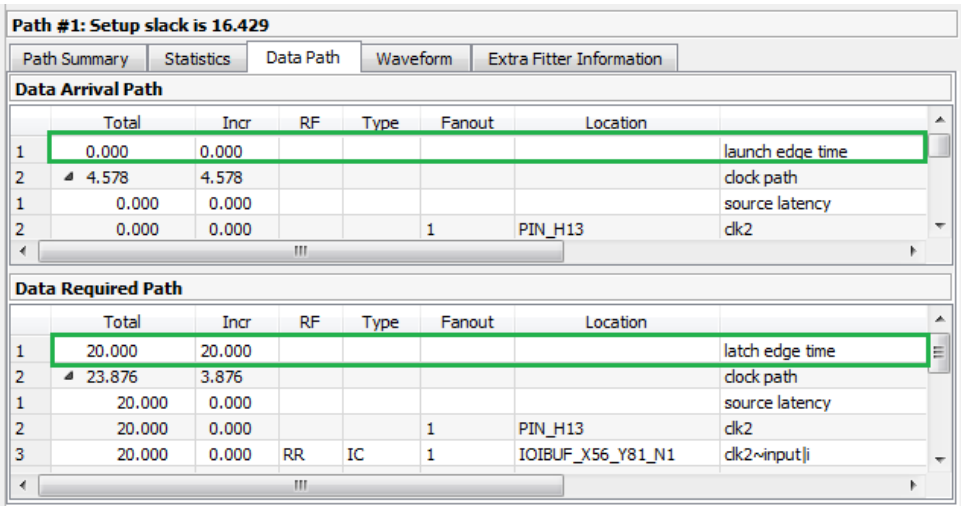
Figure 54. Hold Relationship 0ns



Adding `set_multicycle_path` constraints adds multicycles to relax the setup relationship, or open the window, making the setup relationship 20 ns while maintaining the hold relationship at 0 ns:

```
set_multicycle_path -from clocktwo -to clocktwo -setup -end 2
set_multicycle_path -from clocktwo -to clocktwo -hold -end 1
```

Figure 55. Setup Relationship 20ns

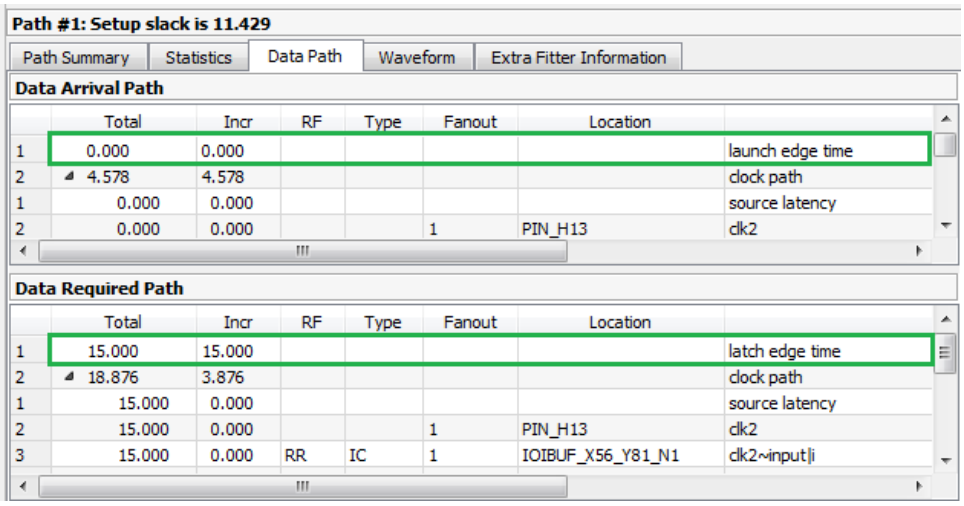


Adding the following `set_max_delay` constraints explicitly overrides the setup relationship:

```
set_max_delay -from [get_registers {regA}] -to \
     [get_registers {regB}] 15
```

Note that the only thing changing for these different constraints are the launch edge time and latch edge times for setup and hold analysis. Every other line item comes from delays inside the FPGA and are static for a given fit. View these reports to analyze how your constraints affect the timing reports.

Figure 56. Using set_max_delay

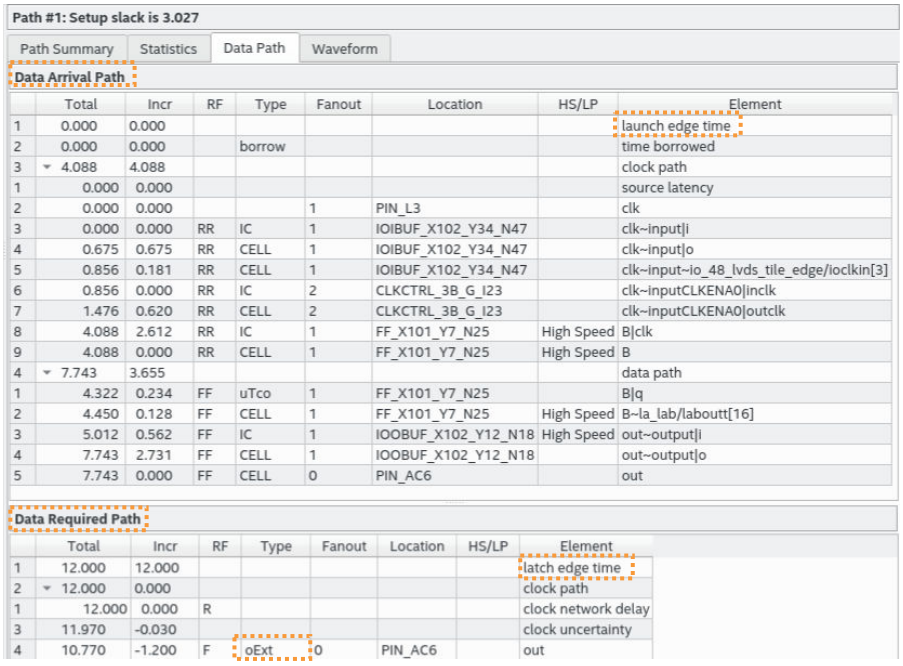


For I/O, you must add `set_input_delay` and `set_output_delay` constraints, as the following example shows. These constraints describe delays on signals from outside of the FPGA design that connect to the design's I/O ports.

```
create_clock -period 10 [get_ports clk]
# Clock used by the transfer, clock relationship is 10ns

# Setup constraints
set_output_delay -clock clk -max 1.2 [get_ports out]
# Subtracted from Data Required Path as oExt
set_max_delay -from [get_registers B] 12
# Sets latch edge time

# Hold constraints
set_output_delay -clock clk -min 2.3 [get_ports out]
# Subtracted from Data Required Path as oExt
set_min_delay -from [get_registers B] 8
# Sets latch edge time
```

The values of these constraints are the delays of the external signals between an external register and a port on the design. The `-clock` argument to the `set_input_delay` and `set_output_delay` specifies the clock domain that the external signal belongs to, or rather, the clock domain of the external register connected to the I/O port. The `-min` and `-max` options specify the worst-case or best-case delay; not specifying either option causes the worst- and best-case delays to be equal. I/O delays display as **iExt** or **oExt** in the **Type** column, as the following example reports shows.

Figure 57. Setup Slack Path Report and Waveforms for a Reg-To-Output Same-Clock Transfer



Path #1: Setup slack is 3.027

Path Summary | Statistics | Data Path | Waveform

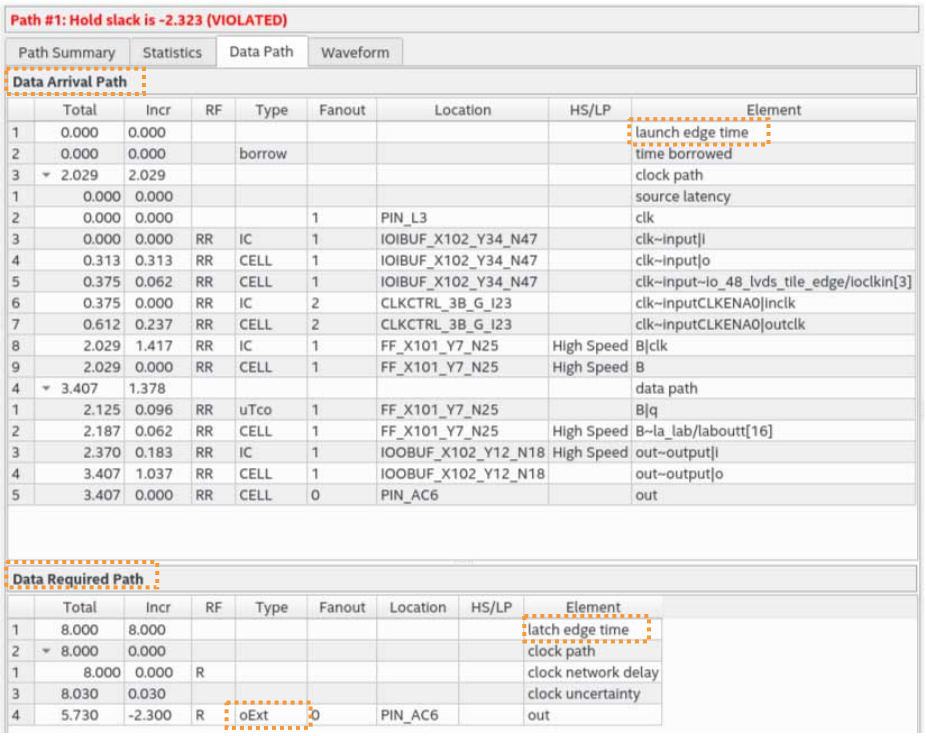
Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	HS/LP	Element
1	0.000	0.000						launch edge time
2	0.000	0.000		borrow				time borrowed
3	4.088	4.088						clock path
1	0.000	0.000						source latency
2	0.000	0.000			1	PIN_L3		clk
3	0.000	0.000	RR	IC	1	IOIBUF_X102_Y34_N47		clk~input|i
4	0.675	0.675	RR	CELL	1	IOIBUF_X102_Y34_N47		clk~input|o
5	0.856	0.181	RR	CELL	1	IOIBUF_X102_Y34_N47		clk~input~io_48_lvds_tile_edge/ioclkin[3]
6	0.856	0.000	RR	IC	2	CLKCTRL_3B_G_I23		clk~inputCLKENA0|inclk
7	1.476	0.620	RR	CELL	2	CLKCTRL_3B_G_I23		clk~inputCLKENA0|outclk
8	4.088	2.612	RR	IC	1	FF_X101_Y7_N25	High Speed	B|clk
9	4.088	0.000	RR	CELL	1	FF_X101_Y7_N25	High Speed	B
4	7.743	3.655						data path
1	4.322	0.234	FF	uTco	1	FF_X101_Y7_N25		B|q
2	4.450	0.128	FF	CELL	1	FF_X101_Y7_N25	High Speed	B~la_lab/laboutt[16]
3	5.012	0.562	FF	IC	1	IOOBUF_X102_Y12_N18	High Speed	out~output|i
4	7.743	2.731	FF	CELL	1	IOOBUF_X102_Y12_N18		out~output|o
5	7.743	0.000	FF	CELL	0	PIN_AC6		out

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	HS/LP	Element
1	12.000	12.000						latch edge time
2	12.000	0.000						clock path
1	12.000	0.000	R					clock network delay
3	11.970	-0.030						clock uncertainty
4	10.770	-1.200	F	oExt	0	PIN_AC6		out

Figure 58. Hold Slack Path Report and Waveforms for a Reg-To-Output Same-Clock Transfer



Path #1: Hold slack is -2.323 (VIOLATED)

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	HS/LP	Element
1	0.000	0.000						launch edge time
2	0.000	0.000		borrow				time borrowed
3	2.029	2.029						clock path
1	0.000	0.000						source latency
2	0.000	0.000			1	PIN_L3		clk
3	0.000	0.000	RR	IC	1	IOIBUF_X102_Y34_N47		clk~input|i
4	0.313	0.313	RR	CELL	1	IOIBUF_X102_Y34_N47		clk~input|o
5	0.375	0.062	RR	CELL	1	IOIBUF_X102_Y34_N47		clk~input~io_48_lvds_tile_edge/ioclkin[3]
6	0.375	0.000	RR	IC	2	CLKCTRL_3B_G_I23		clk~inputCLKENA0|inclk
7	0.612	0.237	RR	CELL	2	CLKCTRL_3B_G_I23		clk~inputCLKENA0|outclk
8	2.029	1.417	RR	IC	1	FF_X101_Y7_N25	High Speed	B|clk
9	2.029	0.000	RR	CELL	1	FF_X101_Y7_N25	High Speed	B
4	3.407	1.378						data path
1	2.125	0.096	RR	uTco	1	FF_X101_Y7_N25		B|q
2	2.187	0.062	RR	CELL	1	FF_X101_Y7_N25	High Speed	B~la_lab/laboutt[16]
3	2.370	0.183	RR	IC	1	IOOBUF_X102_Y12_N18	High Speed	out~output|i
4	3.407	1.037	RR	CELL	1	IOOBUF_X102_Y12_N18		out~output|o
5	3.407	0.000	RR	CELL	0	PIN_AC6		out

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	HS/LP	Element
1	8.000	8.000						latch edge time
2	8.000	0.000						clock path
1	8.000	0.000	R					clock network delay
3	8.030	0.030						clock uncertainty
4	5.730	-2.300	R	oExt	0	PIN_AC6		out

A clock relationship, which is the difference between the launching and latching clock edge of a transfer, is determined by the clock waveform, multicycle constraints, and minimum and maximum delay constraints. The Timing Analyzer also adds the value of `set_output_delay` as an **oExt** value. For outputs, this value is part of the **Data Required Path**, since this is the external part of the analysis. The setup report subtracts the `-max` value, making the setup relationship harder to meet, since the **Data Arrival Path** delay must be shorter than the **Data Required Path** delay. The Timing Analyzer also subtracts the `-min` value. This subtraction is why a negative number causes more restrictive hold timing. The **Data Arrival Path** delay must be longer than the **Data Required Path** delay.

Related Information

- Running Post-Synthesis Early Timing Analysis on page 34
- Running Post-Fit Timing Analysis on page 37
- Scripting Timing Analysis on page 164
- Relaxing Setup with Multicycle (set_multicyle_path) on page 104
- Creating Virtual Clocks on page 80
- I/O Constraints on page 93

2.3. Applying Timing Constraints

This section provides examples and describes how to correctly apply SDC timing constraints that guide design synthesis, Fitter placement, and produce accurate timing analysis results under various circumstances.
You can define a set of initial timing constraints, and then iteratively modify those constraints as the design progresses.

Early in the design cycle, you can use SDC-on-RTL constraints to target analysis of RTL nodes. This analysis provides a stable reference for constraints that can remain unchanged in subsequent compilation stages, such as clock definitions. Establishing a set of SDC-on-RTL constraints enables their propagation and application throughout the entire design cycle. Concurrently, you can create a conventional `.sdc` file for analysis of the remaining design elements, providing flexibility for iterative constraint adjustments as the design evolves.

This section also outlines the proper application of recommended conventional SDC timing constraints. Conventional SDC constraints guide Fitter placement via `.sdc` files, offering alternative approaches to achieve precise control over constraints throughout the design flow.

Related Information

- Using Timing Constraints throughout the Design Flow on page 27
- Applying Entity-Bound Timing Constraints on page 56
- Recommended Initial Conventional SDC Constraints on page 49

2.3.1. Recommended Initial Conventional SDC Constraints

Include the following basic SDC constraints in your conventional `.sdc` file. The following example shows application of the recommended conventional SDC constraints for a simple dual-clock design:

```
create_clock -period 20.00 -name adc_clk [get_ports adc_clk]
create_clock -period 8.00 -name sys_clk [get_ports sys_clk]

derive_pll_clocks

derive_clock_uncertainty
```

Note: Only Arria 10 and Cyclone® 10 GX devices support the **Derive PLL Clocks** (`derive_pll_clocks`) constraint. For all other supported devices, the Timing Analyzer automatically derives PLL clocks from constraints bound to the related IP.

Create Clock (create_clock) on page 50

Derive PLL Clocks (derive_pll_clocks) on page 50

Derive Clock Uncertainty (derive_clock_uncertainty) on page 51

Set Clock Groups (set_clock_groups) on page 52

2.3.1.1. Create Clock (create_clock)

The **Create Clock** (`create_clock`) constraint allows you to define the properties and requirements for a clock in the design. You must define clock constraints to determine the performance of your design and constrain the external clocks coming into the FPGA. You can enter the constraints in the Timing Analyzer GUI, or in the `.sdc` file directly.

You specify the **Clock name** (`-name`), clock **Period** (`-period`), rising and falling **Waveform edge** values (`-waveform`), and the target signal(s) to which the constraints apply.

The following command creates the `sys_clk` clock with an 8ns period, and applies the clock to the `fpga_clk` port.:

```
create_clock -name sys_clk -period 8.0 \
     [get_ports fpga_clk]
```

Note: Tcl and `.sdc` files are case-sensitive. Ensure that references to pins, ports, or nodes match the case of names in your design.

By default, the `sys_clk` example clock has a rising edge at time 0 ns, a 50% duty cycle, and a falling edge at time 4 ns. If you require a different duty cycle, or to represent an offset, specify the `-waveform` option.

Typically, you name a clock with the same name as the port you assign. In the example above, the following constraint accomplishes this:

```
create_clock -name fpga_clk -period 8.0 [get_ports fpga_clk]
```

There are now two unique objects called `fpga_clk`, a port in your design and a clock applied to that port.

Note: In Tcl syntax, square brackets execute the command inside them. `[get_ports fpga_clk]` executes a command that finds and returns a collection of all ports in the design that match `fpga_clk`.

Warning: Constraints that you define in the Timing Analyzer apply directly to the timing database, but do not automatically transfer to the `.sdc` file. Click **Write SDC File** on the Timing Analyzer **Tasks** pane to preserve constraints changes from the GUI in an `.sdc` file.

Related Information

Creating Base Clocks on page 79

2.3.1.2. Derive PLL Clocks (derive_pll_clocks)

The **Derive PLL Clocks** (`derive_pll_clocks`) constraint automatically creates clocks for each output of any PLL in your design.

Note: Only Arria 10 and Cyclone 10 GX devices support the **Derive PLL Clocks** (`derive_pll_clocks`) constraint. For all other supported devices, the Timing Analyzer automatically derives PLL clocks from constraints bound to the related IP.

The constraint can generate multiple clocks for each output clock pin if the PLL is using clock switchover: one clock for the `inclk[0]` input clock pin, and one clock for the `inclk[1]` input clock pin. Specify the **Create base clocks** (`create_base_clocks`) option to create base clocks on the inputs of the PLLs by default. By default the clock name is the same as the output clock pin name, or specify the **Use net name as clock name** (`use_net_name`) option to use the net name.

When you create PLLs, you must define the configuration of each PLL output. This definition allows the Timing Analyzer to automatically constrain the PLLs with the `derive_pll_clocks` command. This command also constrains transceiver clocks and adds multicycles between LVDS SERDES and user logic.

The `derive_pll_clocks` command prints an Info message to show each generated clock the command creates.

As an alternative to `derive_pll_clocks` you can copy-and-paste each `create_generated_clock` assignment into the `.sdc` file. However, if you subsequently modify the PLL setting, you must also change the generated clock constraint in the `.sdc` file. Examples of this type of change include modifying an existing output clock, adding a new PLL output, or making a change to the PLL's hierarchy. Use of `derive_pll_clocks` eliminates this requirement.

Related Information

- Creating Base Clocks on page 79
- Deriving PLL Clocks on page 85

2.3.1.3. Derive Clock Uncertainty (derive_clock_uncertainty)

The **Derive Clock Uncertainty** (`derive_clock_uncertainty`) constraint applies setup and hold clock uncertainty for clock-to-clock transfers in the design. This uncertainty represents characteristics like PLL jitter, clock tree jitter, and other factors of uncertainty.

You can enable the **Add clock uncertainty assignment** (`-add`) to add clock uncertainty values from any **Set Clock Uncertainty** (`set_clock_uncertainty`) constraint. You can **Overwrite existing clock uncertainty assignments** (`-overwrite`) any `set_clock_uncertainty` constraints.

```
create_clock -period 10.0 -name fpga_sys_clk [get_ports fpga_sys_clk] \
    derive_clock_uncertainty -add - overwrite
```

The Timing Analyzer generates an information message if you omit `derive_clock_uncertainty` from the `.sdc` file.

Related Information

Accounting for Clock Effect Characteristics on page 90

2.3.1.4. Set Clock Groups (set_clock_groups)

The **Set Clock Groups** (`set_clock_groups`) constraint allows you to specify which clocks in the design are unrelated. By default, the Timing Analyzer assumes that all clocks with a common base or parent clock are related, and that all transfers between those clock domains are valid for timing analysis. You can exclude transfers between specific clock domains from timing analysis by cutting clock groups.

Conversely, clocks without a common parent or base clock are always unrelated, but timing analysis includes the transfers between such clocks, unless those clocks are in different clock groups (or if all of their paths are cut with false path constraints).

You define the clock signals to include in each Group (-group), then specify the relationship between different groups, and then specify whether the groups are **Logically exclusive** (`-logically_exclusive`), **Physically exclusive** (`-physically_exclusive`, or **Asynchronous** (`-asynchronous`) from one another.

```
set_clock_groups -asynchronous -group {<clock1>...<clockn>} ... \
    -group {<clocka>...<clockn>}
```

- `-logically_exclusive`—defines clocks that are logically exclusive and not active at the same time, such as multiplexed clocks
- `-physically_exclusive`—defines clocks that that cannot be physically on the device at the same time.
- `-asynchronous`—defines completely unrelated clocks that have different ideal clock sources. This flag means the clocks are both switching, but not in a way that can synchronously pass data.

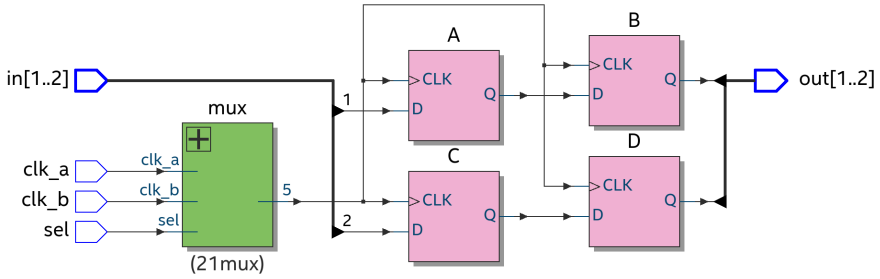
For example, if there are paths between an 8ns clock and 10ns clock, even if the clocks are completely asynchronous, the Timing Analyzer attempts to meet a 2ns setup relationship between these clocks, unless you specify that they are not related.

The following shows example constraints for a clock mux with two inputs, with multi-rate clocks, and the appropriate combinations of logical and physical exclusivity:

```
# First profile
create_clock -name clk_a1 -period 10 [get_ports clk_a]
create_clock -name clk_b1 -period 20 [get_ports clk_b]

# Second profile
create_clock -name clk_a2 -period 100 [get_ports clk_a] -add
create_clock -name clk_b2 -period 200 [get_ports clk_b] -add

# Mark base clocks as asynchronous to each other
set_clock_groups -asynchronous -group {clk_a?} -group {clk_b?}

# Define muxed clocks for each profile
set muxout [get_pins -compatibility_mode {mux*|combout}]
foreach profile {1 2} {
     set mux_clk_a "mux_clk_a${profile}"
     set mux_clk_b "mux_clk_b${profile}"

     create_generated_clock -name $mux_clk_a -source [get_ports clk_a] \
          -master_clock "clk_a${profile}" $muxout -add
     create_generated_clock -name $mux_clk_b -source [get_ports clk_b] \
          -master_clock "clk_b${profile}" $muxout -add

     # Mark each muxed clock as logically exclusive to each other
     set_clock_groups -logically_exclusive -group $mux_clk_a \
          -group $mux_clk_b
}
```

```
# Mark profile clocks as physically exclusive to each other
# (Do this after defining the derived clocks so they get cut too)
set_clock_groups -physically_exclusive -group {*clk_?1} \
     -group {*clk_?2}
```

Figure 59. Example Constraints Design Topology



Although the **Set Clock Groups** dialog box only permits two clock groups, you can specify an unlimited number of `-group {<group of clocks>}` options in the `.sdc` file. If you omit an unrelated clock from the assignment, the Timing Analyzer acts conservatively and analyzes that clock in context with all other domains to which the clock connects.

The Timing Analyzer does not currently analyze crosstalk explicitly. Instead, the timing models use extra guard bands to account for any potential crosstalk-induced delays. The Timing Analyzer treats the `-asynchronous` and `-exclusive` options the same for crosstalk-related analysis, but treats asynchronous and exclusive clock groups differently for things like max skew reporting and synchronizer detection.

A clock cannot be within multiple groups (`-group`) in a single assignment; however, you can have multiple `set_clock_groups` assignments.

Another way to cut timing between clocks is to use `set_false_path`. To cut timing between `sys_clk` and `dsp_clk`, you can use:

```
set_false_path -from [get_clocks sys_clk] -to [get_clocks dsp_clk]
```

```
set_false_path -from [get_clocks dsp_clk] -to [sys_clk]
```

This technique is effective if there are only a few clocks, but can become unmanageable with a large number of constraints. In a simple design with three PLLs that have multiple outputs, the `set_clock_groups` command can cut timing between clocks with less than ten lines, but the `set_false_path` command may require more than 50 lines.

Related Information

- Creating Generated Clocks (create_generated_clock) on page 82
- Relaxing Setup with Multicycle (set_multicyle_path) on page 104
- Accounting for a Phase Shift (-phase) on page 105

2.3.2. Example Circuit and Conventional SDC File

The following `.sdc` file demonstrates constraining a dual-clock, phase-locked loop (PLL) example that illustrates. and other common synchronous design elements.

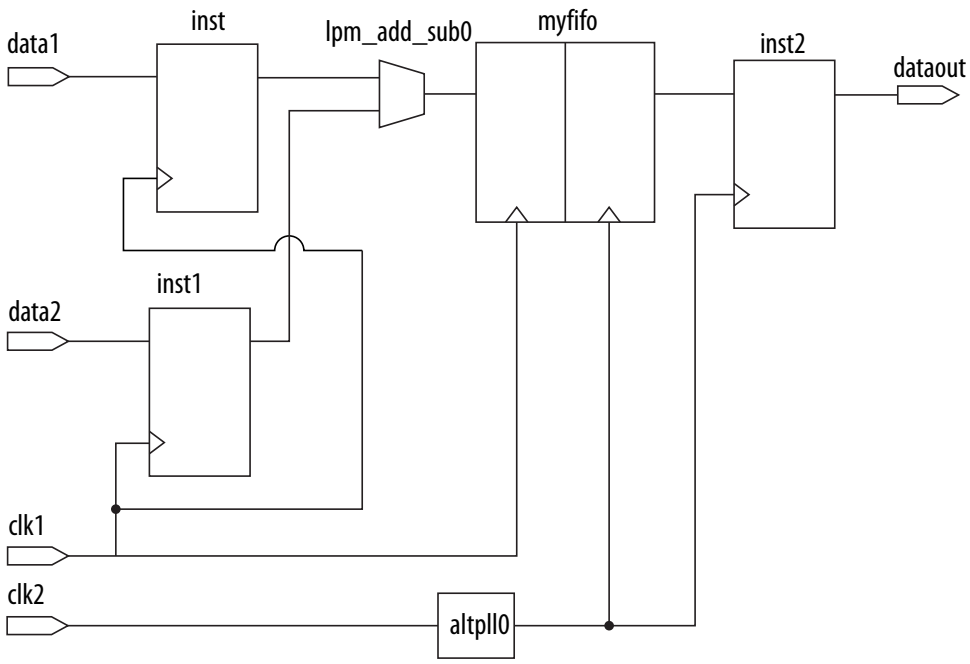
Example 2. Conventional .sdc Constraints Example

```
# Create clock constraints
create_clock -name clockone -period 10.000Ns [get_ports {clk1}]
create_clock -name clocktwo -period 10.000Ns [get_ports {clk2}]
# Create virtual clocks for input and output delay constraints
create clock -name clockone_ext -period 10.000Ns
create clock -name clocktwo_ext -period 10.000Ns
# derive PLL clocks to create the altpll0| clock referenced later
derive_pll_clocks
# derive clock uncertainty
derive_clock_uncertainty
# Specify that clockone and clocktwo are unrelated by assigning
# them to separate asynchronous groups
set_clock_groups \
  -asynchronous \
  -group {clockone} \
  -group {clocktwo altpll0|altpll_component|auto_generated|pll1|clk[0]}
# set input and output delays
set_input_delay -clock { clockone_ext } -max 4 [get_ports {data1}]
set_input_delay -clock { clockone_ext } -min -1 [get_ports {data1}]
set_input_delay -clock { clockone_ext } -max 4 [get_ports {data2}]
set_input_delay -clock { clockone_ext } -min -1 [get_ports {data2}]
set_output_delay -clock { clocktwo_ext } -max 6 [get_ports {dataout}]
set_output_delay -clock { clocktwo_ext } -min -3 [get_ports {dataout}]
```

The conventional `.sdc` file contains the following constraints that you typically include for most designs:

- Definitions of `clockone` and `clocktwo` as base clocks, and assignment of those constraints to nodes in the design.
- Definitions of `clockone_ext` and `clocktwo_ext` as virtual clocks, which represent clocks driving external devices interfacing with the FPGA.
- Automated derivation of generated clocks on PLL outputs.
- Derivation of clock uncertainty.
- Specification of two clock groups, the first containing `clockone` and its related clocks, the second containing `clocktwo` and the output of the PLL. This specification overrides the default analysis of all clocks in the design as related to each other.
- Specification of input and output delays for the design.

Figure 60. Dual-Clock Design Constraint Example



Related Information

Asynchronous Clock Groups (-asynchronous) on page 89

2.3.3. SDC File Precedence

To ensure proper integration into the compilation flow, you must add any SDC-on-RTL and conventional SDC files to your project, as Step 1: Specifying General Timing Analyzer Settings describes. Alternatively, you can add files to your project by modifying the assignments in the project `.qsf` file directly.

The Compiler processes conventional SDC files in the order listed in the `.qsf`. You can add, remove, or change the processing order of `.sdc` files using **Assignments ➤ Settings ➤ Timing Analyzer**, or by modifying the `.qsf` directly.

Note: SDC-on-RTL files take precedence and the Compiler always processes `.rtlsdc` files before conventional `.sdc` files that target the timing netlist, regardless of order in **Assignments ➤ Settings ➤ Timing Analyzer**.

When using the `read_sdc` command at the command line without any arguments, the Compiler reads constraints in the following sequence:

1. Initially, the Compiler reads any SDC-on-RTL constraints.
2. Next, the Compiler reads any synthesis-only constraints that apply to only the synthesis stage.
3. Next, the Compiler reads any conventional SDC constraints. For conventional SDC constraints, the following order applies:

a. First, the Compiler processes constraints embedded in HDL files.
b. Finally, the Compiler processes `.sdc` files based on file order.

2.3.4. Iteratively Modifying Constraints

You initially establish SDC-on-RTL constraints during the Analysis & Elaboration stage of the compilation flow. Making iterative changes to SDC-on-RTL constraints may require you to rerun Analysis & Elaboration multiple times to apply revised constraints to the netlist.

It is best to designate constraints that remain constant across compilation stages as SDC-on-RTL constraints. Subsequently, you can iteratively modify and reanalyze the constraints in the rest of your design using conventional constraint files.

To iteratively modify constraints, follow these steps

1. Click **Tools ➤ Timing Analyzer**.
2. Generate the reports you want to analyze. Double-click **Report All Summaries** under **Macros** to generate setup, hold, recovery, and removal summaries, summaries for supported reports, and a list of all the defined clocks in the design. These summaries cover all paths you constrain in your design. Whenever modifying or correcting constraints, generate the **Constraint Diagnostic** reports to identify unconstrained parts of your design, or ignored constraints.
3. Analyze the results in the reports. When done modifying constraints, rerun the reports to find any unexpected results. For example, a cross-domain path might indicate that you forgot to cut a transfer by including a clock in a clock group.
4. Create or edit the appropriate constraints in your `.sdc` file and save the file.
5. Double-click **Reset Design** in the **Tasks** pane. This removes all constraints from your design. Removing all constraints from your design allows rereading the SDC files, including your changes.
6. Regenerate the reports you want to analyze.
7. Reanalyze the results.
8. Repeat steps 4-7 as necessary.

Using this approach, timing analysis runs with updated constraints, preserving the existing logic placement. The Fitter relies on the original constraints for design place-and-route, while the Timing Analyzer incorporates the newly applied constraints. If any timing issues arise in relation to the updated constraints, rerun the Fitter stage of compilation. Furthermore, for enhanced control over your design, consider converting select refined constraints to the SDC-on-RTL approach, as Specifying SDC-on-RTL Timing Constraints describes.

Related Information

Relaxing Setup with Multicycle (set_multicyle_path) on page 104

2.3.5. Applying Entity-Bound Timing Constraints

Entity-bound timing constraints enable meticulous control of timing constraints by allowing you to confine (bind) a constraint set to a specific design entity or a group of entities. You can define entity-based SDC-on-RTL constraints that enable early timing analysis after running only Analysis & Elaboration. You can similarly use conventional SDCs for post-fit timing analysis.

Related Information

- Using Entity-Based SDC-on-RTL Constraints on page 57
- Using Entity-Bound SDC Files on page 64

2.3.5.1. Using Entity-Based SDC-on-RTL Constraints

A typical design includes a combination of third-party IPs and actively evolving RTL components. You can use entity-based SDC-on-RTL constraints to define precise timing constraints at module boundaries, helping to ensure the seamless integration of IP and constraints. Entity-based SDC-on-RTL constraints allow IP authors to encapsulate the SDC constraints for their IP.

Conventional SDC timing constraints generally apply globally throughout a design, rather than to specific entities. However, proper encapsulation of IP SDCs allows you to use the IP without encountering unexpected SDC leaks. Entity binding prefixes filters with the full path name of each IP, effectively limiting the scope of the SDC constraints. This entity binding effectively prevents any SDC leaks and any potential impact on design paths with a matching name.

IP authors can optimize these constraints for post-synthesis Early Timing Analysis within the context IP instantiation in the design hierarchy. Even if an IP author does not know where the IPs are instantiated yet, the constraints remain effective. This approach allows IP authors to implement SDC constraints without requiring detailed information about the eventual placement of the IP within the design hierarchy.

The Compiler reads entity-based SDC-on-RTL constraints in designs and IP cores during Analysis & Elaboration. The Compiler preserves the constraints in a low-level entity database. The Compiler processes these constraints in the SDC read-in order and applies the constraints to the hierarchical netlist objects during compilation.

QSF Assignment Syntax

```
set_instance_assignment -name RTL_SDC_FILE <sdc_file_name> \
   -entity <entity_name> [-no_sdc_promotion]
```

Where:

Argument	Description
`RTL_SDC_FILE`	Specifies the SDC-on-RTL file name.
`-entity`	Specifies an entity-based assignment. The SDC file applies to each instance of the design entity. The instance hierarchy path implicitly applies to the `pattern` argument search for `dni::get_*` (`get_cells`, `get_pins`, `get_ports`, and `get_nets`) commands.
`[-no_sdc_promotion]`	An optional argument that requires the `-entity` flag. For entity-based constraints, the `[-no_sdc_promotion]` argument removes the default behavior of the instance hierarchy path being the default implicit with the netlist search commands. This argument specifies which individual command to apply the instance hierarchy path in the netlist object search. Use the `get_entity_current_instance` Tcl command to obtain the current instance hierarchy path of the entity. For example: `set_false_path -from [get_pins [get_entity_current_instance]|ff_src|clk] \` `-to [get_pins [get_entity_current_instance]|ff_dst|d]]`

2.3.5.1.1. Targeting Constraints to Module Inputs and Outputs

SDC-on-RTL allows you to define constraints at module boundaries, even if some internal connections within the modules or IP remain partially unknown. It is best to apply SDC-on-RTL constraints at the module boundaries, specifically at the input and output boundaries of each module.

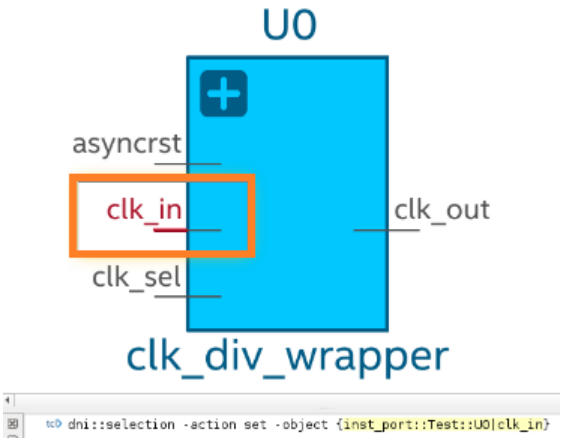
When targeting your timing constraints to the inputs and outputs of a module, you can target the following different element types, depending on your circumstances:

inst_port—these elements are retrieved in collections due to applying the `get_pins` filter. They target inputs and outputs of modules in a manner similar to addressing pins on registers and LUTs.

```
# inside
get_pins {clk_in} clk_dic.rtlsdc
```

Note: Use `get_pins` for constraints that expect pins as targets.

Figure 61. Targeting the Instance Port

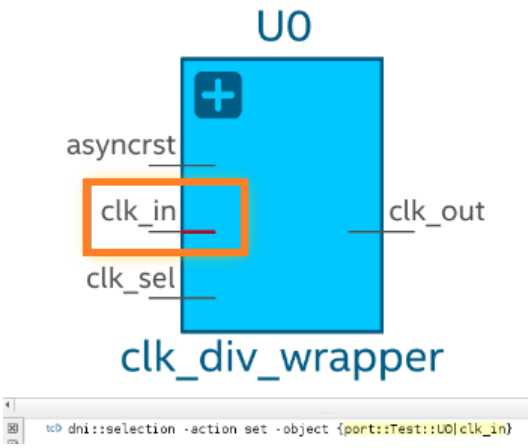


port— these elements reside within the module and are primarily for use in targeting ports in entity-bound constraints. You can employ the `get_ports` filter for this purpose.

```
# Inside
get_ports {clk_in} clk_dic.rtlsdc
```

Note: Use `get_ports` for constraints that expect ports as targets.

Figure 62. Targeting the Port



2.3.5.1.2. Entity Based SDC-on-RTL Constraint Scope

The entity-based SDC-on-RTL approach offers diverse scoping possibilities for determining the constraint's scope of influence.

Table 7. Entity-based Constraint Scope

Constraint Scope Type	Features	To Enable Instance-based Scoping
Automatic	• Accessible only by `.qsf` assignment. • Under automatic scoping, constraints apply to every instance of the assigned entity across the project. • Each result from any `get` command (for example, `get_pins`, `get_ports`, and so on) in the SDC file is prepended with the instance's path. • All `get` commands are confined to the target elements within the bound instance associated with the SDC-on-RTL file.	Use the following arguments: `name RTL_SDC_FILE <sdc_on_rtl_file_name>` `-entity <entity_name>` `-library <library_name>`
Manual	• Accessible only by `.qsf` assignment. • The `-no_sdc_promotion` setting disables automatic scoping, necessitating full hierarchical path for targeting nodes. • Allows targeting nodes beyond entity boundaries. • The `get_entity_current_instance` command delivers the top-level path to the current instance, allowing you to merge filters targeting elements in the current instance with commands addressing those beyond entity boundaries.	Use the following arguments: `-name RTL_SDC_FILE <sdc_on_rtl_file_name>` `-entity <entity_name>` `-library <library_name>` `-no_sdc_promotion` Prepend each collection filter with `get_entity_current_instance` to target nodes within the entity boundaries. For example: `get_pins [get_entity_current_instance]|reg[*]|q`

continued...

Constraint Scope Type	Features	To Enable Instance-based Scoping
Disabled	• Accessible by QSF assignment only. • `-no_sdc_promotion` and the `-no_auto_inst_discovery` arguments together disable scoping, treating an entity-bound SDC-on-RTL as a global scope. The SDC file is read only once for the entire compilation, rather processing for each instance linked. • This mode is ideal when bundling the SDC-on-RTL with an entity destined for export as a `.qdb` file, while preserving the ability to specify global, top-level paths in their `get` commands.	Use the following arguments: `-name RTL_SDC_FILE <sdc_on_rtl_file_name>` `-entity <entity_name>` `-library <library_name>` `-no_sdc_promotion` `-no_auto_inst_discovery`

When you define entity-bound SDC files, the software applies the constraints using automatic scoping, unless the `-no_sdc_promotion` or `-no_auto_inst_discovery` arguments are present.

Automatic scoping involves prepending filters with the instance's path. To provide clarity, the following table illustrates how paths are interpreted in various Tcl commands due to the automatic scoping of constraints:

Table 8. Automatic Scope of Constraints

Constraint Example	Auto-Scope Constraint Interpretation for Instance X|Y
`set_false_path -from [get_pins reg_a|clk]`	`set_false_path -from [get_pins X|Y|reg_a|clk]`
`set_false_path -from [get_pins reg_a|clk] -to [get_pins reg_b|d]`	`set_false_path -from [get_pins X|Y|reg_a|clk] -to [get_pins X|Y|reg_b|d]`
`set_false_path –from [get_clocks clk_1] –to [get_clocks clk_2]`	`set_false_path –from [get_clocks clk_1] –to [get_clocks clk_2]`
`set_max_delay –from [get_ports in] -to [get_pins reg_a|d] 2.0`	`set_max_delay –from [get_ports in] -to [get_pins X|Y|reg_a|d] 2.0`
`get_ports *`	`get_ports *`
`get_clocks *`	`get_clocks *`
`get_ports a`	`get_ports a`
`get_clocks a`	`get_clocks a`

When automatic scoping is disabled through QSF assignments, including the use of the `-no_sdc_promotion` argument, you must manually prepend the top-level path to achieve the same behavior as automatic scoping. To simplify this process, use the `-get_entity_current_instance` command that returns the top-level path of the

current instance. The following table illustrates how paths are interpreted when you use the `-get_entity_current_instance` command to add the top-level path to certain Tcl commands:

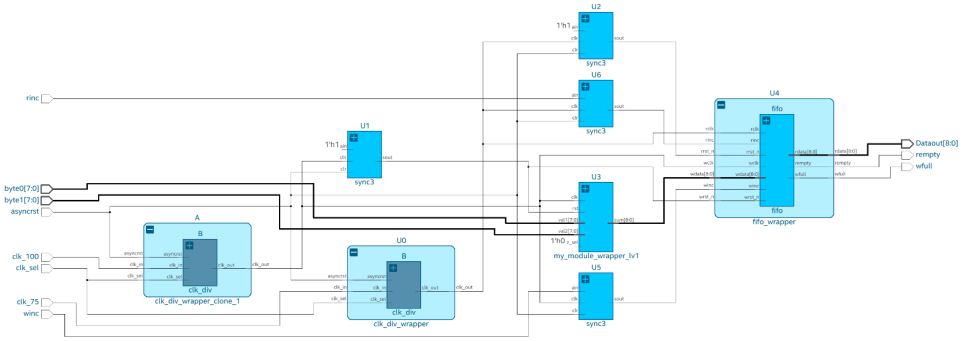
Table 9. Manual Scope of Constraints

Constraint Example	Manual Scope Constraint Interpretation
`set_false_path –from [get_entity_current_instance |reg_a|clk –to [get_entity_current_instance]|reg_b|d`	`set_false_path –from i1|inner|reg_a|clk –to i1|inner|reg_b|d` `set_false_path –from i2|inner|reg_a|clk –to i2|inner|reg_b|d` `set_false_path –from i3|reg_a|clk –to i3|reg_b|d`
`create_generated_clock –divide_by 2 –source \` `  [get_ports inclk] –name \` `  [get_entity_current_instance]_divclk \` `  [get_entity_current_instance]|div` `set_multicycle_path –from \` `  [get_entity_current_instance]|a \` `  –to [get_entity_current_instance]|b 2`	`# Evaluated for instances i1 and i2` `create_generated_clock –divide_by 2 –source \` `  [get_ports inclk] –name i1_divclk i1|div` `set_multicycle_path –from i1|a –to i1|b 2 \` `create_generated_clock –divide_by 2 –source \` `  [get_ports inclk] –name i2_divclk \` `  i2|div set_multicycle_path –from i2|a \` `  –to i2|b 2 \`

2.3.5.1.3. Automatic Scope Example for SDC-on-RTL

This example illustrates how to employ entity-based SDC-on-RTL constraints with automatic scope in your design. The following example uses two instances of `clk_div` and an additional `fifo` instance to illustrate how to apply this automatic scope approach:

Figure 63. Entity-Based SDC-on-RTL Design Example



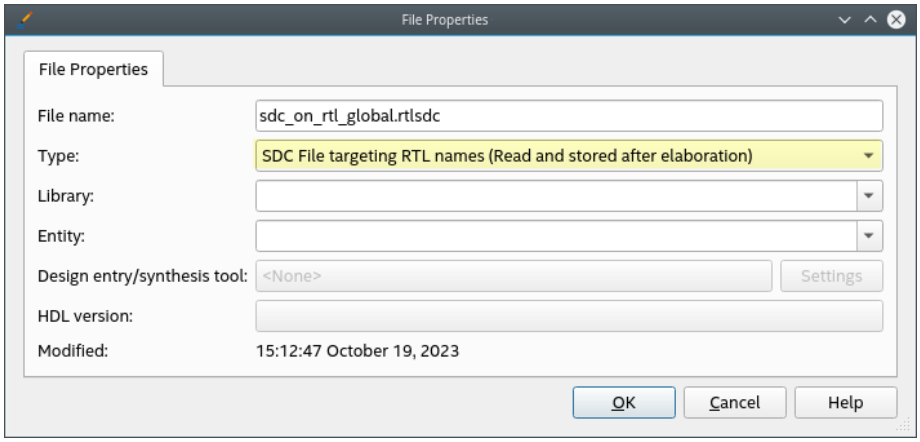
1. Apply global SDC-on-RTL constraints to the design:

```
# sdc_on_rtl_global.rtlsdc

create_clock -period 100MHz [get_ports clk_100]
create_clock -period 75MHz [get_ports clk_75]
```

2. Use the **File Properties** dialog to assign this `.rtlsdc` file as the **SDC File Targeting RTL names** or use the following `.qsf` assignment:

```
set_global_assignment -name RTL_SDC_FILE <filename>
```

Figure 64. File Properties Dialog

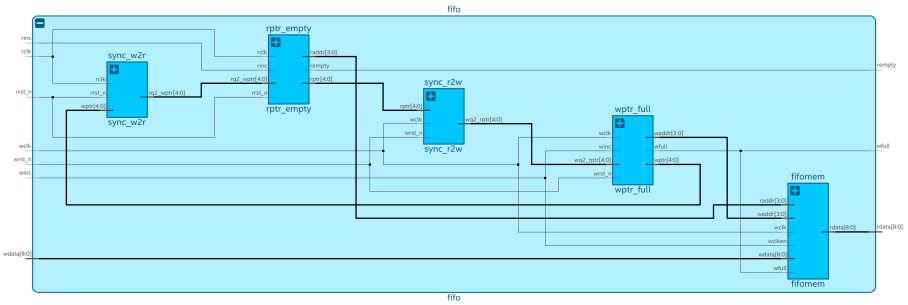


3. Define the following contents of the first entity based `.rtlsdc` file that is confined to the `clk_div` modules. In this case, the inner content of the module is unknown, but you can describe the known multiplexed clock behavior of the output in the `.rtlsdc`:

```
# clk_dic.rtlsdc
set current_instance [get_entity_current_instance];

create_generated_clock -name ${current_instance}_clk_mux_2 -source
[get_ports clk_in] -divide_by 2 [get_ports clk_out]
create_generated_clock -name ${current_instance}_clk_mux_1 -source
[get_ports clk_in] [get_ports clk_out] -add
```

In contrast, you can still define constraints using the following entity-bound approach for the `fifo` module that for which inner logic is already known. This approach can be advantageous, especially when multiple instances of the same module share identical constraints.

Figure 65. FIFO Module




```
set_false_path -from [get_pins wptr_full|wptr[*]|clk] -to [get_pins sync_w2r|
rq1_wptr[*]|d]
set_false_path -from [get_pins rptr_empty|rbin[4]|clk] -to [get_pins
sync_r2w|wq1_rptr[4]|d]
set_false_path -from [get_pins rptr_empty|rptr[*]|clk] -to [get_pins
```

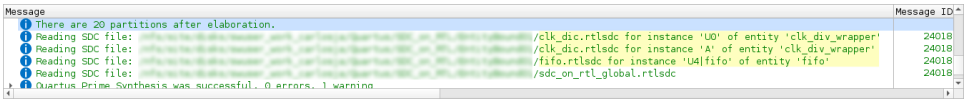
```
sync_r2w|wq1_rptr[*]|d]
set_false_path -from [get_pins wptr_full|wbin[4]|clk] -to [get_pins sync_w2r|
rq1_wptr[4]|d]
```

The following `.qsf` assignments add the `clk_div.rtlsdc` and `fifo.rtlsdc` files to the project, and define the file behavior as entity-based SDC-on-RTL:

```
set_global_assignment -name RTL_SDC_FILE clk_dic.rtlsdc -entity clk_div_wrapper -
library clk_div_wrapper
set_global_assignment -name RTL_SDC_FILE fifo.rtlsdc -entity fifo -library fifo
```

This assignment reduces the scope of each `.rtlsdc` file to the entities that match the assigned name. During Analysis & Elaboration, messages confirm that the Compiler appropriately applies each `.rtlsdc` file according to its assigned module.

Figure 66. Analysis & Elaboration Messages Confirm .rtlsdc Files Applied



You can further validate the correct application of each `.rtlsdc` file in the SDC File List report. The SDC File List report contains a comprehensive list, delineating each SDC file read, the file's assigned instance, and the file uses the SDC-on-RTL approach.

Figure 67. SDC File List Report

SDC File List

Show: Visible | Hide | <<Filter>> (use !<string> to invert filter)

	SDC File Path	Instance	Entity	Library	Promoted	Status	SDC on RTL	Processing Time
1	sdc_on_rtl_global.rtlsdc				No	OK	Yes	00:00:00
2	clk_dic.rtlsdc	U0	clk_div_wrapper	clk_div_wrapper	Yes	OK	Yes	00:00:00
3	clk_dic.rtlsdc	A	clk_div_wrapper	clk_div_wrapper	Yes	OK	Yes	00:00:00
4	fifo.rtlsdc	U4|fifo	fifo	fifo	Yes	OK	Yes	00:00:00

You can further confirm the correct application of constraints according to the constraint target and purpose in the Create Generated Clocks and Set False Paths reports.

Figure 68. Create Generated Clock Report

Create Generated Clock

Show: Visible | Hide | <<Filter>> (use !<string> to invert filter)

	SDC Command	Name	Source	Divide By	Master Clock	Targets	Add Clock	Location
1	create_generated_clock	U0_clk_mux_2	[get_pins {clk_75}]	2	clk_75	[get_pins {U0|B|clk_out|input}]	-add	clk_dic.rtlsdc:3
2	create_generated_clock	U0_clk_mux_1	[get_pins {clk_75}]		clk_75	[get_pins {U0|B|clk_out|input}]	-add	clk_dic.rtlsdc:4
3	create_generated_clock	A_clk_mux_2	[get_pins {clk_100}]	2	clk_100	[get_pins {A|B|clk_out|input}]	-add	clk_dic.rtlsdc:3
4	create_generated_clock	A_clk_mux_1	[get_pins {clk_100}]		clk_100	[get_pins {A|B|clk_out|input}]	-add	clk_dic.rtlsdc:4

Figure 69. Set False Path Report

Set False Path

Show: Visible | Hide | <<Filter>> (use !<string> to invert filter)

	SDC Command	From	To	Location
1	set_false_path	[get_pins {U4|fifo|wptr_full|wptr...k U4|fifo|wptr_full|wptr[2]|clk}]	[get_pins {U4|fifo|sync_w2r|rq1...4|fifo|sync_w2r|rq1_wptr[2]|d}]	fifo.rtlsdc:1
2	set_false_path	[get_pins {U4|fifo|rptr_empty|rbin[4]|clk}]	[get_pins {U4|fifo|sync_r2w|wq1_rptr[4]|d}]	fifo.rtlsdc:2
3	set_false_path	[get_pins {U4|fifo|rptr_empty|rpt... U4|fifo|rptr_empty|rptr[2]|clk}]	[get_pins {U4|fifo|sync_r2w|wq1...4|fifo|sync_r2w|wq1_rptr[2]|d}]	fifo.rtlsdc:3
4	set_false_path	[get_pins {U4|fifo|wptr_full|wbin[4]|clk}]	[get_pins {U4|fifo|sync_w2r|rq1_wptr[4]|d}]	fifo.rtlsdc:4
5	set_false_path	[get_pins {clk_sel|input clk_sel}]		sdc_on_rtl...l.rtlsdc:8
6	set_false_path	[get_ports {asyncrst}]		sdc_on_rtl...l.rtlsdc:9
7	set_false_path	[get_ports {rinc}]		sdc_on_rt...rtlsdc:10
8	set_false_path	[get_ports {winc}]		sdc_on_rt...rtlsdc:11

2.3.5.1.4. Manual Scope Example for SDC-on-RTL

You can change the scope of entity-based SDC-on-RTL files to manual by including the `-no_sdc_promotion` parameter in the `RTL_SDC_FILE` file definition. This parameter prevents the Compiler from prepending each collection filter with the full path of the current instance.

For example, to change to manual the scoping of the `clk_dic.rtlsdc` file associated with the `clk_div_wrapper` entity , add the `-no_sdc_promotion` parameter as follows:

```
set_global_assignment -name RTL_SDC_FILE clk_dic.rtlsdc -entity clk_div_wrapper \
-library clk_div_wrapper -no_sdc_promotion
```

To scope your collection filters precisely, use the `get_entity_current_instance` command. Specifying the top-level path to the present instance streamlines the process of scoping filters directed towards elements within instance boundaries.

```
set current_instance [get_entity_current_instance]
create_generated_clock -name ${current_instance}_clk_mux_2 -source \
  [get_ports $current_instance|clk_in] -divide_by 2 [get_ports $current_instance|
clk_out]

create_generated_clock -name ${current_instance}_clk_mux_1 -source \
[get_ports $current_instance|clk_in] [get_ports $current_instance|clk_out] -add
```

When the automatic scope is disabled for a designated entity-based SDC-on-RTL file, the SDC File List report indicates this change by displaying "No" in the promoted column.

Figure 70. SDC File List Report

SDC File List

Show: Visible | Hide | <<Filter>> (use !<string> to invert filter)

	SDC File Path	Instance	Entity	Library	Promoted	Status	SDC on RTL	Processing Time
1	sdc_on_rtl_global.rtlsdc				No	OK	Yes	00:00:00
2	clk_dic.rtlsdc	U0	clk_div_wrapper	clk_div_wrapper	No	OK	Yes	00:00:00
3	clk_dic.rtlsdc	A	clk_div_wrapper	clk_div_wrapper	No	OK	Yes	00:00:00
4	fifo.rtlsdc	U4|fifo	fifo	fifo	Yes	OK	Yes	00:00:00

2.3.5.2. Using Entity-Bound SDC Files

Throughout the design flow, most timing constraints specified in a Synopsis Design Constraints (SDC) file have a global scope across your project. However, if you want to associate a distinct set of constraints with a specific design entity, you can use the `SDC_ENTITY_FILE` assignment to assign SDC files to particular entity modules within your project.

Entity-bound SDC files significantly enhance the precision of timing constraints by allowing you to target constraints exclusively to specific entities where they are required. This approach bypasses the inadvertent ramifications of global constraints, which could encompass more targets than intended. As a result, you gain greater control over the precise locations within your design where these constraints take effect.

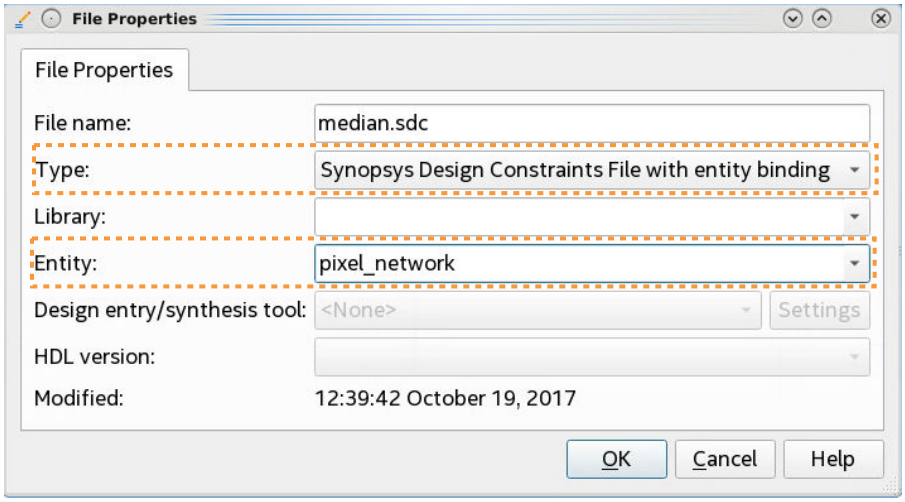
In addition to enhanced constraint precision, entity-bound SDC files also provide portability of the constraints. When you export a partition that contains entity-bound SDC constraints, you can optionally include these constraints in the exported file using

the **Include entity-bound SDC files for the selected partition** option in the **Export Design Partition** dialog box. Alternatively, you can specify this with the-`include_sdc_entity_in_partition` argument via a Tcl command.

To associate a specific SDC file with an entity using the entity-bound SDC file approach within the Quartus Prime Pro Edition software, follow these steps:

1. Generate a new SDC file and include it in your project by clicking **Project ➤ Add/ Remove files in project**.
2. Navigate to the files list and select the newly created SDC file.
3. Click the **Properties** button.
4. In the **Type** drop-down list, select **Synopsys Design Constraints File with entity binding**.

Figure 71. Entity Binding



5. In the **Entity** drop-down list, identify the entity you intend to bind to the SDC file.
6. Click **OK** to save the changes.

Alternatively, you can define the association between a specific SDC file and an entity by using the following assignment in the `.qsf` file:

QSF Assignment Syntax:

```
set_global_assignment -entity <entity_name> -name SDC_ENTITY_FILE
<sdc_file_name> \
-library <library_name> [-no_sdc_promotion] [-no_auto_inst_discovery]
```

Where:

Argument	Description
`-entity <entity_name>`	Mandatory argument. It defines the entity you want to bind to the SDC file.
`-name SDC_ENTITY_FILE <sdc_file_name>`	Specifies the SDC file name. The file's name is relative to the project path. In terms of scoping, the default setup for `SDC_ENTITY_FILE` is automatic constraint scoping. Automatic scoping means that each result from any `get` command (for example, `get_pins`, `get_ports`, and so on) in the SDC file is prepend with the instance's path. This confines all `get` commands to target items solely within the bound

continued...

Argument	Description
	instance associated with the SDC file. You can deactivate this configuration (which facilitates instance-level targeting) using the `-no_sdc_promotion` argument available through SDC commands. With this choice, you can handle the responsibility of manually scoping your collections either by explicitly providing the top-level path to the current instance or using the `get_current_instance` command that delivers the top-level path to the current instance. Such an approach proves invaluable when combining local commands with those that necessitate targeting global, top-level paths or objects outside the instance associated with the SDC file.
`-library <library_name>`	Indicates a library for the referenced entity. If you choose not to specify a library, the Quartus Prime Pro Edition software automatically defaults to the `altera_work` library.
`-no_sdc_promotion` `-no_auto_inst_discovery`	Converts any entity-bound SDC into a global SDC file read just once for the entire compilation, making it particularly fitting for bundling an SDC with an entity designated for export as a `qdb` file. This configuration still permits SDC's collection filters to specify global top-level paths in your `get` commands.

2.3.5.2.1. Entity-Bound SDC Constraint Scope

The entity-bound SDC approach offers diverse scoping possibilities for your constraints, each dictating the extent of their influence.

Table 10. Entity-bound Constraint Scope

Constraint Scope Type	Features	To Enable Instance-bound Scoping
Automatic	• Default mode applied to entity-bound SDC files defined in the Quartus Prime Pro Edition GUI. • Under automatic scoping, constraints apply to every instance of the assigned entity across the project. • Each result from any `get` command (for example, `get_pins`, `get_ports`, and so on) in the SDC file is prepended with the instance's path. • All `get` commands are confined to the target elements within the bound instance associated with the SDC file.	Default mode for `SDC_ENTITY_FILE`. No additional steps required.
Manual	• Accessible by QSF assignment only. • The `-no_sdc_promotion` setting disables automatic scoping, necessitating full hierarchical path for targeting nodes. • Allows the flexibility to target nodes beyond entity boundaries. • The `get_current_instance` command specifies the top-level path to the current instance, allowing you to merge filters targeting elements in the current instance with commands addressing those beyond entity boundaries.	Use `-no_sdc_promotion`. Append each collection filter with `get_current_instance` to target nodes within the entity boundaries. For example: `get_registers [get_current_instance]|reg[*]`
Disabled	• Accessible by QSF assignment only. • `-no_sdc_promotion` and the `-no_auto_inst_discovery` arguments together disable scoping, treating an entity-bound SDC as a global SDC file. The SDC file is read only once for the entire compilation instead of processing repeatedly for each instance it is linked to. • This mode is ideal when bundling an SDC with an entity destined for export as a `qdb` file, while preserving the capability for SDC's collection filters to specify global, top-level paths in their `get` commands.	Use `-no_sdc_promotion` and `-no_auto_inst_discovery` arguments.

When you define entity-bound SDC files either through the GUI or via `.qsf` assignments (excluding the `-no_sdc_promotion` and `-no_auto_inst_discovery` arguments), the constraints use automatic scoping. Automatic scoping involves

prepending filters with the instance's path. To provide clarity, the following table illustrates how paths are interpreted in various Tcl commands due to the automatic scoping of constraints:

Table 11. Automatic Scope of Constraints

Constraint Example	Auto-Scope Constraint Interpretation for Instance X|Y
`set_false_path -from [get_keepers a]`	`set_false_path -from [get_keepers X|Y|a]`
`set_false_path -from [get_registers a] -to [get_registers b]`	`set_false_path -from [get_registers X|Y|a] -to [get_registers X|Y|b]`
`set_false_path –from [get_clocks clk_1] –to [get_clocks clk_2]`	`set_false_path –from [get_clocks clk_1] –to [get_clocks clk_2]`
`set_max_delay –from [get_ports in] -to [get_registers A] 2.0`	`set_max_delay –from [get_ports in] -to [get_registers X|Y|A] 2.0`
`get_ports *`	`get_ports *`
`get_clocks *`	`get_clocks *`
`get_ports a`	`get_ports a`
`get_clocks a`	`get_clocks a`

When you disable automatic scoping through `.qsf` assignments, including the use of the `-no_sdc_promotion` argument, you must manually prepend the top-level path to achieve the same behavior as automatic scoping. To simplify this process, use the `-get_current_instance` command to return the top-level path of the current instance. The following table illustrates how paths are interpreted when the `-get_current_instance` command is employed to add the top-level path to certain Tcl commands:

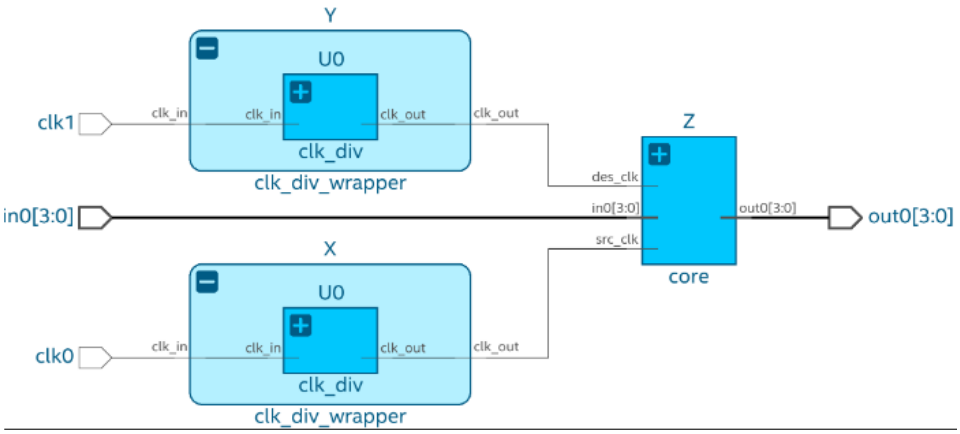
Table 12. Manual Scope of Constraints

Constraint Example	Manual Scope Constraint Interpretation
`set_false_path –from [get_current_instance]|d\` `  –to [get_current_instance]|e`	`set_false_path –from i1|inner|d –to i1|inner|e` `set_false_path –from i2|inner|d –to i2|inner|e` `set_false_path –from i3|d –to i3|e`
`create_generated_clock –divide_by 2 –source \` `  [get_ports inclk] –name \` `  [get_current_instance]_divclk \` `  [get_current_instance]|div` `set_multicycle_path –from [get_current_instance]|a \` `  –to [get_current_instance]|b 2`	`create_generated_clock –divide_by 2 –source \` `  [get_ports inclk] –name “i1_divclk” i1|div` `set_multicycle_path –from i1|a –to i1|b 2 \` `  create_generated_clock –divide_by 2 –source \` `  [get_ports inclk] –name “i2_divclk” i2|div` `set_multicycle_path –from i2|a –to i2|b 2`

2.3.5.2.2. Automatic Scope Entity-bound Constraint Example

In the following design, two instances of the `clk_div` entity are constrained using the entity-bound SDC file approach. After running the Fitter's Plan stage and the timing netlist becomes available, follow the steps below to associate an SDC file with a particular entity in your design:

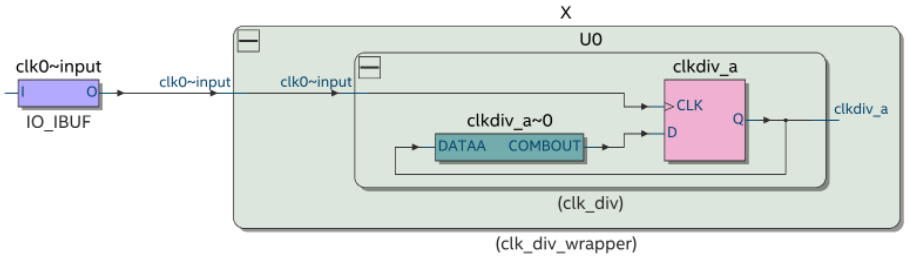
Figure 72. Automatic Scope Example



1. Create a non-entity-bound SDC file that defines constraints targeted by a global scope and add it to the project. For this design, two clocks (`clk0` and `clk1` are defined.

```
# global.sdc
create_clock -period 100MHz -name clk_100 [get_ports clk0]
create_clock -period 75MHz -name clk_75 [get_ports clk1]
```

2. Create a second SDC file to constrain your target entity and add it to the project. This file follows the entity-bound approach and associates with the `clk_div` entity. Consequently, you define constraints as if this entity were at the top-level hierarchy, with path names relative to the entity. For example, the `get_pins clkdiv_a|q` command does not require the `X|U0` hierarchy. In this example, the `get_current_instance` Tcl command generates a unique name for each clock.



In this specific case, the `.sdc` file creates a new clock on the output of the module, and the `get_current_instance` Tcl command generates a unique name for each clock.

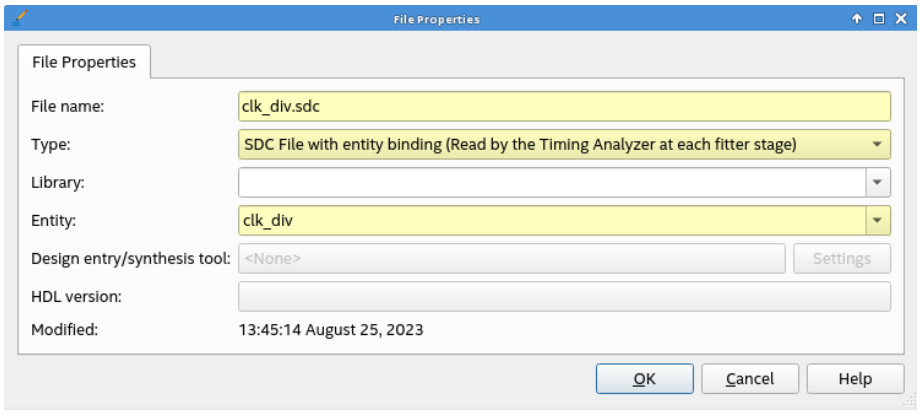
```
# clk_div.sdc
set unique_clock_name "[get_current_instance]_clkout"
create_generated_clock -divide_by 2 -source [get_pins clkdiv_a|clk] -name
$unique_clock_name [get_pins clkdiv_a|q]
```

3. Open the **File Properties** dialog for the `.sdc` file associated with the `clk_div` entity from the files list in the left-hand **Project Tasks** pane.
   a. Select the **SDC File with entity binding (Read by the Timing Analyzer at each fitter stage)** option in the **Type** list.
   b. In the **Entity** list, select **`clk_div`**.

      This step applies automatic scoping to the entity-bound file, where all paths in filter commands are prepended with the top-level path of the current hierarchy. For instance, the command `get_registers clkdiv_a` is dynamically transformed into its fully hierarchical counterpart, such as `get_registers X|U0|clkdiv_a`.

      Note: You must first complete Analysis & Synthesis to populate the list of entities.

Figure 73. File Properties Dialog

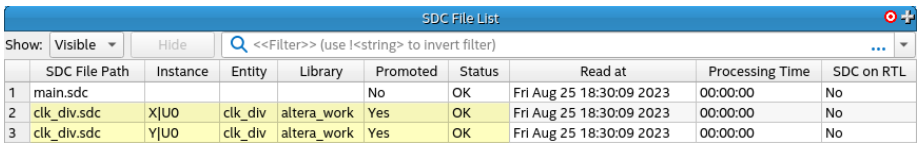


Alternatively, you can use the following `.qsf` assignment to set the entity binding:

```
set_global_assignment -name SDC_ENTITY_FILE clk_div.sdc -entity clk_div
```

4. Recompile the project to apply the changes. This results in the corresponding `.sdc` file being effectively bound to the entity in the automatic scope mode.
5. Verify the implementation of the entity-bound property by reviewing the **SDC File List** report in the Timing Analyzer. This report provides a comprehensive list of the applied SDC files for the design. For entity-bound SDC files, the report includes the associated instance, entity name, library, and the status of automatic scoping:

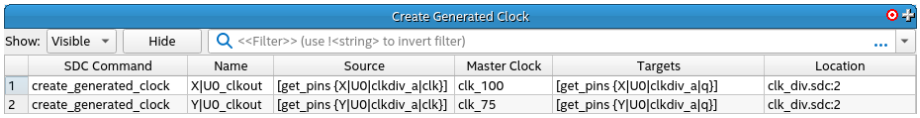
Figure 74. SDC File List Report in the Timing Analyzer



	SDC File Path	Instance	Entity	Library	Promoted	Status	Read at	Processing Time	SDC on RTL
1	main.sdc				No	OK	Fri Aug 25 18:30:09 2023	00:00:00	No
2	clk_div.sdc	X|U0	clk_div	altera_work	Yes	OK	Fri Aug 25 18:30:09 2023	00:00:00	No
3	clk_div.sdc	Y|U0	clk_div	altera_work	Yes	OK	Fri Aug 25 18:30:09 2023	00:00:00	No

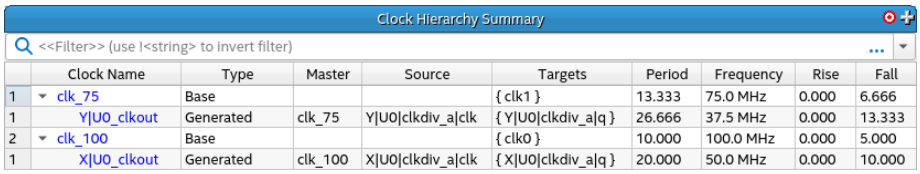
6. Determine the correct application of each constraint according to the intended purpose. For example, examine the generated clocks and cross-reference with the clock hierarchy to determine if the constraints are successfully applied, as shown in the following images:

Figure 75. Create Generated Clock Window in the Timing Analyzer



	SDC Command	Name	Source	Master Clock	Targets	Location
1	create_generated_clock	X|U0_clkout	[get_pins {X|U0|clkdiv_a|clk}]	clk_100	[get_pins {X|U0|clkdiv_a|q}]	clk_div.sdc:2
2	create_generated_clock	Y|U0_clkout	[get_pins {Y|U0|clkdiv_a|clk}]	clk_75	[get_pins {Y|U0|clkdiv_a|q}]	clk_div.sdc:2

Figure 76. Clock Hierarchy Summary Window in the Timing Analyzer



	Clock Name	Type	Master	Source	Targets	Period	Frequency	Rise	Fall
1	clk_75	Base			{ clk1 }	13.333	75.0 MHz	0.000	6.666
1	Y|U0_clkout	Generated	clk_75	Y|U0|clkdiv_a|clk	{ Y|U0|clkdiv_a|q }	26.666	37.5 MHz	0.000	13.333
2	clk_100	Base			{ clk0 }	10.000	100.0 MHz	0.000	5.000
1	X|U0_clkout	Generated	clk_100	X|U0|clkdiv_a|clk	{ X|U0|clkdiv_a|q }	20.000	50.0 MHz	0.000	10.000

2.3.5.2.3. Manual Scope Entity-bound Constraint Example

To modify the scope of the entity-bound SDC file to manual, for example, to target elements outside the entity, utilize `.qsf` assignments. By including the `-no_sdc_promotion` parameter in the entity-bound SDC file definition, you can prevent the Compiler from prepending each collection filter with the full path of the current instance. For example, to change to manual scope, the `.sdc` file associated with the `clk_div` entity in the automatic scope example above, add the `-no_sdc_promotion` parameter as follows:

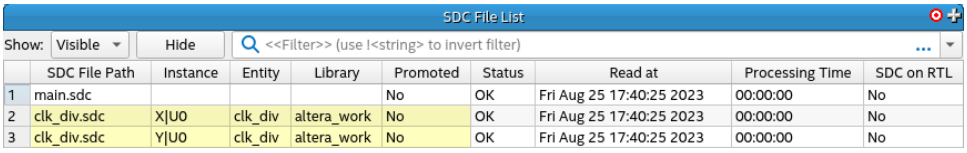
```
set_global_assignment -name SDC_ENTITY_FILE clk_div.sdc -entity clk_div -
no_sdc_promotion
```

To scope your collection filters precisely, use the `get_current_instance` command. By specifying the top-level path to the present instance, `get_current_instance` streamlines the process of scoping filters directed towards elements confined within instance boundaries.

```
# clk_div.sdc
set current_entity_instance [get_current_instance]
set unique_clock_name "${current_entity_instance}_clkout"
create_generated_clock -divide_by 2 -source [get_pins clkdiv_a|clk] \
-name $unique_clock_name [get_pins ${current_entity_instance}|clkdiv_a|q]
```

If you disable the automatic scope for a designated entity-bound SDC file, the **SDC File List** report within the Timing Analyzer indicates this change by displaying "No" in the promoted column:

Figure 77. SDC File List Report in the Timing Analyzer

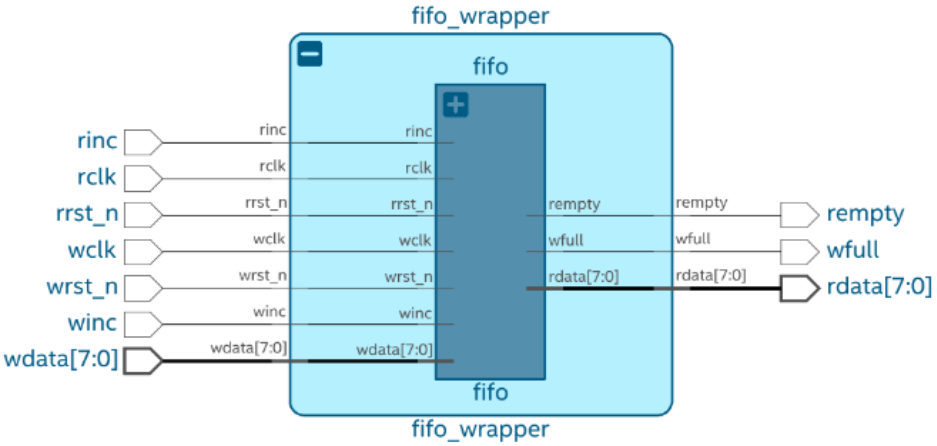


	SDC File Path	Instance	Entity	Library	Promoted	Status	Read at	Processing Time	SDC on RTL
1	main.sdc				No	OK	Fri Aug 25 17:40:25 2023	00:00:00	No
2	clk_div.sdc	X|U0	clk_div	altera_work	No	OK	Fri Aug 25 17:40:25 2023	00:00:00	No
3	clk_div.sdc	Y|U0	clk_div	altera_work	No	OK	Fri Aug 25 17:40:25 2023	00:00:00	No

2.3.5.2.4. Exporting a Design Partition with Entity-bound Constraints

The following example illustrates exporting a partition that includes entity bound constraints for use in another project. This example uses the `fifo` entity.

Figure 78. Entity-Bound Constraints Design Example



Perform these steps to export a design partition with entity-bound constraints:

1. After applying the constraints to the `fifo` entity, click **Fitter** on the Compilation Dashboard to run the Fitter. The Messages window report when the Fitter is complete.
2. Click **Assignments ➤ Design Partitions Window** and define **Default Type** design partition for the `fifo` entity in the **Assignments View** tab of the **Design Partition** dialog box.

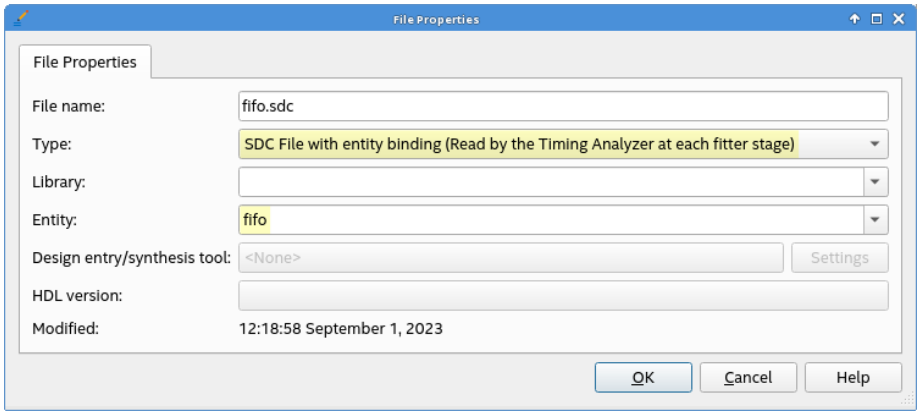
Figure 79. Assignments View of the Design Partition Window

Assignments View | Compilation View

Partition Name	Hierarchy Path	Type	Preservation Level	Empty	Partition Database File
<<new>>					
root_partition	|				
fifo	fifo_wrapper|fifo	Default	Not Set	No	

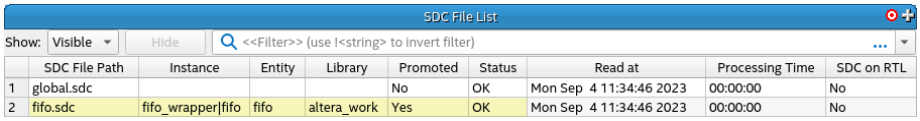
3. Specify the entity-bound SDC **File name** and **Type** to establish the binding between the `fifo` entity and the `.sdc` file.

Figure 80. File Properties Dialog



4. Click **Compile Design** on the Compilation Dashboard to run a full compilation and apply the changes to your project. The Timing Analyzer opens by default following a successful full compilation.
5. Confirm the correct association of the `.sdc` file with the `fifo` entity by reviewing the **SDC File List** report in the Timing Analyzer.

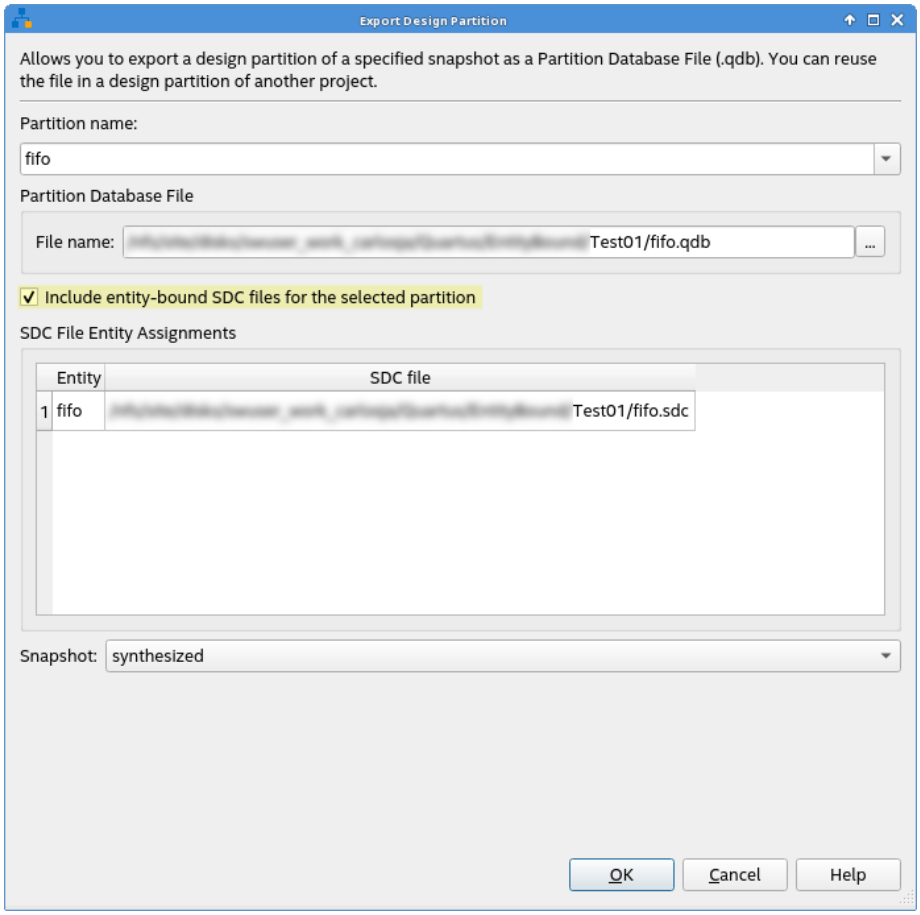
Figure 81. SDC File List Report in the Timing Analyzer



	SDC File Path	Instance	Entity	Library	Promoted	Status	Read at	Processing Time	SDC on RTL
1	global.sdc				No	OK	Mon Sep 4 11:34:46 2023	00:00:00	No
2	fifo.sdc	fifo_wrapper|fifo	fifo	altera_work	Yes	OK	Mon Sep 4 11:34:46 2023	00:00:00	No

6. Click **Project ➤ Export Design Partition** and specify the following options:
   a. In the **Partition name** list, select the partition to export.
   b. In **Partition Database File**, specify the partition database file.
   c. Turn on the **Include entity-bound SDC files for the selected partition** option for entity-bound SDC files linked to the chosen partitions.
   d. For **Snapshot**, specify **Synthesized** or **Final**.

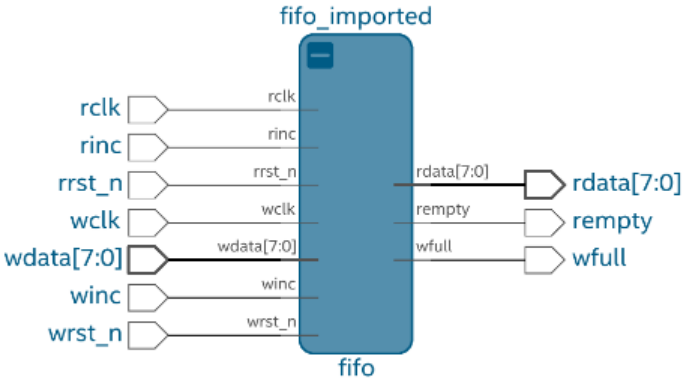
Figure 82. Export Design Partition Dialog



2.3.5.2.5. Importing a Design Partition with Entity-bound Constraints

Importing a partition with entity-bound SDC files requires that you define a black box wrapper for the entity in your design. This wrapper declares a partition in which the data from the imported partition is utilized. Consider the following example with an entity named `fifo_imported`:

Figure 83. Example of an Entity Named `fifo_imported`



To import a design partition with entity-bound constraints, perform these steps:

1. To run design synthesis, click **Analysis & Synthesis** on the Compilation Dashboard.
2. To create a new partition within the wrapper entity, click **Assignments ➤ Design Partitions Window** and specify the following options on the **Assignments View** tab:
   a. Specify the **Partition Name** and the **Hierarchy Path** of the entity instance.
   b. Specify **Default** for the partition **Type**.
   c. Specify **synthesized** in the **Preservation Level** column.
   d. Specify the `.qdb` file from the previous project as the **Partition Database File**

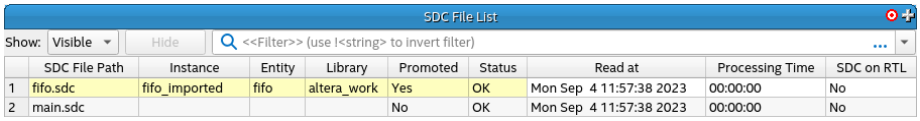
Figure 84. Assignments View of the Design Partition Dialog

Assignments View | Compilation View

Partition Name	Hierarchy Path	Type	Preservation Level	Empty	Partition Database File
<<new>>					
root_partition					
fifo_imported	*fifo_imported*	*Default*	*synthesized*	*No*	*../Test01/fifo.qdb*

3. To run a full compilation, click **Compile Design** on the Compilation Dashboard. The Timing Analyzer appears automatically following successful compilation.
4. Verify the correct application of the entity-bound SDC file assignment in the **SDC File List** report.

Figure 85. SDC File List Report in the Timing Analyzer



	SDC File Path	Instance	Entity	Library	Promoted	Status	Read at	Processing Time	SDC on RTL
1	fifo.sdc	fifo_imported	fifo	altera_work	Yes	OK	Mon Sep 4 11:57:38 2023	00:00:00	No
2	main.sdc				No	OK	Mon Sep 4 11:57:38 2023	00:00:00	No

2.3.6. Constraining Design Partition Ports

You can assign clock definitions and SDC exceptions to design partition ports. The block-based design and partial reconfiguration design flows require the use of design partitions.

The Compiler represents design partition ports in your timing netlist as combinational nodes with persistent names that the Compiler cannot optimize away. You can safely refer to these ports as clock sources or `-through` points in SDC constraints. You can also use design partition port names as `-to` and `-from` points in the `report_path` command.

If a port on `partition_a` has the name `clk_divide`, then the SDC constraint is:

```
create_generated_clock –source clock -divide_by 2 \
     top|partition_a|clk_divide
```

If a set of ports on `partition_b` has the name `data_input[0..7]`, then the SDC constraint is:

```
set_multicycle_path –from top|partition_a|data_reg* \
     -through top|partition_b|data_input* 2
```

You can use multiple `-through` clauses. This technique allows you to specify paths that go through output ports of one design partition, and then through the input ports of another, downstream design partition.

To add constraints to partition ports:

1. Run Analysis & Synthesis or run full compilation on a design containing design partitions.
2. To open the RTL Viewer and locate the partition ports of interest, click **Tools ➤ Netlist Viewers ➤ RTL Viewer**.
3. Using the same names as the **RTL Viewer**, add clock and other SDC constraints to the `.sdc` file for your project. You can use wildcards to refer to more than one port.
4. Recompile the design to apply the new definitions and constraints.

   Aside from block-based and PR flows, this technique also aids in emulation of ASICs using FPGAs. In this type of design, clock networks often span multiple hierarchies of partitions. Typically, designers remove the clock-dividing circuitry from the netlist, since they cannot easily emulate this circuitry on Intel FPGAs. For such clock networks, this technique allows you to define different versions of the clock signal in places where the circuitry is removed.

   You must design and place your partitions strategically, and then define the appropriate ports on these partitions. Ensure that your ports and partitions coincide with the part of the clock network which contains the special circuitry. You can manually edit the emulated ASIC netlist to annotate appropriate clock

definitions and clock relationships. You can also use this technique in any projects where arbitrary locations on paths require constrained timing or defined clock sources.

Related Information

- Output Constraints (set_output_delay) on page 95
- Input Constraints (set_input_delay) on page 94

2.3.6.1. Timing Analysis of Imported Compilation Results

You can preserve the compilation results for your design as a version-compatible Quartus database file (`.qdb`) that you can open in a later version of the Quartus Prime software without compatibility issues.

When you import and open the `.qdb` in a later version of software, you can run timing analysis on the imported compilation results without re-running the Compiler.

2.3.7. Using Fitter Overconstraints

Fitter overconstraints are timing constraints that you adjust to overcome modeling inaccuracies, mis-correlation, or other deficiencies in logic optimization. You can overconstrain setup and hold paths in the Fitter to enable more aggressive timing optimization of specific paths.

Overconstraints for Stratix 10 Designs

One typically writes overconstraints that apply only during the Fitter, with a conditional statement that checks the name of the executable that is reading the `.sdc` file. The following example shows a conditional statement that applies an overconstraint during the Fitter.

```
# Example Fitter overconstraint targeting specific nodes
if { $::TimingAnalyzerInfo(nameofexecutable) eq "quartus_fit"} {
  set_max_delay -from ${my_src_regs} -to ${my_dst_regs} 1ns
}
```

One typically applies overconstraints because a particular path (or set of paths) requires more than the default optimization effort from the Fitter. Therefore, overconstraints typically specify particular node or register names in the path or paths that require extra optimization.

For devices that support the Hyperflex® architecture, such as Stratix 10 and Agilex™ FPGA portfolio devices, timing constraints that apply to particular node or register names prevent the Hyper-Retimer from optimizing paths containing those nodes or registers.

Because the Hyper-Retimer is component of the Fitter, an overconstraint that conditionally applies during the Fitter is counter-productive during the Hyper-Retimer portion of the Fitter. The overconstraint actually prevents the optimization instead of focusing the optimization effectively.

When designing for the Hyperflex architecture, use the `is_post_route` Tcl function form of conditional statement to apply the overconstraint during placement and routing but not during the Hyper-Retimer. `is_post_route` allows you to apply

overconstraints and adjust slack for stages of the Fitter, such as Plan, Place, and Route. `is_post_route` also allows post-route retiming via the Hyper-Retimer without affecting sign-off timing analysis.

```
# Example Fitter overconstraint targeting specific nodes (allows for post-route
retiming)
if { ! [is_post_route]} {
  set_max_delay -from ${my_src_regs} -to ${my_dst_regs} 1ns
}
```

Note: The `is_post_route` function is inclusive. To exclude the function, use the negation syntax (`!`).

Overconstraints for Designs that Target All Other Device Families

You can assign Fitter overconstraints that check the name of the current executable, (either `quartus_fit` or `quartus_sta`) to apply different constraints for Fitter optimization and sign-off timing analysis.

```
set fit_flow 0
if { $::TimingAnalyzerInfo(nameofexecutable) == "quartus_fit" } {
   set fit_flow 1
}
if {$fit_flow} {
  # Example Fitter overconstraint targeting specific nodes (restricts retiming)
  set_max_delay -from ${my_src_regs} -to ${my_dst_regs} 1ns
}
```

Other Overconstraint Combinations

You can apply timing constraints to specific stages of the Compiler, in addition to the Fitter as described above. The following table shows different combinations of compile stages and the conditional expression that applies those constraints during the stage or stages.

Table 13. Overconstraint Combinations

Compile Stages	Tcl Condition	Notes
Fitter	`if { $::TimingAnalyzerInfo(nameofexecutable) eq "quartus_fit" }`	Applies constraints during entire Fitter stage. Use for fitter overconstraints with devices that do not support Hyperflex architecture.
Signoff timing	`if { $::TimingAnalyzerInfo(nameofexecutable) eq "quartus_sta" }`	Applies constraints during timing analysis. This condition is uncommon. Typically you apply timing analysis constraints during the entire compile flow.
Plan, Place, and Route	`if { ! [is_post_route] }`	Applies constraints during the Plan, Place, and Route stages in devices that support the Hyperflex architecture. Use for fitter overconstraints .
Retime (HyperRetimer), Finalize, Signoff timing	`if { [is_post_route] }`	Applies constraints after the Route stage completes in devices that support the Hyperflex architecture. This combination of stages is

continued...

Compile Stages	Tcl Condition	Notes
		uncommon. Typically you apply timing analysis constraints during the entire compile flow.
Retime (HyperRetimer) and Finalize	`if { [is_post_route] && $::TimingAnalyzerInfo(nameofexecutable) eq "quartus_fit" }`	Applies constraints during Retiming and Finalize stages in devices that support the Hyperflex architecture. This combination of stages is uncommon.

2.4. Timing Constraint Descriptions

This section provides examples and describes how to correctly apply SDC timing constraints that guide design synthesis, Fitter placement, and produce accurate timing analysis results.
You can define a set of initial timing constraints, and then iteratively modify those constraints as the design progresses.

Early in the design cycle, you can use SDC-on-RTL constraints to target analysis of RTL nodes. This analysis provides a stable reference for constraints that can remain unchanged in subsequent compilation stages, such as clock definitions. Establishing a set of SDC-on-RTL constraints enables their propagation and application throughout the entire design cycle. Concurrently, you can create a conventional `.sdc` file for analysis of the remaining design elements, providing flexibility for iterative constraint adjustments as the design evolves.

This section also outlines the proper application of recommended conventional SDC timing constraints. Conventional SDC constraints guide Fitter placement via `.sdc` files, offering alternative approaches to achieve precise control over constraints throughout the design flow.

2.4.1. Clock Constraints

You must define all clocks and any associated clock characteristics, such as uncertainty, latency or skew. The Timing Analyzer supports `.sdc` commands that accommodate various clocking schemes, such as:

- Base clocks
- Virtual clocks
- Multifrequency clocks
- Generated clocks

You can verify correct implementation of clock constraints by using **Report Clocks** (`report_clocks`) to generate clock timing reports. You can use **Check Timing** (`check_timing`) to report problems with a variety of timing constraints, such as missing clocks.

Related Information

- Report Clocks on page 135
- Check Timing on page 161

2.4.1.1. Creating Base Clocks

Base clocks are the primary input clocks to the device. The **Create Clock** (`create_clock`) constraint allows you to define the properties and requirements for base clocks in the design. Unlike clocks that are generated in the device (such as an on-chip PLL), base clocks are generated by off-chip oscillators or forwarded from an external device. Define base clocks at the top of your `.sdc` file, because generated clocks and other constraints often reference base clocks. The Timing Analyzer ignores any constraints that reference an undefined clock.

The following examples show common use of the `create_clock` constraint:

create_clock Command

The following specifies a 100 MHz requirement on a `clk_sys` input clock port:

```
create_clock -period 100Mhz -name clk_sys [get_ports clk_sys]
```

100 MHz Shifted by 90 Degrees Clock Creation

The following creates a 10 ns clock, with a 50% duty cycle, that is phase shifted by 90 degrees, and applies to port `clk_sys`. This type of clock definition commonly refers to source synchronous, double-rate data that is center-aligned with respect to the clock.

```
create_clock -period 10ns -waveform { 2.5 7.5 } [get_ports clk_sys]
```

Two Oscillators Driving the Same Clock Port

You can apply multiple clocks to the same target with the `-add` option. For example, to specify that you can drive the same clock input at two different frequencies, enter the following commands in your `.sdc` file:

```
create_clock -period 10ns -name clk_100 [get_ports clk_sys]
create_clock -period 5ns -name clk_200 [get_ports clk_sys] -add
```

Although uncommon to define more than two base clocks for a port, you can define as many as are appropriate for your design, making sure you specify `-add` for all clocks after the first.

Creating Multifrequency Clocks

You must create a multifrequency clock if your design has more than one clock source feeding a single clock node. The additional clock may act as a low-power clock, with a lower frequency than the primary clock. If your design uses multifrequency clocks, use the `set_clock_groups` command to define clocks that are physically exclusive (that is, clocks that are not physically present at the same time).

Use the `create_clock` command with the `-add` option to create multiple clocks on a clock node. You can create a 10 ns clock applied to clock port `clk`, and then add an additional 15 ns clock to the same clock port. The Timing Analyzer analyzes both clocks.

```
create_clock –period 10ns –name clock_primary –waveform { 0 5 } \
    [get_ports clk]
create_clock –period 15ns –name clock_secondary –waveform { 0 7.5 } \
    [get_ports clk] -add
```

You can verify correct implementation of clock constraints by using **Report Clocks** (`report_clocks`) to generate clock timing reports. You can use **Check Timing** (`check_timing`) to report problems with a variety of timing constraints, such as missing clocks.

Related Information

- Accounting for Clock Effect Characteristics on page 90
- Report Clocks on page 135
- Check Timing on page 161

2.4.1.1.1. Automatic Clock Detection and Constraint Creation

Use the `derive_clocks` command to automatically create base clocks in your design The `derive_clocks` command is equivalent to using the `create_clock` command for each register or port feeding the clock pin of a register. The `derive_clocks` command creates clock constraints on ports or registers to ensure every register in your design has a clock constraints, and it applies one period to all base clocks in your design.

The following command specifies a base clock with a 100 MHz requirement for unconstrained base clock nodes.

```
derive_clocks -period 10
```

The `derive_clocks` command names the automatically created base clocks according to the name of the register or port that is the target of each clock. Automatically derived clocks have the suffix "~derived". You can choose another suffix to append with the `-suffix` option for the `derive_clocks` command.

Caution: If your design has more than a single clock, the `derive_clocks` command constrains all the clocks to the same specified frequency. To achieve a realistic analysis of your design's timing requirements, do not use `derive_clocks` command for final timing sign-off. Instead, use `create_clock` and `create_generated_clock` commands to make individual clock constraints for all clocks in your design.

If you want to create some base clocks automatically, use the `-create_base_clocks` option to `derive_pll_clocks`. With this option, the `derive_pll_clocks` command automatically creates base clocks for each PLL, based on the input frequency information that you specify when you generate the PLL. This feature works for simple port-to-PLL connections. Base clocks do not automatically generate for complex PLL connectivity, such as cascaded PLLs. You can also use the command `derive_pll_clocks -create_base_clocks` to create the input clocks for all PLL inputs automatically.

2.4.1.2. Creating Virtual Clocks

A virtual clock is a clock without a real source in the design, or a clock that does not interact directly with the design. You can use virtual clocks in I/O constraints to represent clocks that drive external devices connected to the FPGA..

To create virtual clocks, use the `create_clock` constraint with no value for the *<targets>* option.

This following example defines a 100 MHz virtual clock because the command includes no <*targets*>.

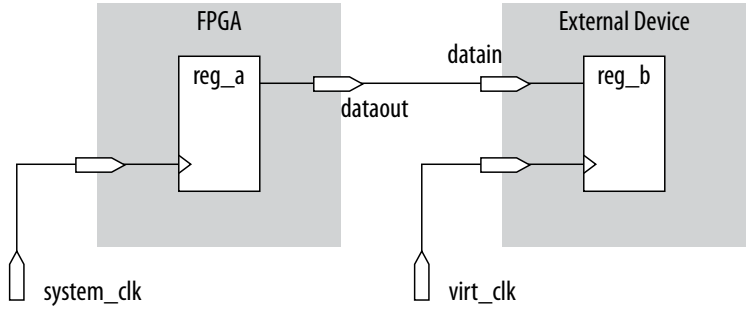
```
create_clock -period 10 -name my_virt_clk
```

I/O Constraints with Virtual Clocks

You can use a base clock to constrain the circuit in the FPGA and a virtual clock to represent the clock driving the external device. .

Figure 86. Virtual Clock Board Topology

The figure shows the base clock (`system_clk`), virtual clock (`virt_clk`), and output delay for the virtual clock constraints example



The following example creates the 10 ns `virt_clk` virtual clock, with a 50% duty cycle, with the first rising edge occurring at 0 ns. The virtual clock can then become the clock source for an output delay constraint.

Example 3. Virtual Clock Constraints

```
#create base clock for the design
create_clock -period 5 [get_ports system_clk]
#create the virtual clock for the external register
create_clock -period 10 -name virt_clk
#set the output delay referencing the virtual clock
set_output_delay -clock virt_clk -max 1.5 [get_ports dataout]
set_output_delay -clock virt_clk -min 0.0 [get_ports dataout]
```

You can verify correct implementation of clock constraints by using **Report Clocks** (`report_clocks`) to generate clock timing reports. You can use **Check Timing** (`check_timing`) to report problems with a variety of timing constraints, such as the number of unreferenced virtual clocks without constraint.

Related Information

- Report Clocks on page 135
- Check Timing on page 161

2.4.1.2.1. Specifying I/O Interface Uncertainty

Virtual clocks are recommended for I/O constraints because they most accurately represent the clocking topology of the design. An additional benefit is that you can specify different uncertainty values on clocks that interface with external I/O ports and clocks that feed register-to-register paths inside the FPGA.

2.4.1.2.2. I/O Interface Clock Uncertainty Example

To specify I/O interface uncertainty, you must create a virtual clock and constrain the input and output ports with the `set_input_delay` and `set_output_delay` commands that reference the virtual clock.

When the `set_input_delay` or `set_output_delay` commands reference a clock port or PLL output, the virtual clock allows the `derive_clock_uncertainty` command to apply separate clock uncertainties for internal clock transfers and I/O interface clock transfers.

Create the virtual clock with the same properties as the original clock that is driving the I/O port, as the following example shows:

Example 4. SDC Commands to Constrain the I/O Interface

```
# Create the base clock for the clock port
create_clock -period 10 -name clk_in [get_ports clk_in]
# Create a virtual clock with the same properties of the base clock
# driving the source register
create_clock -period 10 -name virt_clk_in
# Create the input delay referencing the virtual clock and not the base
# clock
# DO NOT use set_input_delay -clock clk_in <delay value>
# [get_ports data_in]
set_input_delay -clock virt_clk_in <delay value> [get_ports data_in]
```

2.4.1.3. Creating Generated Clocks (create_generated_clock)

The **Create Generate Clock** (`create_generated_clock`) constraint allows you to define the properties and constraints of an internally generated clock in the design. You specify the **Clock name** (`-name`), the **Source** node (`-source`) from which clock derives, and the **Relationship to the source** properties. Define generated clocks for any node that modifies the properties of a clock signal, including modifying the phase, frequency, offset, or duty cycle.

You apply generated clocks most commonly on the outputs of PLLs, on register clock dividers, clock muxes, and clocks forwarded to other devices from an FPGA output port, such as source synchronous and memory interfaces. In the `.sdc` file, enter generated clocks after the base clocks definitions. Generated clocks automatically account for all clock delays and clock latency to the generated clock target.

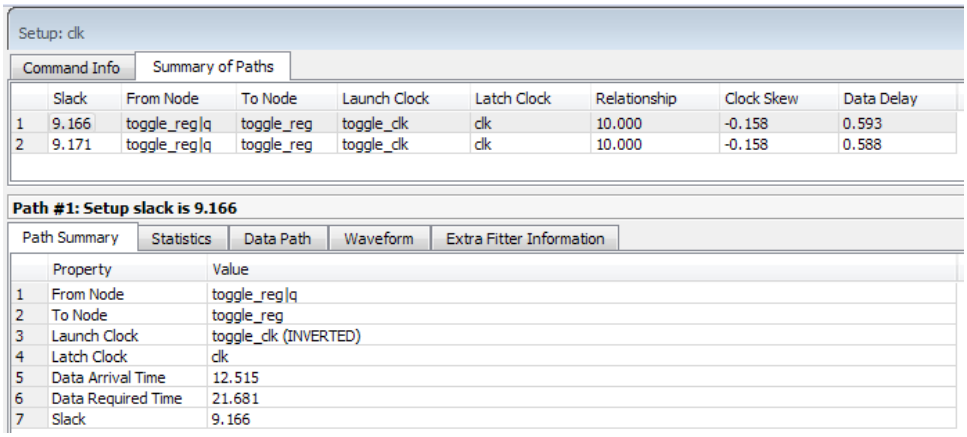
The `-source` option specifies the name of a node in the clock path that you use as reference for your generated clock. The source of the generated clock must be a node in your design netlist, and not the name of a clock you previously define. You can use any node name on the clock path between the input clock pin of the target of the generated clock and the target node of its reference clock as the source node.

Specify the input clock pin of the target node as the source of your new generated clock. By accepting a node as the generated clock's source clock, the generated clock constraint decouples from the source clock. If you change the source clock for the generated clock and the source node is the same, you do not have to edit the generated clock constraint.

If you have multiple base clocks feeding a node that is the source for a generated clock, you must define multiple generated clocks. You associate each generated clock with one base clock using the `-master_clock` option in each generated clock statement. In some cases, generated clocks generate with combinational logic.

Depending on how your clock-modifying logic synthesizes, the source or target node can change from one compilation to the next. If the name changes after you write the generated clock constraint, the Compiler ignores the generated clock because that target name no longer exists in the design. To avoid this problem, use a synthesis attribute or synthesis assignment to retain the final combinational node name of the clock-modifying logic. Then use the kept name in your generated clock constraint.

Figure 87. Example of clock-as-data



When you create a generated clock on a node that ultimately feeds the data input of a register, this creates a special case of "clock-as-data." The Timing Analyzer treats clock-as-data differently. For example, if you use clock-as-data with DDR, you must consider both the rise and the fall of this clock, and the Timing Analyzer reports both rise and fall. With clock-as-data, the Compiler treats the **From Node** as the target of the generated clock, and the **Launch Clock** as the generated clock.

In Example of clock-as-data, the first path is from `toggle_clk (INVERTED)` to `clk`, and the second path is from `toggle_clk` to `clk`. The slack in both cases is slightly different due to the difference in rise and fall times along the path. The **Data Delay** column reports the ~5 ps difference. Only the path with the lowest slack value requires consideration. The Timing Analyzer only reports the worst-case path between the two (rise and fall). In this example, if you do not define the generated clock on the register output, then timing analysis reports only one path with the lowest slack value.

You can use the `derive_pll_clocks` command to automatically generate clocks for all PLL clock outputs. The properties of the generated clocks on the PLL outputs match the properties you define for the PLL.

You can verify correct implementation of clock constraints by using **Report Clocks** (`report_clocks`) to generate clock timing reports. You can use **Check Timing** (`check_timing`) to report problems with a variety of timing constraints, such as the number of generated clocks that are invalid.

Related Information

- Deriving PLL Clocks on page 85
- Report Clocks on page 135
- Check Timing on page 161
- create_generated_clock

- derive_pll_clocks

2.4.1.3.1. Clock Divider Example (-divide_by)

A common form of generated clock is the divide-by-two register clock divider. The following example constraint creates a half-rate clock on the divide-by-two register.

```
create_clock -period 10ns -name clk_sys [get_ports clk_sys]
create_generated_clock -name clk_div_2 -divide_by 2 -source \
    [get_ports clk_sys] [get_pins reg|q]
```

To specify the clock pin of the register as the clock source:

```
create_clock -period 10ns -name clk_sys [get_ports clk_sys]
create_generated_clock -name clk_div_2 -divide_by 2 -source \
    [get_pins reg|clk] [get_pins reg|q]
```

Figure 88. Clock Divider

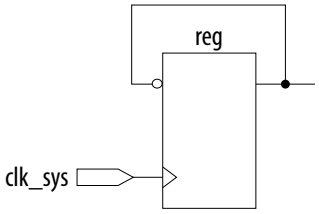
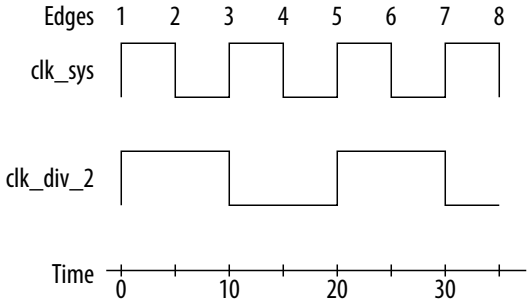


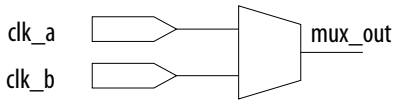
Figure 89. Clock Divider Waveform



2.4.1.3.2. Clock Multiplexer Example

The output of a clock multiplexer (mux) is a form of generated clock. Each input clock requires one generated clock on the output. The following `.sdc` example also includes the `set_clock_groups` command to indicate that the two generated clocks can never be active simultaneously in the design. Therefore, the Timing Analyzer does not analyze cross-domain paths between the generated clocks on the output of the clock mux.

Figure 90. Clock Mux




```
create_clock -name clock_a -period 10 [get_ports clk_a]
create_clock -name clock_b -period 10 [get_ports clk_b]
create_generated_clock -name clock_a_mux -source [get_ports clk_a] \
```

```
    [get_pins clk_mux|mux_out]
create_generated_clock -name clock_b_mux -source [get_ports clk_b] \
    [get_pins clk_mux|mux_out] -add
set_clock_groups -logically_exclusive -group clock_a_mux -group clock_b_mux
```

2.4.1.4. Deriving PLL Clocks

The **Derive PLL Clocks** (`derive_pll_clocks`) constraint automatically creates clocks for each output of any PLL in your design. `derive_pll_clocks` detects your current PLL settings and automatically creates generated clocks on the outputs of every PLL by calling the `create_generated_clock` command.

Note: Only Arria 10 and Cyclone 10 GX devices support the **Derive PLL Clocks** (`derive_pll_clocks`) constraint. For all other supported devices, the Timing Analyzer automatically derives PLL clocks from constraints bound to the related IP.

Create Base Clock for PLL input Clock Ports

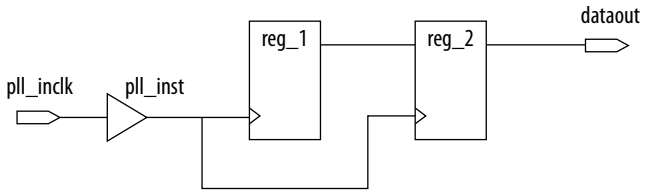
If your design contains transceivers, LVDS transmitters, or LVDS receivers, use the `derive_pll_clocks` to constrain this logic in your design and create timing exceptions for those blocks.

```
create_clock -period 10.0 -name fpga_sys_clk [get_ports fpga_sys_clk]
derive_pll_clocks
```

Include the `derive_pll_clocks` command in your `.sdc` file after any `create_clock` command. Each time the Timing Analyzer reads the `.sdc` file, the appropriate generated clock is created for each PLL output clock pin. If a clock exists on a PLL output before running `derive_pll_clocks`, the pre-existing clock has precedence, and an auto-generated clock is not created for that PLL output.

The following shows a simple PLL design with a register-to-register path:

Figure 91. Simple PLL Design



The Timing Analyzer generates messages like the following example when you use the `derive_pll_clocks` command to constrain the PLL.

Example 5. derive_pll_clocks Command Messages

```
Info:
Info: Deriving PLL Clocks:
Info: create_generated_clock -source pll_inst|altpll_component|pll|inclk[0] -
divide_by 2 -name
pll_inst|altpll_component|pll|clk[0] pll_inst|altpll_component|pll|clk[0]
Info:
```

The input clock pin of the PLL is the node `pll_inst|altpll_component|pll|inclk[0]` which is the `-source` option. The name of the output clock of the PLL is the PLL output clock node, `pll_inst|altpll_component|pll|clk[0]`.

If the PLL is in clock switchover mode, multiple clocks generate for the output clock of the PLL; one for the primary input clock (for example, `inclk[0]`), and one for the secondary input clock (for example, `inclk[1]`). Create exclusive clock groups for the primary and secondary output clocks since they are not active simultaneously.

You can verify correct implementation of clock constraints by using **Report Clocks** (`report_clocks`) to generate clock timing reports. You can use **Check Timing** (`check_timing`) to report problems with a variety of timing constraints, such as the number of instances where clocks that are assigned to a PLL do not correspond properly with the PLL settings you define in design files.

Related Information

- Creating Clock Groups (set_clock_groups) on page 86
- Report Clocks on page 135
- Check Timing on page 161

2.4.1.5. Creating Clock Groups (set_clock_groups)

The **Set Clock Groups** (`set_clock_groups`) constraint allows you to specify which clocks in the design are unrelated.

The `set_clock_groups` command allows you to cut timing between unrelated clocks in different groups. The Timing Analyzer performs the same analysis regardless of whether you specify `-exclusive` or `-asynchronous` groups. You define a clock group with the `-group` option. The Timing Analyzer excludes the timing paths between clocks for each of the separate groups.

The following tables show the impact of `set_clock_groups`.

Table 14. set_clock_groups -group A

Destination\Source	A	B	C	D
A	Analyzed	Cut	Cut	Cut
B	Cut	Analyzed	Analyzed	Analyzed
C	Cut	Analyzed	Analyzed	Analyzed
D	Cut	Analyzed	Analyzed	Analyzed

Table 15. set_clock_groups -group {A B}

Destination\Source	A	B	C	D
A	Analyzed	Analyzed	Cut	Cut
B	Analyzed	Analyzed	Cut	Cut
C	Cut	Cut	Analyzed	Analyzed
D	Cut	Cut	Analyzed	Analyzed

Table 16. set_clock_groups -group A -group B

Destination\Source	A	B	C	D
A	Analyzed	Cut	Analyzed	Analyzed

continued...

B	Cut	Analyzed	Analyzed	Analyzed
C	Analyzed	Analyzed	Analyzed	Analyzed
D	Analyzed	Analyzed	Analyzed	Analyzed

Table 17. set_clock_groups -group {A C} -group {B D}

Destination\Source	A	B	C	D
A	Analyzed	Cut	Analyzed	Cut
B	Cut	Analyzed	Cut	Analyzed
C	Analyzed	Cut	Analyzed	Cut
D	Cut	Analyzed	Cut	Analyzed

Table 18. set_clock_groups -group {A C D}

Destination\Source	A	B	C	D
A	Analyzed	Cut	Analyzed	Analyzed
B	Cut	Analyzed	Cut	Cut
C	Analyzed	Cut	Analyzed	Analyzed
D	Analyzed	Cut	Analyzed	Analyzed

You can verify correct implementation of clock constraints by using **Report Clocks** (`report_clocks`) to generate clock timing reports. You can use **Check Timing** (`check_timing`) to report problems with a variety of timing constraints, such as missing clocks.

Related Information

- Timing Exception Precedence on page 99
- Report Clocks on page 135
- Check Timing on page 161
- set_clock_groups Command, Quartus Prime Help

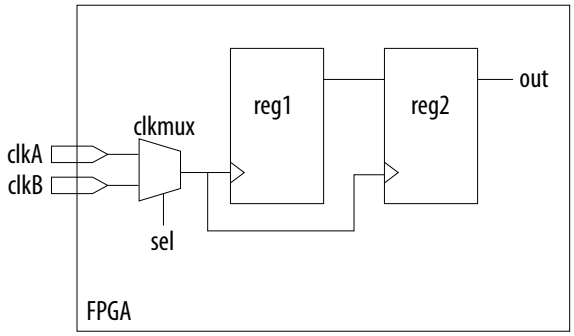
2.4.1.5.1. Exclusive Clock Groups (-logically_exclusive or -physically_exclusive)

You can use the `logically_exclusive` option to declare that two clocks are physically active simultaneously, but the two clocks are not actively used at the same time (that is, the clocks are logically mutually exclusive). The `physically_exclusive` option declares clocks that cannot be physically on the device at the same time.

If you define multiple clocks for the same node, you can use clock group assignments with the `logically_exclusive` option to declare clocks as mutually exclusive. This technique can be useful for multiplexed clocks.

For example, consider an input port that is clocked by either a 100-MHz or 125-MHz clock. You can use the `logically_exclusive` option to declare that the clocks are mutually exclusive and eliminate clock transfers between the 100-MHz and 125-MHz clocks, as the following diagrams and example SDC constraints show:

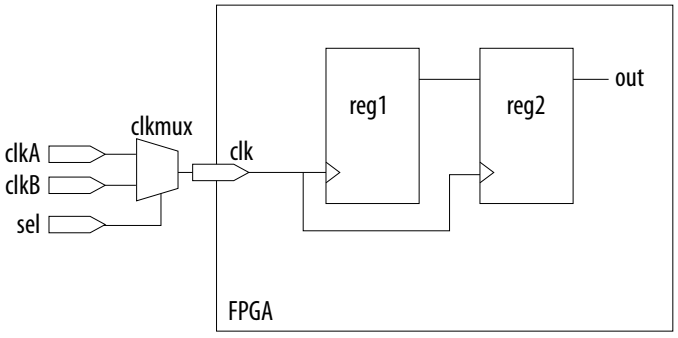
Figure 92. Synchronous Path with Clock Mux Internal to FPGA



Example SDC Constraints for Internal Clock Mux

```
# Create a clock on each port
create_clock -name clk_100 -period 10 [get_ports clkA]
create_clock -name clk_125 -period 8 [get_ports clkB]
# Create derived clocks on the output of the mux
create_generated_clock -name mux_100 -source [get_ports clkA] \
     [get_pins clkmux|combout]
create_generated_clock -name mux_125 -source [get_ports clkB] \
     [get_pins clkmux|combout] -add
# Set the two clocks as exclusive clocks
set_clock_groups -logically_exclusive -group {mux_100} -group {mux_125}
```

Figure 93. Synchronous Path with Clock Mux External to FPGA



Example SDC Constraints for External Clock Mux

```
# Create virtual clocks for the external primary clocks
create_clock -period 10 -name clkA
create_clock -period 20 -name clkB
# Create derived clocks on the port clk
create_generated_clock -name mux_100 -master_clock clkA [get_ports clk]
create_generated_clock -name mux_125 -master_clock clkB [get_ports clk] -add
# Assume no clock network latency between the external clock sources & the \
     clock mux output
set_clock_latency -source 0 [get_clocks {mux_100 mux_125}]
# Set the two clocks as exclusive clocks
set_clock_groups -physically_exclusive -group mux_100 -group mux_125
```

2.4.1.5.2. Asynchronous Clock Groups (-asynchronous)

Use the `-asynchronous` option to create asynchronous clock groups. You can use asynchronous clock groups to break the timing relationship when data transfers through a FIFO between clocks running at different rates.

2.4.1.5.3. set_clock_groups Constraint Tips

When you use `derive_pll_clocks` to create clocks, it can be time consuming to determine all the clock names to include in `set_clock_groups` constraints. However, you can use the following technique to somewhat automate clock constraint creation, even if you do not know all of the clock names.

1. Create a basic `.sdc` file that contains the Recommended Initial Conventional SDC Constraints, except omit the `set_clock_groups` constraint for now.
2. To add the `.sdc` to the project, click **Assignments ➤ Settings ➤ Timing Analyzer**. Specify the `.sdc` file under **SDC files to include in the project**.
3. To open the Timing Analyzer, click **Tools ➤ Timing Analyzer**.
4. In the **Task** pane, double-click **Report Clocks**. The Timing Analyzer reads your `.sdc`, applies the constraints (including `derive_pll_clocks`), and reports all the clocks.
5. From the Clocks Summary report, copy all the clock names that appear in the first column. The report lists the clock names in the correct format for recognition in the Timing Analyzer.
6. Open .sdc file and the paste the clock names into the file, one clock name per line.
7. Format the list of clock names list into the `set_clock_groups` command by cutting and pasting clock names into appropriate groups. Next, paste the following template into the `.sdc` file:

```
set_clock_groups -asynchronous -group { \
} \
 -group { \
} \
-group  { \
} \
-group { \
}
```

8. Cut and paste the clock names into groups to define their relationship, adding or removing groups, as necessary. Format the groups to make the code readable.

   Note: This command can be difficult to read on a single line. You can use the Tcl line continuation character "\" to make this more readable. Place a space after the last character, and then place the "\" character at the end of the line. Be careful not to include any spaces after the escape character. Otherwise, the space becomes the escape character, rather than the end-of-line character.

```
set_clock_groups -asynchronous \
    -group {adc_clk \
       the_adc_pll|altpll_component_autogenerated|pll|clk[0] \
       the_adc_pll|altpll_component_autogenerated|pll|clk[1] \
       the_adc_pll|altpll_component_autogenerated|pll|clk[2] \
    } \
    -group {sys_clk \
       the_system_pll|altpll_component_autogenerated|pll|clk[0] \
       the_system_pll|altpll_component_autogenerated|pll|clk[1] \
```

```
    } \
    -group {the_system_pll|altpll_component_autogenerated|pll|clk[2] \
    }
```

Note: The last group has a PLL output `system_pll|..|clk[2]` while the input clock and other PLL outputs are in different groups. If you use PLLs, and the input clock frequency does not relate to the frequency of the PLL's outputs, you must treat the PLLs asynchronously. Typically the outputs of a PLL are related and are in the same group, but this is not a requirement.

For designs with complex clocking, creating clock groups can be an iterative process. For example, a design with two DDR3 cores and high-speed transceivers can have thirty or more clocks. In such cases, you start by adding the clocks that you manually create. Timing Analyzer assumes that the clocks not appearing in the clock groups command relate to every clock and conservatively groups the known clocks. If the design still has failing paths between unrelated clock domains, you can add the new clock domains, as necessary. In this case, a large number of the clocks are not in the `set_clock_groups` command, because they are either cut in the `.sdc` file for the IP (such as the `.sdc` files that the DDR3 cores generate), or they connect only to related clock domains.

For many designs, that is all that's necessary to constrain the IP.

Related Information

Multicycle Paths on page 102

2.4.1.6. Accounting for Clock Effect Characteristics

The clocks you create with the Timing Analyzer are ideal clocks that do not account for any board effects. You can account for clock effect characteristics with clock latency and clock uncertainty constraints.

You can verify correct implementation of clock constraints by using **Report Clocks** (`report_clocks`) to generate clock timing reports. You can use **Check Timing** (`check_timing`) to report problems with a variety of timing constraints, such as missing clocks.

Related Information

- Report Clocks on page 135
- Check Timing on page 161

2.4.1.6.1. Set Clock Latency (set_clock_latency)

The **Set Clock Latency** (`set_clock_latency`) constraint allows you to specify additional delay (that is, latency) in a clock network. This delay value represents the external delay from a virtual (or ideal) clock through the longest **Late** (`-late`) or shortest **Early** (`-early`) path, with reference to the **Rise** (`-rise`) or **Fall** (`-fall`) of the clock transition.

When calculating setup analysis, the Timing Analyzer uses the late clock latency for the data arrival path and the early clock latency for the clock arrival path. . When calculating hold analysis, the Timing Analyzer uses the early clock latency for the data arrival time and the late clock latency for the clock arrival time.

There are two forms of clock latency: clock source latency, and clock network latency. Source latency is the propagation delay from the origin of the clock to the clock definition point (for example, a clock port). Network latency is the propagation delay from a clock definition point to a register's clock pin. The total latency at a register's clock pin is the sum of the source and network latencies in the clock path.

To specify source latency to any clock ports in your design, use the `set_clock_latency` command.

Note: The Timing Analyzer automatically computes network latencies. Therefore, you can only characterize source latency with the `set_clock_latency` command using the `-source` option.

Related Information

set_clock_latency Command, Quartus Prime Help

2.4.1.6.2. Clock Uncertainty

By default, the Timing Analyzer creates clocks that are ideal and have perfect edges. To mimic clock-level effects like jitter, you can add uncertainty to those clock edges. The Timing Analyzer automatically calculates appropriate setup and hold uncertainties and applies those uncertainties to all clock transfers in your design, even if you do not include the `derive_clock_uncertainty` command in your `.sdc` file. Setup and hold uncertainties are a critical part of constraining your design correctly.

The Timing Analyzer subtracts setup uncertainty from the data required time for each applicable path and adds the hold uncertainty to the data required time for each applicable path. This slightly reduces the setup and hold slack on each path.

The Timing Analyzer accounts for uncertainty clock effects for three types of clock-to-clock transfers: intraclock transfers, interclock transfers, and I/O interface clock transfers.

- Intraclock transfers occur when the register-to-register transfer takes place in the device and the source and destination clocks come from the same PLL output pin or clock port.
- Interclock transfers occur when a register-to-register transfer takes place in the core of the device and the source and destination clocks come from a different PLL output pin or clock port.
- I/O interface clock transfers occur when data transfers from an I/O port to the core of the device or from the core of the device to the I/O port.

To manually specify clock uncertainty, use the `set_clock_uncertainty` command. You can specify the uncertainty separately for setup and hold. You can also specify separate values for rising and falling clock transitions. You can override the value that the `derive_clock_uncertainty` command automatically applies.

The `derive_clock_uncertainty` command accounts for PLL clock jitter, if the clock jitter on the input to a PLL is within the input jitter specification for PLL's in the target device. If the input clock jitter for the PLL exceeds the specification, add additional uncertainty to your PLL output clocks to account for excess jitter with the `set_clock_uncertainty -add` command. Refer to the device handbook for your device for jitter specifications.

You can also use `set_clock_uncertainty -add` to account for peak-to-peak jitter from a board when the jitter exceeds the jitter specification for that device. In this case you add uncertainty to both setup and hold equal to 1/2 the jitter value:

```
set_clock_uncertainty –setup –to <clock name>  \
    -setup –add <p2p jitter/2>
```

```
set_clock_uncertainty –hold –enable_same_physical_edge –to <clock name> \
    –add <p2p jitter/2>
```

There is a complex set of precedence rules for how the Timing Analyzer applies values from `derive_clock_uncertainty` and `set_clock_uncertainty`, which depend on the order of commands and options in your `.sdc` files. The Help topics below contain complete descriptions of these rules. These precedence rules are easier to implement if you follow these recommendations:

- To assign your own clock uncertainty values to any clock transfers, put your `set_clock_uncertainty` exceptions after the `derive_clock_uncertainty` command in the `.sdc` file.
- When you use the `-add` option for `set_clock_uncertainty`, the value you specify is additive to the `derive_clock_uncertainty` value. If you do not specify `-add`, the value you specify replaces the value from `derive_clock_uncertainty`.

2.4.1.7. Constraining CDC Paths

It is essential to apply timing constraints to the multibit clock domain crossing (CDC) paths in your design. You can use the following constraints to constrain CDC paths:

Attention: As of Quartus Prime Pro Edition software version 21.3, the `set_false_path` constraint does not override the `set_max_skew` constraint. You can now apply the `set_false_path` and `set_max_skew` constraints on the same path without override.

Table 19. CDC Path Constraints

Constraints	Description
`set_false_path` `set_clock_groups -asynchronous`	Both constraints prevent the Compiler from optimizing slack between asynchronous domain crossings. `set_clock_groups` is the most aggressive constraint. • Clock-based false paths are less aggressive because these constraints only cut timing on the `from_clock` to `to_clock` order specified. • Clock-based false paths are unlike clock groups that cut the path in both directions. • Path-based false paths are the most specific constraint because they cut only on the specified `from` and `to` nodes.
`set_max_skew`	Sets a bound on the allowable skew between different bus bits.

continued...

Constraints	Description
	• Check the timing of your clock domain crossing by running the **Report Max Skew Summary** command. • The actual skew requirements depends on your design characteristics, and how you handle the clock domain crossing.
`set_net_delay -max` `set_data_delay`	Sets a bound on the allowable datapath delay on any bit of a bus transfer. • `set_net_delay` is for constraining individual clock edges and nets. Run the **Report Net Delay Summary** command to report data for this constraint. • `set_data_delay` is for constraining entire paths. Run the **Report Data Delay** command to report data for this constraint.

The following shows example constraints for a clock domain crossing between `data_a` in clock domain `clk_a`, and `data_b` in clock domain `clk_b`:

```
create_clock -name clk_a -period 4.000 [get_ports {clk_a}]
create_clock -name clk_b -period 4.500 [get_ports {clk_b}]
set_clock_groups -asynchronous -group [get_clocks {clk_a}] -group \
     [get_clocks {clk_b}]
set_net_delay -from [get_registers {data_a[*]}] -to [get_registers \
     {data_b[*]}] -max -get_value_from_clock_period \
     dst_clock_period -value_multiplier 0.8
set_max_skew -from [get_keepers {data_a[*]}] -to [get_keepers \
     {data_b[*]}] -get_skew_value_from_clock_period min_clock_period \
     -skew_value_multiplier 0.8
```

The following examples show applying `set_false_path` for a design that contains a DCFIFO block to avoid timing failures in the synchronization registers. These examples are for constraining single-bit synchronizer CDC paths:

- For paths crossing from the write into the read domain, apply a false path assignment between registers `delayed_wrptr_g` and `rs_dgwp`:

```
set_false_path -from [get_registers {*dcfifo*delayed_wrptr_g[*]}] \
-to [get_registers {*dcfifo*rs_dgwp*}]
```

- For paths crossing from the read into the write domain, apply a false path assignment between registers `rdptr_g` and `ws_dgrp`:

```
set_false_path -from [get_registers {*dcfifo*rdptr_g[*]}] \
-to [get_registers {*dcfifo*ws_dgrp*}]
```

You can verify correct implementation of clock constraints in the following CDC related reports:

Related Information

- Report Clock Transfers on page 137
- Report CDC Viewer on page 139
- Report Asynchronous CDC on page 142

2.4.2. I/O Constraints

The Timing Analyzer reviews setup and hold relationships for designs in which an external source interacts with a register internal to the design. The Timing Analyzer supports input and output external delay modeling with the `set_input_delay` and `set_output_delay` commands. You can specify the clock and minimum and maximum arrival times relative to the clock.

Specify internal and external timing requirements before you fully analyze a design. With external timing requirements specified, the Timing Analyzer verifies the I/O interface, or periphery of the device, against any system specification.

You can use the **Check Timing** (`check_timing`) command to report problems with a variety of timing constraints, such as the number of input ports that are not clocks that have no input delay constraint.

Related Information

Check Timing on page 161

2.4.2.1. Input Constraints (set_input_delay)

Input constraints specify delays for all external signals feeding the FPGA. Specify input requirements for all input ports in your design.

```
set_input_delay -clock { clock } -clock_fall -fall -max 20 foo
```

Use the **Set Input Delay** (`set_input_delay`) constraint to specify external input delay requirements. Specify the **Clock name** (`-clock`) to reference the virtual or actual clock. You can specify a clock to allow the Timing Analyzer to correctly derive clock uncertainties for interclock and intraclock transfers. The clock defines the launching clock for the input port. The Timing Analyzer automatically determines the latching clock inside the device that captures the input data, because all clocks in the device are defined.

Figure 94. Input Delay Diagram

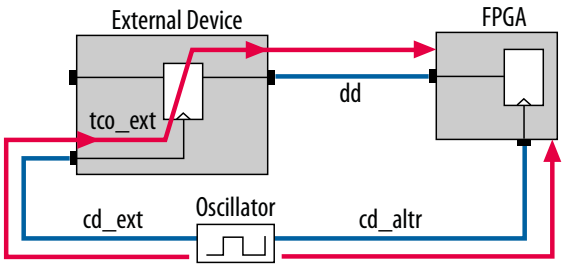


Figure 95. Input Delay Calculation

$$\text{input delay}_{MAX} = (\text{cd\_ext}_{MAX} - \text{cd\_altr}_{MIN}) + \text{tco\_ext}_{MAX} + \text{dd}_{MAX}$$
$$\text{input delay}_{MIN} = (\text{cd\_ext}_{MIN} - \text{cd\_altr}_{MAX}) + \text{tco\_ext}_{MIN} + \text{dd}_{MIN}$$

If your design includes partition boundary ports, you can use the `-blackbox` option with `set_input_delay` to assign input delays. The `-blackbox` option creates a new keeper timing node with the same name as the boundary port. This new node permits the propagation of timing paths through the original boundary port and acts as a `set_input_delay` constraint. The new keeper timing nodes display when you use the `get_keepers` command. You can remove these black box constraints with `remove_input_delay -blackbox`.

You can use the **Check Timing** (`check_timing`) command to report problems with a variety of timing constraints, such as the number of input ports that are not clocks that have no input delay constraint.

Related Information

Check Timing on page 161

2.4.2.2. Output Constraints (set_output_delay)

Output constraints specify all external delays from the device for all output ports in your design.

```
set_output_delay -clock { clock } -clock_fall -rise -max 2 foo
```

Use the **Set Output Delay** (`set_output_delay`) constraint to specify external output delay requirements. Specify the **Clock name** (`-clock`) to reference the virtual or actual clock. When specifying a clock, the clock defines the latching clock for the output port. The Timing Analyzer automatically determines the launching clock inside the device that launches the output data, because all clocks in the device are defined. The following figure is an example of an output delay referencing a virtual clock.

Figure 96. Output Delay Diagram

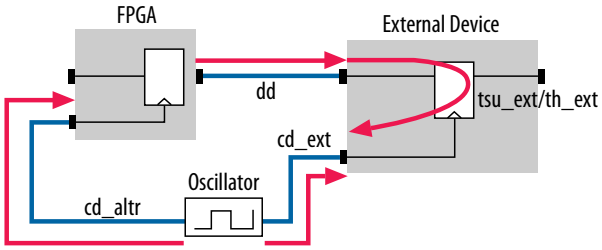


Figure 97. Output Delay Calculation

$$\text{output delay}_{MAX} = \text{dd}_{MAX} + \text{tsu\_ext} + (\text{cd\_altr}_{MAX} - \text{cd\_ext}_{MIN})$$
$$\text{output delay}_{MIN} = (\text{dd}_{MIN} - \text{th\_ext} + (\text{cd\_altr}_{MIN} - \text{cd\_ext}_{MAX}))$$

If your design includes partition boundary ports, you can use the `-blackbox` option with `set_ouput_delay` to assign output delays. The `-blackbox` option creates a new keeper timing node with the same name as the boundary port. This new node permits the propagation of timing paths through the original boundary port and acts as a `set_output_delay` constraint. The new keeper timing nodes display when you use the `get_keepers` command.

You can remove blackbox constraints with `remove_output_delay -blackbox`.

You can use the **Check Timing** (`check_timing`) command to report problems with a variety of timing constraints, such as the number of output ports that have no output delay constraint.

Related Information

- Check Timing on page 161

- set_input_delay Command, Quartus Prime Help
- set_output_delay Command, Quartus Prime Help

2.4.3. Delay and Skew Constraints

You can specify skew and delays to model external device timing and board timing parameters.

2.4.3.1. Advanced I/O Timing and Board Trace Model Delay

The Timing Analyzer can use advanced I/O timing and board trace model constraints to model I/O buffer delays in your design.

If you change any advanced I/O timing settings or board trace model assignments, recompile your design before you analyze timing, or use the `-force_dat` option to force delay annotation when you create a timing netlist.

Example 6. Forcing Delay Annotation

```
create_timing_netlist -force_dat
```

2.4.3.2. Maximum Skew (set_max_skew)

The **Set Max Skew** (`set_max_skew`) constraint specifies the maximum allowable skew between the sets of registers or ports you specify. In order to constrain skew across multiple paths, you must constrain all such paths within a single `set_max_skew` constraint.

```
set_max_skew -from_clock { clock } -to_clock { * } -from foo -to blat 2
```

The `set_max_delay`, `set_min_delay`, `set_multicycle_path`, and `set_false_path` constraints do not affect the `set_max_skew` timing constraint. However, the `set_clock_groups` constraint does impact the `set_max_skew` constraint.

Note: Exclusive clock groups (set with `set_clock_groups -exclusive`) override `set_max_skew` constraints.

The Timing Analyzer does not compare two paths for skew if their clocks are exclusive to each other. However, the Timing Analyzer does analyze for skew paths whose clocks are asynchronous.

Table 20. set_max_skew Options

Arguments	Description
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-fall_from_clock <names>`	Valid source clocks (Tcl matches string patterns). Analysis only considers paths from falling clock edges.
`-fall_to_clock <names>`	Valid destination clocks (Tcl matches string patterns). Analysis only considers paths from falling clock edges.
`-from <names>`(2)	Valid sources (Tcl matches string patterns).

continued...

Arguments	Description
`-from_clock <names>`	Valid source clocks (Tcl matches string patterns).
`-get_skew_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period>`	Option to interpret skew constraint as a multiple of the clock period.
`-rise_from_clock <names>`	Valid source clocks (Tcl matches string patterns). Analysis only considers paths from rising clock edges.
`-rise_to_clock <names>`	Valid destination clocks (Tcl matches string patterns). Analysis only considers paths to rising clock edges.
`-skew_value_multiplier <multiplier>`	Value by which the clock period multiplies to compute skew requirement.
`-to <names>`[2]	Valid destinations (Tcl matches string patterns)
`-to_clock <names>`	Valid destination clocks (Tcl matches string patterns).
<skew>	The value of the skew you require.

Applying maximum skew constraints between clocks applies the constraint from all register or ports driven by the clock you specify (with the `-from` option) to all registers or ports driven by the clock you specify (with the `-to` option).

Maximum skew analysis can include data arrival times, clock arrival times, register micro parameters, clock uncertainty, on-die variation, and clock pessimism removal. Among these, the Fitter only disables clock pessimism removal by default.

Use `-get_skew_value_from_clock_period` to set the skew as a multiple of the launching or latching clock period, or whichever of the two has a smaller period. If you use this option, set `-skew_value_multiplier`, and you may not set the positional skew option. If more than one clock clocks the set of skew paths, Timing Analyzer uses the clock with smallest period to compute the skew constraint.

Click **Report Max Skew...** (`report_max_skew`) to view the max skew analysis. Since skew occurs between two or more paths, no results display if the `-from`/`-from_clock` and `-to`/`-to_clock` filters satisfy less than two paths.

Related Information

- Timing Exception Precedence on page 99
- report_max_skew Command, Quartus Prime Help

2.4.3.3. Net Delay (set_net_delay)

Use the `set_net_delay` command to set the net delays and perform minimum or maximum timing analysis across nets. A net delay constraint is invalid if there is a combinational cell between the From and To nodes.

(2) Legal values for the -from and -to options are collections of clocks, registers, ports, pins, cells or partitions in a design.

The `-from` and `-to` options can be string patterns or pin, port, register, or net collections. When you use pin or net collection, include output pins or nets in the collection.

```
set_net_delay -from reg_a -to reg_c -max 20
```

Table 21. set_net_delay Options

Arguments	Description
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-from <names>`(3)	Valid source pins, ports, registers or nets (Tcl matches string patterns).
`-get_value_from_clock_period <src_clock_period|dst_clock_period| min_clock_period|max_clock_period>`	Option to interpret net delay constraint as a multiple of the clock period.
`-max`	Specifies maximum delay.
`-min`	Specifies minimum delay.
`-to <names>`(4)	Valid destination pins, ports, registers or nets (Tcl matches string patterns).
`-value_multiplier <multiplier>`	Value by which the clock period multiplies to compute net delay requirement.
`<delay>`	Delay value.

If you use the `-min` option, the Timing Analyzer calculates slack by determining the minimum delay on the edge. If you use `-max` option, the Timing Analyzer calculates slack by determining the maximum edge delay.

Use `-get_value_from_clock_period` to set the net delay requirement as a multiple of the launching or latching clock period, or whichever of the two has a smaller or larger period. If you use this option, you must not set the positional delay option. If more than one clock clocks the set of nets, the Timing Analyzer uses the net with the smallest period to compute the constraint for a `-max` constraint, and the largest period for a `-min` constraint. If no clocks are clocking the endpoints of the net (that is, if the endpoints of the nets are not registers or constraint ports), the Timing Analyzer ignores the net delay constraint.

Related Information

- Timing Exception Precedence on page 99
- report_net_delay Command, Quartus Prime Help

(3) If option is a wildcard ("*") character, all the output pins and registers on timing netlist become valid source points.

(4) If no option, or if option is a wildcard ("*") character, all the output pins and registers on timing netlist become valid destination points.

2.4.4. Timing Exception Constraints

Timing exception assignments allow you to modify (or provide exception to) the default timing analysis behavior to account for your specific design conditions.

You can specify the following timing constraints that modify the default timing analysis behavior:

- Set False Path
- Set Multicycle Path
- Set Minimum Delay
- Set Maximum Delay

Verify correct implementation of timing exception assignments by using the **Report Exceptions** (`report_exceptions`) command to report all exceptions to default timing analysis conditions.

Related Information

Report Exceptions on page 159

2.4.4.1. Timing Exception Precedence

If the same clock or node names occur in multiple timing exceptions, the Timing Analyzer observes the following order of timing exception precedence:

1. **Set False Path** (`set_false_path`) is the first priority. False paths and clock groups have identical priority, except when you use the `-latency_insensitive` or `-no_synchronizer` options with a false path exception. With either option, the false path has priority over a clock group.
2. **Set Clock Groups** (`set_clock_groups`) is the second priority.
3. **Set Minimum Delay** (`set_min_delay`) and **Set Maximum Delay** (`set_max_delay`) are the third priority.
4. **Set Multicycle Path** (`set_multicycle_path`) is the fourth priority.

The false path timing exception has the highest precedence. Within each category, assignments to individual nodes have precedence over assignments to clocks. For exceptions of the same type:

1. `-from <node>` is the first priority.
2. `-to <node>` is the second priority.
3. `-thru <node>` is the third priority.
4. `-from <clock>` is the fourth priority.
5. `-to <clock>` is the fifth priority.

An asterisk wildcard (*) for any of these options applies the same precedence as not specifying the option at all. For example, `-from a -to *` is treated identically to `-from a` with regards precedence.

Precedence example

1. `set_max_delay 1 -from x -to y`
2. `set_max_delay 2 -from x`
3. `set_max_delay 3 -to y`

The first exception has higher priority than either of the other two, since the first exception specifies a `-from` (while #3 does not) and specifies a `-to` (while #2 does not). In the absence of the first exception, the second exception has higher priority than the third, since the second exception specifies a `-from`, which the third does not. Finally, the remaining order of precedence for additional exceptions is order-dependent, such that the assignments most recently created overwrite, or partially overwrite, earlier assignments.

The `set_net_delay`, `set_max_skew`, and `set_data_delay` constraints analyze independently of minimum or maximum delays, or multicycle path constraints.

- The `set_net_delay` exception applies regardless of the existence of a `set_false_path` exception, or `set_clock_groups` exception, or other path-based constraint or exception. It is a net-based exception, and net-based and path-based exceptions are applied independently of each other.
- The `set_max_skew` exception applies on paths cut by an asynchronous clock group, and regardless of any `set_false_path` exception. Exclusive clock groups override max skew exceptions, because paths between exclusive clocks are entirely inactive and should not be analyzed for timing or skew requirements. This precedence allows you to define more targeted constraints on asynchronous CDC bus transfers.
- The `set_data_delay` exception specifies a maximum datapath delay exception for a given path. Exclusive clock groups override data delay exceptions, because paths between exclusive clocks are entirely inactive and should not be analyzed for timing or data delay requirements. Asynchronous clock groups do not override data delay exceptions. False path exceptions override data delay exceptions in the Quartus Prime Pro software version 21.2 and earlier. Beginning in version 21.3, false path exceptions do not override data delay exceptions. This change in precedence allows you to write more targeted constraints on asynchronous CDC bus transfers.

Verify correct implementation of timing exception assignments by using the **Report Exceptions** (`report_exceptions`) command to report all exceptions to default timing analysis conditions.

Related Information

- False Paths (set_false_path) on page 101
- Creating Clock Groups (set_clock_groups) on page 86
- Constraining CDC Paths on page 92
- Creating Clock Groups (set_clock_groups) on page 86
- Maximum Skew (set_max_skew) on page 96
- Minimum and Maximum Delays on page 101
- Net Delay (set_net_delay) on page 97
- Report Data Delay on page 134

- Report Exceptions on page 159

2.4.4.2. False Paths (set_false_path)

The **Set False Path** (`set_false_path`) constraint allows you to exclude a path from timing analysis, such as test logic or any other path not relevant to the circuit's operation. You can specify the source (`-from`), common through elements (`-thru`), and destination (`-to`) elements of that path.

The following SDC command makes false path exceptions from all registers starting with A, to all registers starting with B:

```
set_false_path -from [get_pins A*] -to [get_pins B*]
```

You can specify either a point-to-point or clock-to-clock path as a false path. A false path's `-from` and `-to` targets can be either nodes or clocks. However, the `-thru` targets can only be combinational nodes. For example, you can specify a false path for a static configuration register that writes once during power-up initialization, but does not change state again.

Although signals from static configuration registers often cross clock domains, you may not want to make false path exceptions to a clock-to-clock path, because some data may transfer across clock domains. However, you can selectively make false path exceptions from the static configuration register to all endpoints.

The Timing Analyzer assumes all clocks are related unless you specify otherwise. Use clock groups to more efficiently make false path exceptions between clocks, rather than writing multiple `set_false_path` exceptions between each clock transfer you want to eliminate.

Verify correct implementation of timing exception assignments by using the **Report Exceptions** (`report_exceptions`) command to report all exceptions to default timing analysis conditions.

Related Information

- Timing Exception Precedence on page 99
- Creating Clock Groups (set_clock_groups) on page 86
- Constraining CDC Paths on page 92
- Report Exceptions on page 159
- set_false_path Command, Quartus Prime Help

2.4.4.3. Minimum and Maximum Delays

To specify an absolute minimum or maximum delay for a path, use the **Set Minimum Delay** (`set_min_delay`) or the **Set Maximum Delay** (`set_max_delay`) constraints, respectively. Specifying minimum and maximum delay directly overwrites existing setup and hold relationships with the minimum and maximum values.

Use the `set_max_delay` and `set_min_delay` constraints for asynchronous signals that do not have a specific clock relationship in your design, but require a minimum and maximum path delay. You can create minimum and maximum delay exceptions for port-to-port paths through the device without a register stage in the path. If you

use minimum and maximum delay exceptions to constrain the path delay, specify both the minimum and maximum delay of the path; do not constrain only the minimum or maximum value.

If the source or destination node is clocked, the Timing Analyzer takes into account the clock paths, allowing more or less delay on the data path. If the source or destination node has an input or output delay, the minimum or maximum delay check also includes that delay.

If you specify a minimum or maximum delay between timing nodes, the delay applies only to the path between the two nodes. If you specify a minimum or maximum delay for a clock, the delay applies to all paths where the clock clocks the source node or destination node.

You can create a minimum or maximum delay exception for an output port that does not have an output delay constraint. You cannot report timing for the paths that relate to the output port; however, the Timing Analyzer reports any slack for the path in the setup summary and hold summary reports. Because there is no clock that relates to the output port, the Timing Analyzer reports no clock for timing paths of the output port.

Note: To report timing with clock filters for output paths with minimum and maximum delay constraints, you can set the output delay for the output port with a value of zero. You can use an existing clock from the design or a virtual clock as the clock reference.

Verify correct implementation of timing exception assignments by using the **Report Exceptions** (`report_exceptions`) command to report all exceptions to default timing analysis conditions.

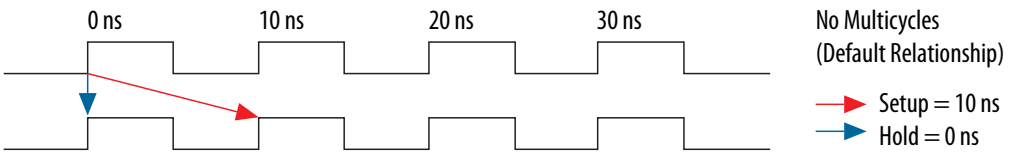
Related Information

- Timing Exception Precedence on page 99
- Report Exceptions on page 159
- set_max_delay Command, Quartus Prime Help
- set_min_delay Command, Quartus Prime Help

2.4.4.4. Multicycle Paths

By default, the Timing Analyzer performs a single-cycle analysis, which is the most restrictive type of analysis. When analyzing a path without a multicycle constraint, the Timing Analyzer determines the setup launch and latch edge times by identifying the closest two active edges in the respective waveforms.

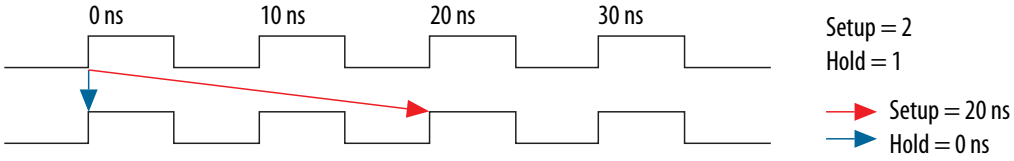
Figure 98. Default Setup and Hold Relationship (No Multicycle)



For hold time analysis, the timing analyzer analyzes the path for two timing conditions for every possible setup relationship, not just the worst-case setup relationship. Therefore, the hold launch and latch times can be unrelated to the setup launch and latch edges.

A multicycle constraint adjusts this default setup or hold relationship by the number of clock cycles you specify, based on the source (`-start`) or destination (`-end`) clock. A setup multicycle constraint of 2 extends the worst-case setup latch edge by one destination clock period. If you do not specify `-start` and `-end` values, the default constraint is `-end`.

Figure 99. Setup and Hold Relationship with Multicycle = 2



Hold multicycle constraints derive from the default hold position (the default value is 0). An end hold multicycle constraint of 1 effectively subtracts one destination clock period from the default hold latch edge.

When the objects are timing nodes, the multicycle constraint only applies to the path between the two nodes. When an object is a clock, the multicycle constraint applies to all paths where the source node (`-from`) or destination node (`-to`) is clocked by the clock. When you adjust a setup relationship with a multicycle constraint, the default hold relationship adjusts automatically.

You can use timing constraints to modify either the launch or latch edge times that the Timing Analyzer uses to determine a setup relationship or hold relationship.

Table 22. Multicycle Constraints

Command	Modification
`set_multicycle_path -setup -end <value>`	Latch edge time of the setup relationship.
`set_multicycle_path -setup -start<value>`	Launch edge time of the setup relationship.
`set_multicycle_path -hold -end <value>`	Latch edge time of the hold relationship.
`set_multicycle_path -hold -start <value>`	Launch edge time of the hold relationship.

Verify correct implementation of timing exception assignments by using the **Report Exceptions** (`report_exceptions`) command to report all exceptions to default timing analysis conditions.

Related Information

Report Exceptions on page 159

2.4.4.4.1. Common Multicycle Applications

Multicycle exceptions adjust the timing requirements for a register-to-register path, allowing the Fitter to optimally place and route a design. Two common multicycle applications are relaxing setup to allow a slower data transfer rate, and altering the setup to account for a phase shift.

2.4.4.4.2. Relaxing Setup with Multicycle (set_multicyle_path)

You can use a multicycle exception when the data transfer rate is slower than the clock cycle. Relaxing the setup relationship increases the window when timing analysis accepts data as valid.

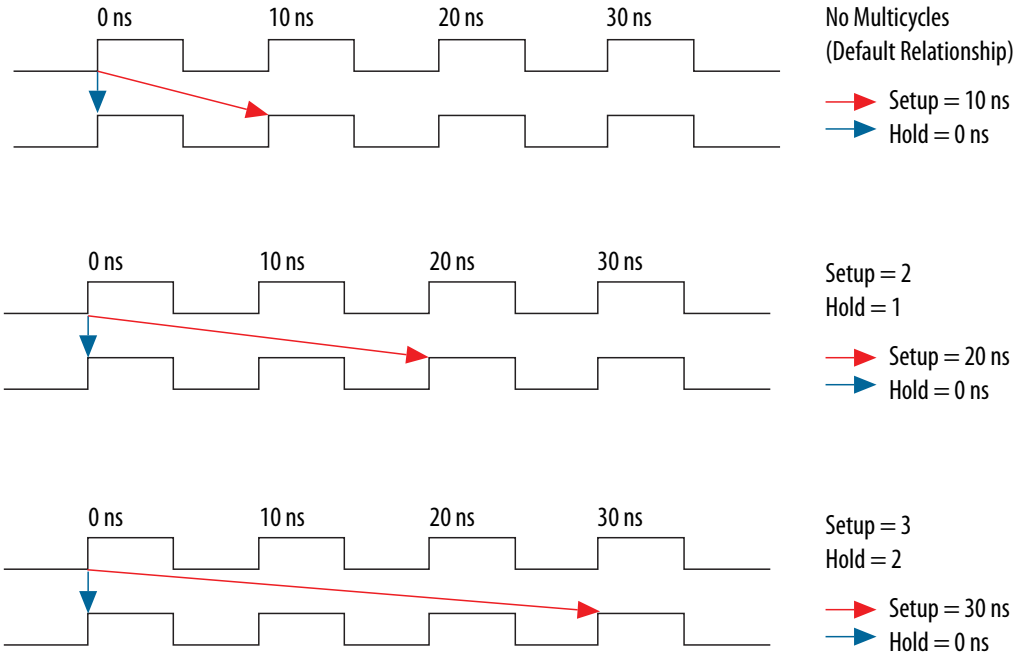
In the following example, the source clock has a period of 10 ns, but the clock enable signal controls a group of latching registers, so the registers only enable every other cycle. The 10 ns clock feeds registers , so the Timing Analyzer reports a setup of 10 ns and a hold of 0 ns. However, the data is transferring every other cycle, so the Timing Analyzer must analyze the relationships as if the clock is operating at 20 ns. The result is a setup of 20 ns, while the hold remains 0 ns, thus extending the window for data recognition.

The following pair of multicycle assignments relax the setup relationship by specifying the `-setup` value of N and the `-hold` value as N-1. You must specify the hold relationship with a `-hold` assignment to prevent a positive hold requirement.

Constraint to Relax Setup and Maintain Hold

```
set_multicycle_path -setup -from src_reg* -to dst_reg* 2
set_multicycle_path -hold -from src_reg* -to dst_reg* 1
```

Figure 100. Multicycle Setup Relationships



You can extend this pattern to create larger setup relationships to ease timing closure requirements. A common use for this exception is when writing to asynchronous RAM across an I/O interface. The delay between address, data, and a write enable may be several cycles. A multicycle exception to I/O ports allows extra time for the address and data to resolve before the enable occurs.

The following constraint relaxes the setup by three cycles:

Three Cycle I/O Interface Constraint

```
set_multicycle_path -setup -to [get_ports {SRAM_ADD[*] SRAM_DATA[*]} 3
set_multicycle_path -hold -to [get_ports {SRAM_ADD[*] SRAM_DATA[*]} 2
```

2.4.4.4.3. Accounting for a Phase Shift (-phase)

In the following example, the design contains a PLL that performs a phase-shift on a clock whose domain exchanges data with domains that do not experience the phase shift. This occurs when the destination clock phase-shifts forward, and the source clock does not shift. The default setup relationship becomes that phase-shift, thus shifting the window when data is valid.

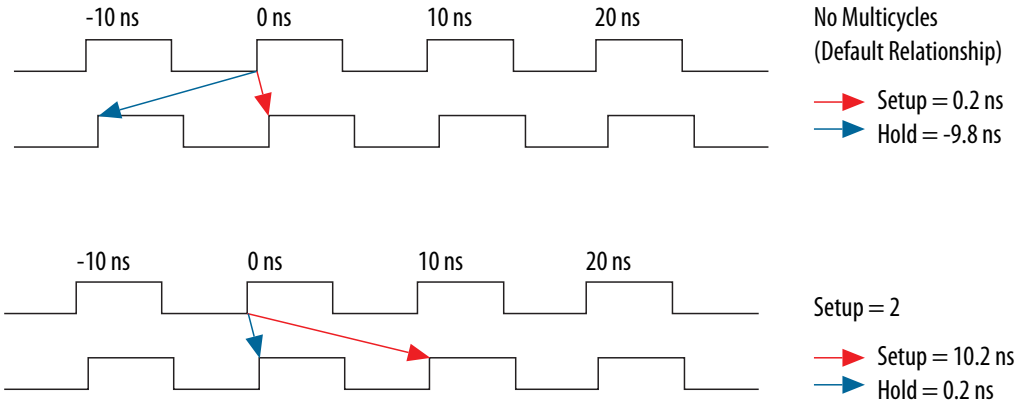
For example, the following code phase-shifts one output of a PLL forward by a small amount, in this case 0.2 ns.

Cross Domain Phase-Shift

```
create_generated_clock -source pll|inclk[0] -name pll|clk[0] pll|clk[0]
create_generated_clock -source pll|inclk[0] -name pll|clk[1] -phase 30 pll|clk[1]
```

The default setup relationship for this phase-shift is 0.2 ns, shown in Figure A, creating a scenario where the hold relationship is negative, which makes achieving timing closure nearly impossible.

Figure 101. Phase-Shifted Setup and Hold



The following constraint allows the data to transfer to the following edge:

```
set_multicycle_path -setup -from [get_clocks clk_a] -to [get_clocks clk_b] 2
```

The hold relationship derives from the setup relationship, making a multicycle hold constraint unnecessary.

Related Information

- Same Frequency Clocks with Destination Clock Offset on page 114
- set_multicycle_path Command, Quartus Prime Help

2.4.4.5. Multicycle Exception Examples

The examples in this section illustrate how the multicycle exceptions affect the default setup and hold analysis in the Timing Analyzer. The multicycle exceptions apply to a simple register-to-register circuit. Both the source and destination clocks are set to 10 ns.

Verify correct implementation of timing exception assignments by using the **Report Exceptions** (`report_exceptions`) command to report all exceptions to default timing analysis conditions.

Related Information

Report Exceptions on page 159

2.4.4.5.1. Default Multicycle Analysis

By default, the Timing Analyzer performs a single-cycle analysis to determine the setup and hold checks. Also, by default, the Timing Analyzer sets the end multicycle setup assignment value to one and the end multicycle hold assignment value to zero.

The source and the destination timing waveform for the source register and destination register, respectively where HC1 and HC2 are hold checks 1 and 2 and SC is the setup check.

Figure 102. Default Timing Diagram

The timing waveforms show the source and destination registers of a data transfer. HC1 and HC2 are the hold checks that Timing Analyzer performs. SC is the setup check that Timing Analyzer performs.

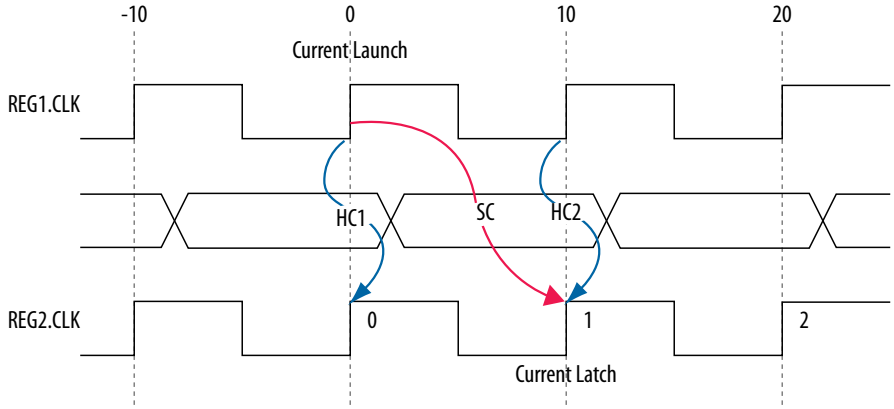
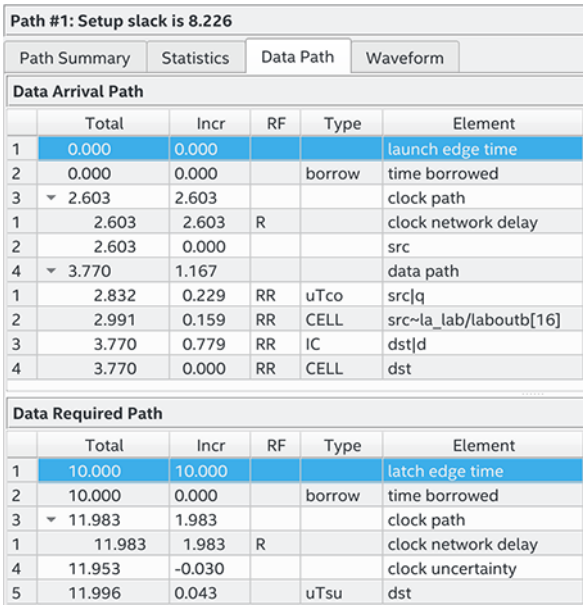


Figure 103. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 10\text{ ns} - 0\text{ ns} \\ &= 10\text{ ns} \end{aligned}$$

The most restrictive default single-cycle setup relationship, with an implied end multicycle setup assignment of one, is 10 ns.

Figure 104. Default Setup Report



Path #1: Setup slack is 8.226

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	2.603	2.603			clock path
1	2.603	2.603	R		clock network delay
2	2.603	0.000			src
4	3.770	1.167			data path
1	2.832	0.229	RR	uTco	src|q
2	2.991	0.159	RR	CELL	src~la_lab/laboutb[16]
3	3.770	0.779	RR	IC	dst|d
4	3.770	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	10.000	10.000			latch edge time
2	10.000	0.000		borrow	time borrowed
3	11.983	1.983			clock path
1	11.983	1.983	R		clock network delay
4	11.953	-0.030			clock uncertainty
5	11.996	0.043		uTsu	dst

Figure 105. Hold Check Calculation

The figure shows the setup timing report with the launch and latch edge times highlighted.

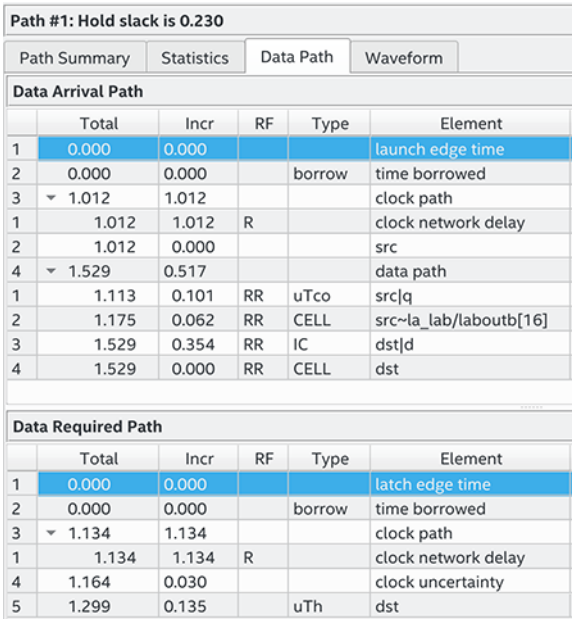
$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 0\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 10\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

The most restrictive default single-cycle hold relationship, with an implied end multicycle hold assignment of zero, is 0ns.

Figure 106. Default Hold Report

The figure shows the hold timing report with the launch and latch edge times highlighted.



Path #1: Hold slack is 0.230

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	1.012	1.012			clock path
1	1.012	1.012	R		clock network delay
2	1.012	0.000			src
4	1.529	0.517			data path
1	1.113	0.101	RR	uTco	src|q
2	1.175	0.062	RR	CELL	src~la_lab/laboutb[16]
3	1.529	0.354	RR	IC	dst|d
4	1.529	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			latch edge time
2	0.000	0.000		borrow	time borrowed
3	1.134	1.134			clock path
1	1.134	1.134	R		clock network delay
4	1.164	0.030			clock uncertainty
5	1.299	0.135		uTh	dst

2.4.4.5.2. End Multicycle Setup = 2 and End Multicycle Hold = 0

In this example, the end multicycle setup assignment value is two, and the end multicycle hold assignment value is zero.

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
    -setup -end 2
```

Note: The Timing Analyzer does not require an end multicycle hold value because the default end multicycle hold value is zero.

In this example, the setup relationship relaxes by a full clock period by moving the latch edge to the next latch edge. The hold analysis is does not change from the default settings. The following shows the setup timing diagram for the analysis that the Timing Analyzer performs. The latch edge is a clock cycle later than in the default single-cycle analysis.

Figure 107. Setup Timing Diagram

The figure shows the setup timing diagram for the analysis that the Timing Analyzer performs. Without the multicycle constraint the latching edge is edge 1. However, with the multicycle constraint the latching edge is edge 2.

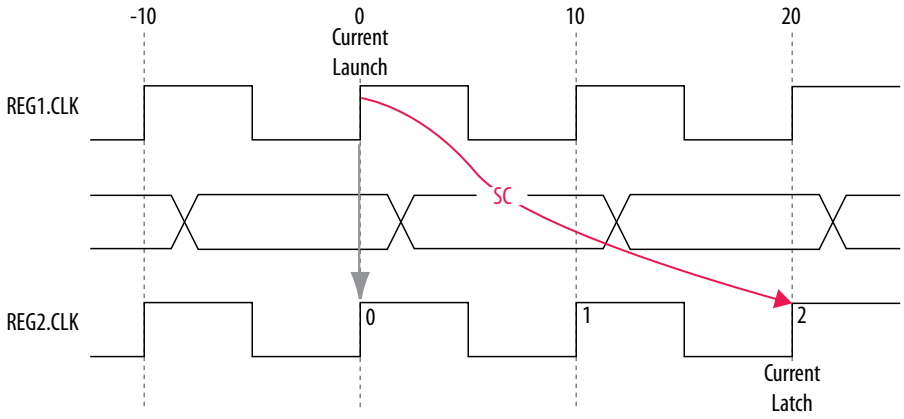


Figure 108. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 20\text{ ns} - 0\text{ ns} \\ &= 20\text{ ns} \end{aligned}$$

The most restrictive setup relationship with an end multicycle setup assignment of two is 20 ns. The following shows the setup report in the Timing Analyzer and highlights the launch and latch edges.

Figure 109. Setup Report with Setup Multicycle Exception

Path #1: Setup slack is 18.226

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	2.603	2.603			clock path
1	2.603	2.603	R		clock network delay
2	2.603	0.000			src
4	3.770	1.167			data path
1	2.832	0.229	RR	uTco	src|q
2	2.991	0.159	RR	CELL	src~la_lab/laboutb[16]
3	3.770	0.779	RR	IC	dst|d
4	3.770	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	20.000	20.000			latch edge time
2	20.000	0.000		borrow	time borrowed
3	21.983	1.983			clock path
1	21.983	1.983	R		clock network delay
4	21.953	-0.030			clock uncertainty
5	21.996	0.043		uTsu	dst

Because the multicycle hold latch and launch edges are the same as the results of hold analysis with the default settings, the multicycle hold analysis in this example is equivalent to the single-cycle hold analysis. The hold checks are relative to the setup check. Normally, the Timing Analyzer performs hold checks on every possible setup check, not only on the most restrictive setup check edges.

Figure 110. Hold Timing Diagram

The figure shows the hold latching edges are now at 10 and 20 ns, instead of 0 and 10 ns.

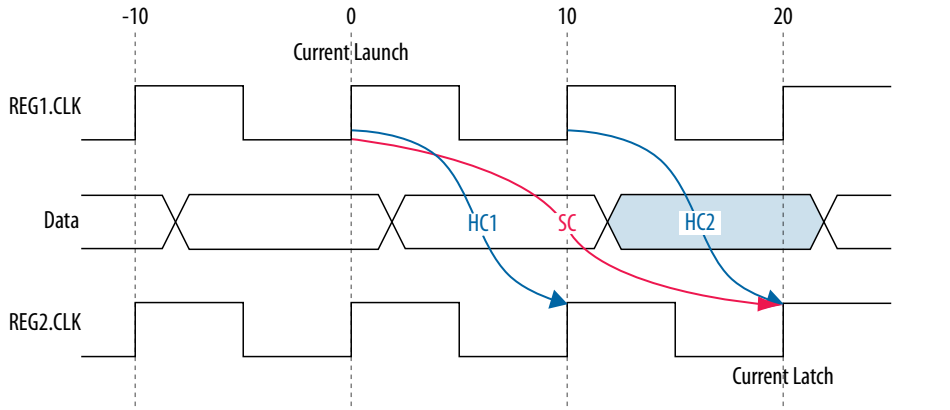
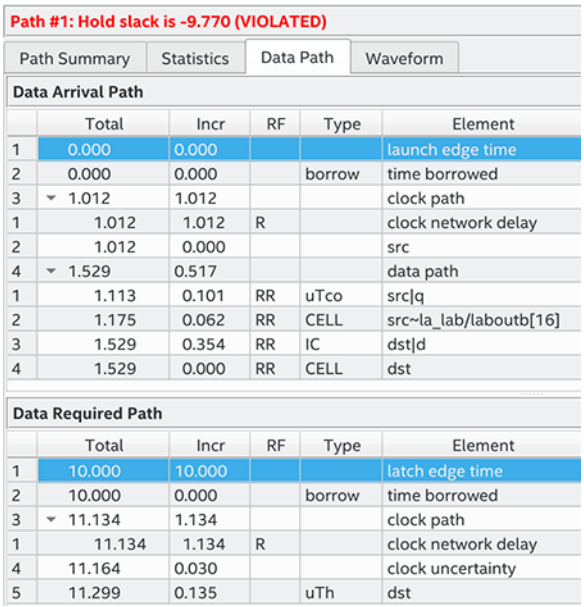


Figure 111. Hold Report with Setup Multicycle Exception



Path #1: Hold slack is -9.770 (VIOLATED)

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	1.012	1.012			clock path
1	1.012	1.012	R		clock network delay
2	1.012	0.000			src
4	1.529	0.517			data path
1	1.113	0.101	RR	uTco	src|q
2	1.175	0.062	RR	CELL	src~la_lab/laboutb[16]
3	1.529	0.354	RR	IC	dst|d
4	1.529	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	10.000	10.000			latch edge time
2	10.000	0.000		borrow	time borrowed
3	11.134	1.134			clock path
1	11.134	1.134	R		clock network delay
4	11.164	0.030			clock uncertainty
5	11.299	0.135		uTh	dst

Figure 112. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 10\ \text{ns} \\ &= -10\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 20\ \text{ns} \\ &= -10\ \text{ns} \end{aligned}$$

2.4.4.5.3. End Multicycle Setup = 2 and End Multicycle Hold = 1

In this example, the end multicycle setup assignment value is two, and the end multicycle hold assignment value is one.

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
    -setup -end 2
set_multicycle_path -from [get_clocks clk_src] -to
\[get_clocks clk_dst] -hold -end 1
```

In this example, the setup relationship relaxes by one clock period by moving the latching edge to the right of the default latching edge by 1 clock period. The hold relationship relaxes by one clock period by moving the latch edges to the left of the default latching edges by one.

The following shows the setup timing diagram for the analysis that the Timing Analyzer performs:

Figure 113. Setup Timing Diagram

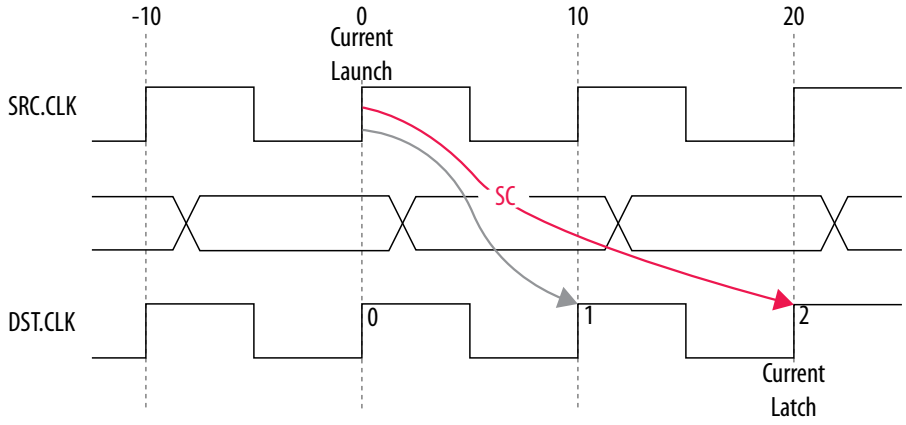


Figure 114. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 20\ \text{ns} - 0\ \text{ns} \\ &= 20\ \text{ns} \end{aligned}$$

The most restrictive hold relationship with an end multicycle setup assignment value of two is 20 ns.

The following shows the setup report for this example in the Timing Analyzer and highlights the launch and latch edges.

Figure 115. Setup Report with Setup and Hold Multicycle Exception

Path #1: Setup slack is 18.226

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	2.603	2.603			clock path
1	2.603	2.603	R		clock network delay
2	2.603	0.000			src
4	3.770	1.167			data path
1	2.832	0.229	RR	uTco	src|q
2	2.991	0.159	RR	CELL	src~la_lab/laboutb[16]
3	3.770	0.779	RR	IC	dst|d
4	3.770	0.000	RR	CELL	dst

Data Required Path

	Total	Incr	RF	Type	Element
1	20.000	20.000			latch edge time
2	20.000	0.000		borrow	time borrowed
3	21.983	1.983			clock path
1	21.983	1.983	R		clock network delay
4	21.953	-0.030			clock uncertainty
5	21.996	0.043		uTsu	dst

The following shows the timing diagram for the hold checks for this example. The hold checks are relative to the setup check.

Figure 116. Hold Timing Diagram

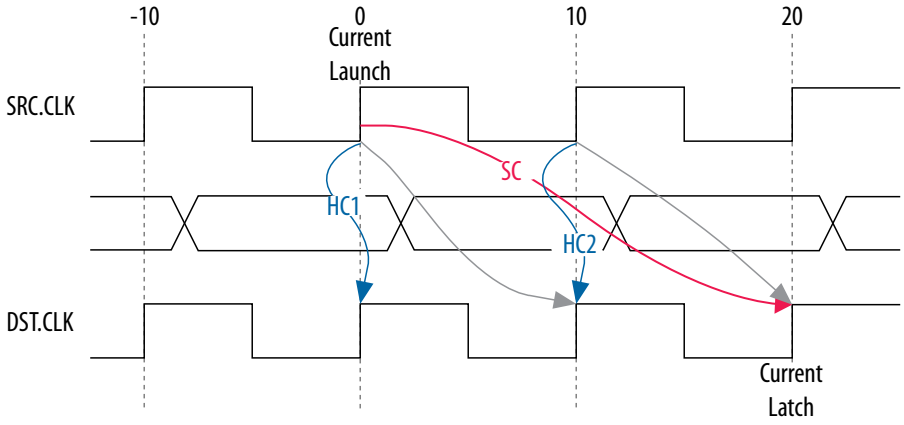


Figure 117. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 0\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 10\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

The most restrictive hold relationship with an end multicycle setup assignment value of two and an end multicycle hold assignment value of one is 0 ns.

The following shows the hold report for this example in the Timing Analyzer and highlights the launch and latch edges.

Figure 118. Hold Report with Setup and Hold Multicycle Exception

Path #1: Hold slack is 0.230

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Element
1	0.000	0.000			launch edge time
2	0.000	0.000		borrow	time borrowed
3	1.012	1.012			clock path
1	1.012	1.012	R		clock network delay
2	1.012	0.000			src
4	1.529	0.517			data path
1	1.113	0.101	RR	uTco	src|q
2	1.175	0.062	RR	CELL	src~la_lab/laboutb[16]
3	1.529	0.354	RR	IC	dst|d
4	1.529	0.000	RR	CELL	dst

Data Required Path

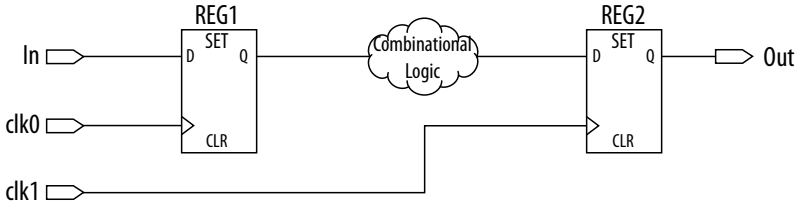
	Total	Incr	RF	Type	Element
1	0.000	0.000			latch edge time
2	0.000	0.000		borrow	time borrowed
3	1.134	1.134			clock path
1	1.134	1.134	R		clock network delay
4	1.164	0.030			clock uncertainty
5	1.299	0.135		uTh	dst

2.4.4.5.4. Same Frequency Clocks with Destination Clock Offset

In this example, the source and destination clocks have the same frequency, but the destination clock is offset with a positive phase shift. Both the source and destination clocks have a period of 10 ns. The destination clock has a positive phase shift of 2 ns with respect to the source clock.

The following example shows a design with the same frequency clocks and a destination clock offset.

Figure 119. Same Frequency Clocks with Destination Clock Offset Diagram



The following timing diagram shows the default setup check analysis that the Timing Analyzer performs.

Figure 120. Setup Timing Diagram

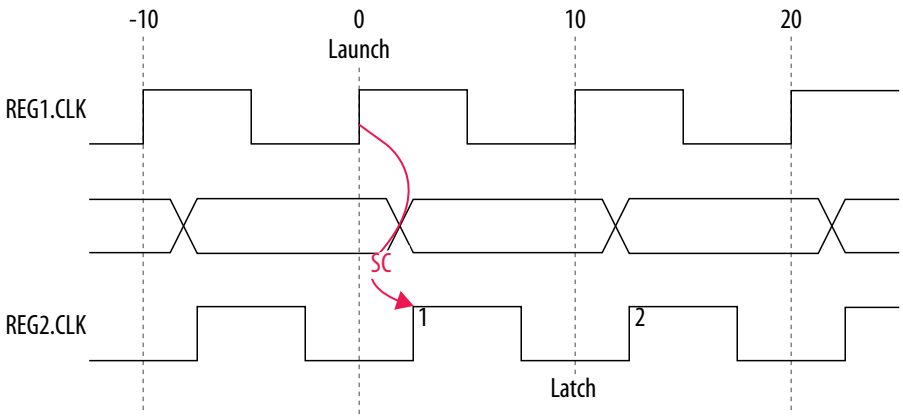


Figure 121. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 2\text{ ns} - 0\text{ ns} \\ &= 2\text{ ns} \end{aligned}$$

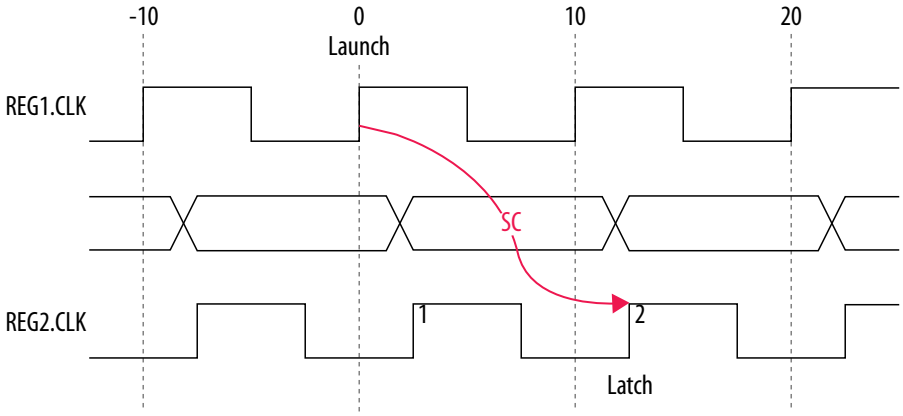
The setup relationship shown is too pessimistic and is not the setup relationship required for typical designs. To adjust the default analysis, you assign an end multicycle setup exception of two. The following shows a multicycle exception that adjusts the default analysis:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 2
```

The following timing diagram shows the preferred setup relationship for this example:

Figure 122. Preferred Setup Relationship



The following timing diagram shows the default hold check analysis that the Timing Analyzer performs with an end multicycle setup value of two.

Figure 123. Default Hold Check

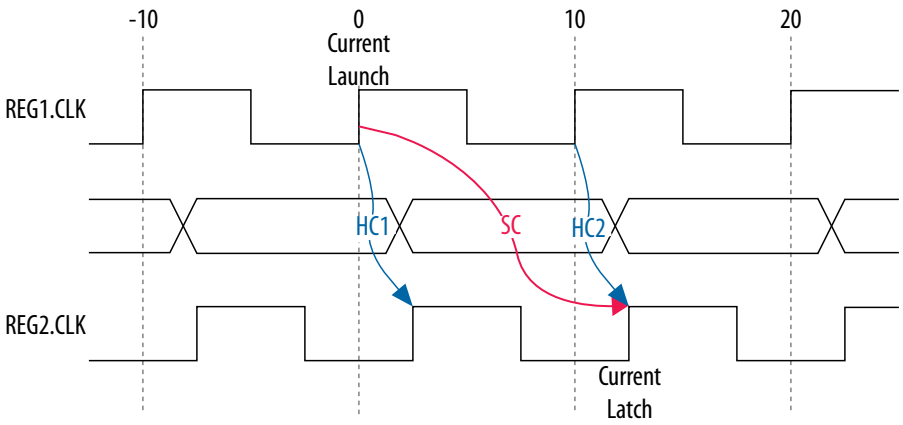


Figure 124. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 2\ \text{ns} \\ &= -2\ \text{ns} \end{aligned}$$

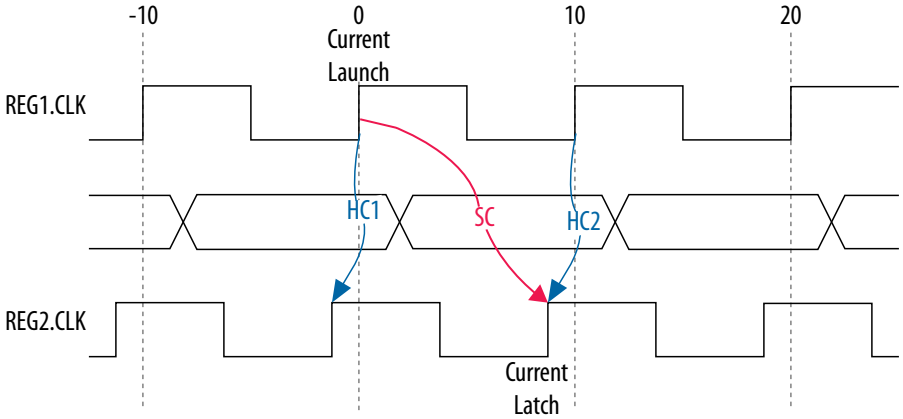
$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 12\ \text{ns} \\ &= -2\ \text{ns} \end{aligned}$$

In this example, the default hold analysis returns the preferred hold requirements and no multicycle hold exceptions are required.

The associated setup and hold analysis if the phase shift is –2 ns. In this example, the default hold analysis is correct for the negative phase shift of 2 ns, and no multicycle exceptions are required.

Figure 125. Negative Phase Shift

The figure shows an example of the setup and hold analysis for a negative phase shift of -2 ns. In this example, the default setup and hold analysis are correct, and no multicycle exceptions are required.

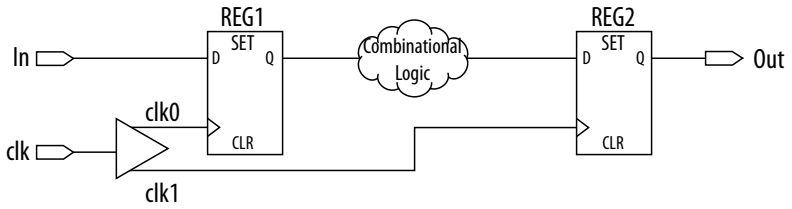


2.4.4.5.5. Destination Clock Frequency is a Multiple of the Source Clock Frequency

In this example, the destination clock frequency value of 5 ns is an integer multiple of the source clock frequency of 10 ns. The destination clock frequency can be an integer multiple of the source clock frequency when a PLL generates both clocks with a phase shift on the destination clock.

The following example shows a design in which the destination clock frequency is a multiple of the source clock frequency.

Figure 126. Destination Clock is Multiple of Source Clock



The following timing diagram shows the default setup check analysis that the Timing Analyzer performs:

Figure 127. Setup Timing Diagram

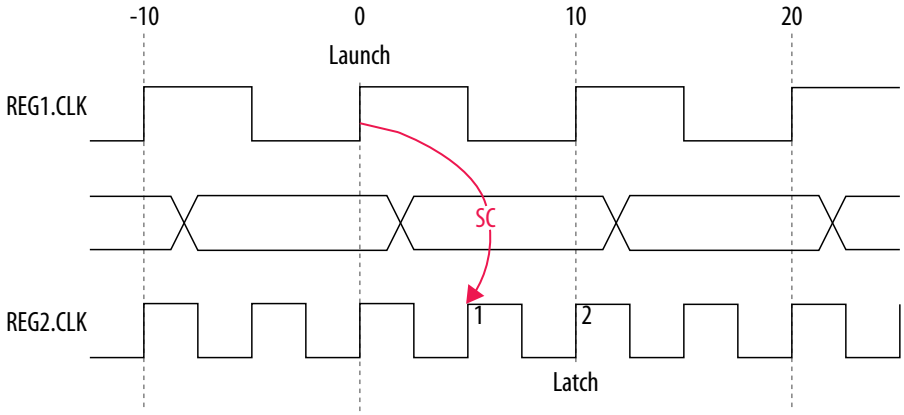


Figure 128. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 5\text{ ns} - 0\text{ ns} \\ &= 5\text{ ns} \end{aligned}$$

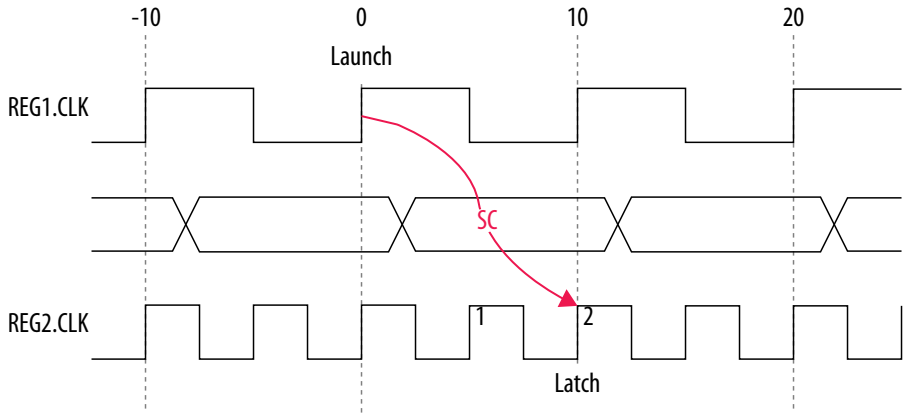
The setup relationship demonstrates that the data requires capture at edge two; therefore, you can relax the setup requirement. To correct the default analysis, you shift the latch edge by one clock period with an end multicycle setup exception of two. The following multicycle exception assignment adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 2
```

The following timing diagram shows the preferred setup relationship for this example:

Figure 129. Preferred Setup Analysis



The following timing diagram shows the default hold check analysis the Timing Analyzer performs with an end multicycle setup value of two.

Figure 130. Default Hold Check

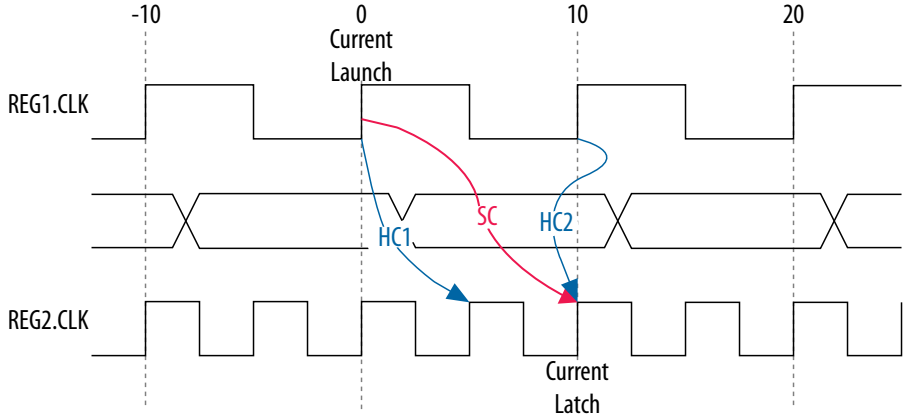


Figure 131. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 5\ \text{ns} \\ &= -5\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 10\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

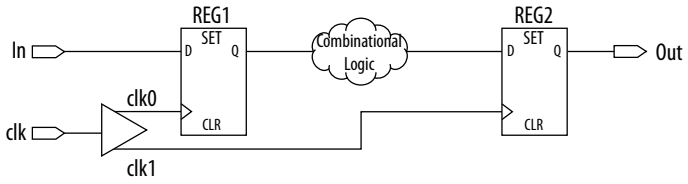
In this example, hold check one is too restrictive. The data is launched by the edge at 0 ns and must check against the data captured by the previous latch edge at 0 ns, which does not occur in hold check one. To correct the default analysis, you must use an end multicycle hold exception of one.

2.4.4.5.6. Destination Clock Frequency is a Multiple of the Source Clock Frequency with an Offset

This example is a combination of the previous two examples. The destination clock frequency is an integer multiple of the source clock frequency, and the destination clock has a positive phase shift. The destination clock frequency is 5 ns, and the source clock frequency is 10 ns. The destination clock also has a positive offset of 2 ns with respect to the source clock. The destination clock frequency can be an integer multiple of the source clock frequency. The destination clock frequency can be with an offset when a PLL generates both clocks with a phase shift on the destination clock.

The following example shows a design in which the destination clock frequency is a multiple of the source clock frequency with an offset.

Figure 132. Destination Clock is Multiple of Source Clock with Offset



The timing diagram for the default setup check analysis the Timing Analyzer performs.

Figure 133. Setup Timing Diagram

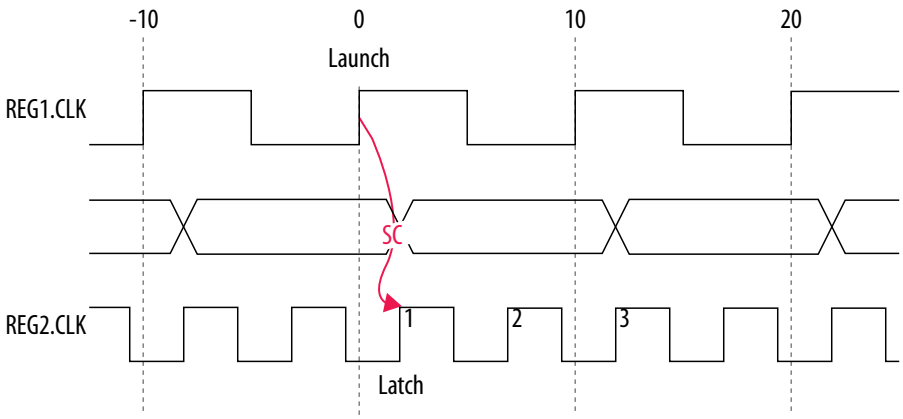


Figure 134. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 2\text{ ns} - 0\text{ ns} \\ &= 2\text{ ns} \end{aligned}$$

The setup relationship in this example demonstrates that the data does not require capture at edge one, but rather requires capture at edge three; therefore, you can relax the setup requirement. To adjust the default analysis, you shift the latch edge by two clock periods, and specify an end multicycle setup exception of three.

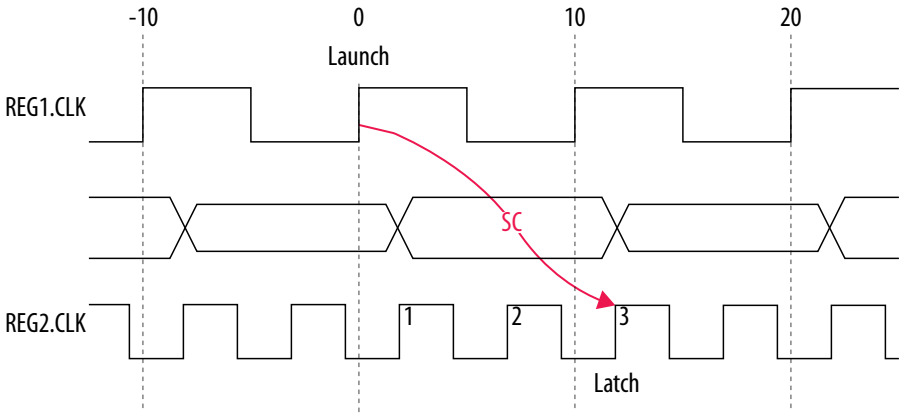
The multicycle exception adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -end 3
```

The timing diagram for the preferred setup relationship for this example.

Figure 135. Preferred Setup Analysis



The following timing diagram shows the default hold check analysis that the Timing Analyzer performs with an end multicycle setup value of three:

Figure 136. Default Hold Check

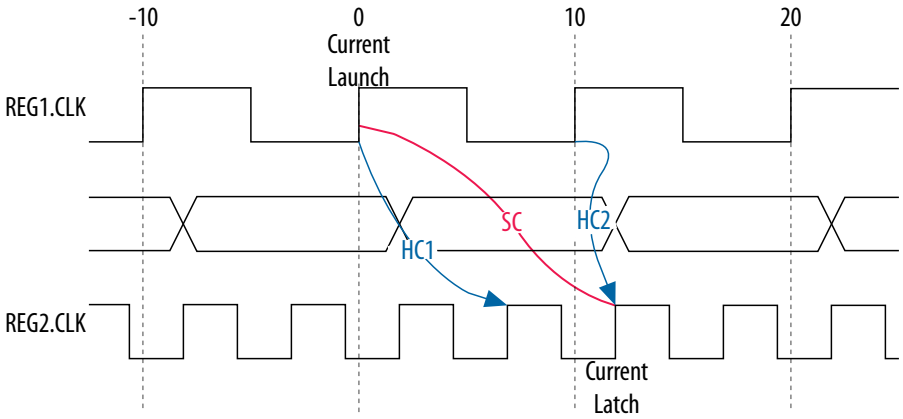


Figure 137. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 7\text{ ns} \\ &= -7\text{ ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\text{ ns} - 12\text{ ns} \\ &= -2\text{ ns} \end{aligned}$$

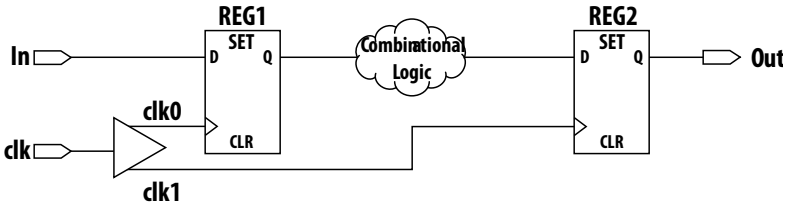
In this example, the hold check one is too restrictive. The data is launched by the edge at 0 ns, and must check against the data that the previous latch edge at 2ns captures. You can use the multicycle hold assignment of 1 to correct this.

2.4.4.5.7. Source Clock Frequency is a Multiple of the Destination Clock Frequency

In this example, the source clock frequency value of 5 ns is an integer multiple of the destination clock frequency of 10 ns. The source clock frequency can be an integer multiple of the destination clock frequency when a PLL generates both clocks and use different multiplication and division factors.

In the following example the source clock frequency is a multiple of the destination clock frequency:

Figure 138. Source Clock Frequency is Multiple of Destination Clock Frequency:



The following timing diagram shows the default setup check analysis the Timing Analyzer performs:

Figure 139. Default Setup Check Analysis

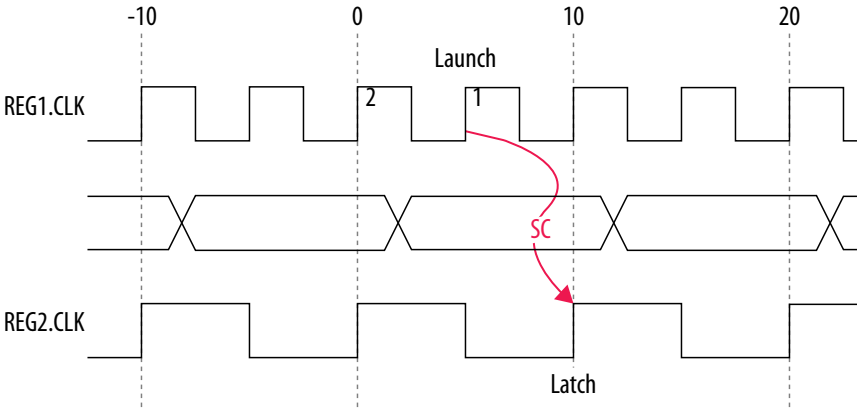


Figure 140. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 10\ \text{ns} - 5\ \text{ns} \\ &= 5\ \text{ns} \end{aligned}$$

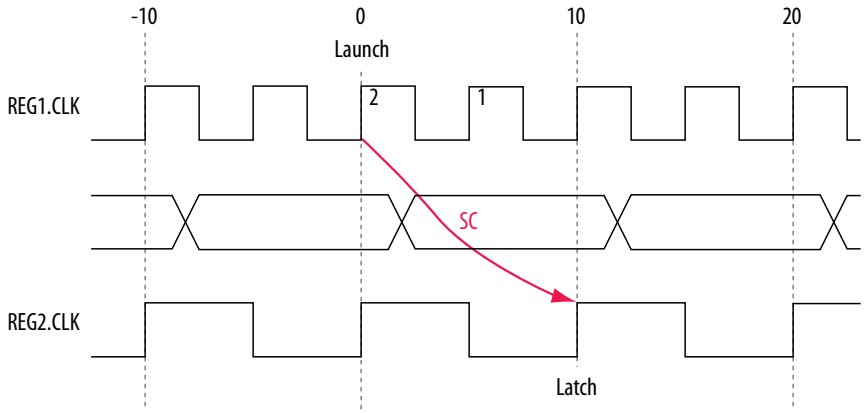
The setup relationship demonstrates that the data launched at edge one does not require capture, and the data launched at edge two requires capture; therefore, you can relax the setup requirement. To correct the default analysis, you shift the launch edge by one clock period with a start multicycle setup exception of two. The following multicycle exception adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -start 2
```

The following timing diagram shows the preferred setup relationship for this example:

Figure 141. Preferred Setup Check Analysis



The following timing diagram shows the default hold check analysis the Timing Analyzer performs for a start multicycle setup value of two:

Figure 142. Default Hold Check

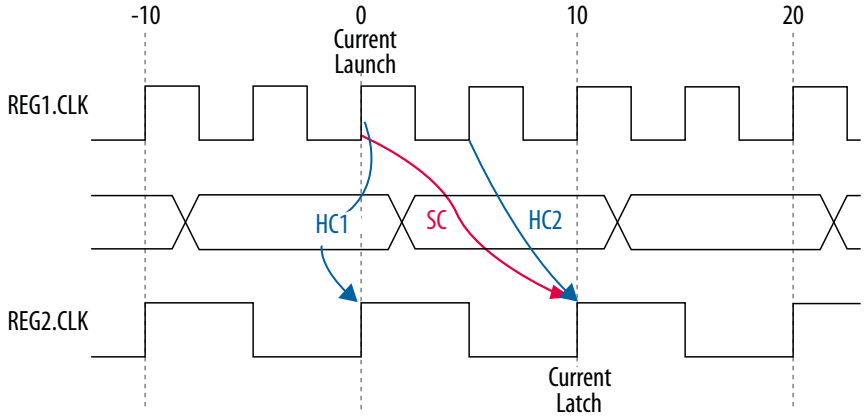


Figure 143. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 0\text{ ns} \\ &= 0\text{ ns} \end{aligned}$$

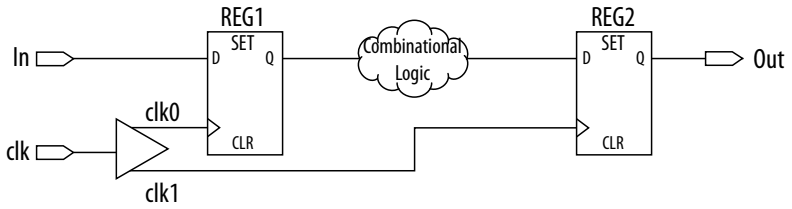
$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 5\text{ ns} - 10\text{ ns} \\ &= -5\text{ ns} \end{aligned}$$

In this example, the hold check two is too restrictive. The data is launched next by the edge at 10 ns and must check against the data captured by the current latch edge at 10 ns, which does not occur in hold check two. To correct the default analysis, you use a start multicycle hold exception of one.

2.4.4.5.8. Source Clock Frequency is a Multiple of the Destination Clock Frequency with an Offset

In this example, the source clock frequency is an integer multiple of the destination clock frequency and the destination clock has a positive phase offset. The source clock frequency is 5 ns and destination clock frequency is 10 ns. The destination clock also has a positive offset of 2 ns with respect to the source clock. The source clock frequency can be an integer multiple of the destination clock frequency with an offset when a PLL generates both clocks with different multiplication.

Figure 144. Source Clock Frequency is Multiple of Destination Clock Frequency with Offset



The following timing diagram shows the default setup check analysis the Timing Analyzer performs:

Figure 145. Setup Timing Diagram

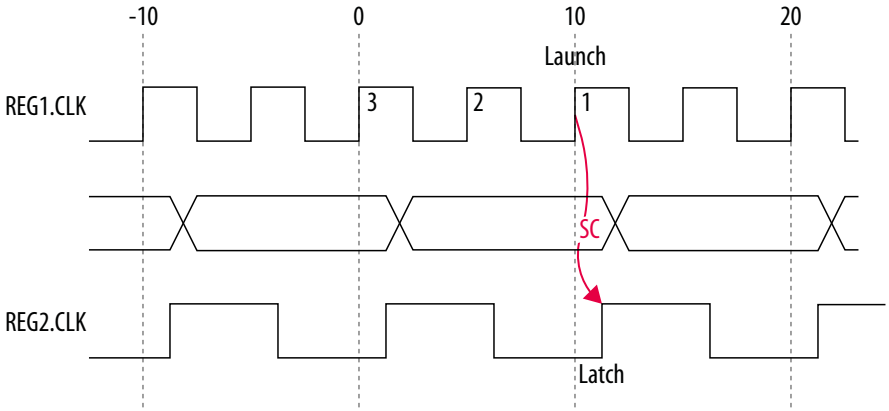


Figure 146. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 12\text{ ns} - 10\text{ ns} \\ &= 2\text{ ns} \end{aligned}$$

The setup relationship in this example demonstrates that the data is not launched at edge one, and the data that is launched at edge three must be captured; therefore, you can relax the setup requirement. To correct the default analysis, you shift the launch edge by two clock periods with a start multicycle setup exception of three.

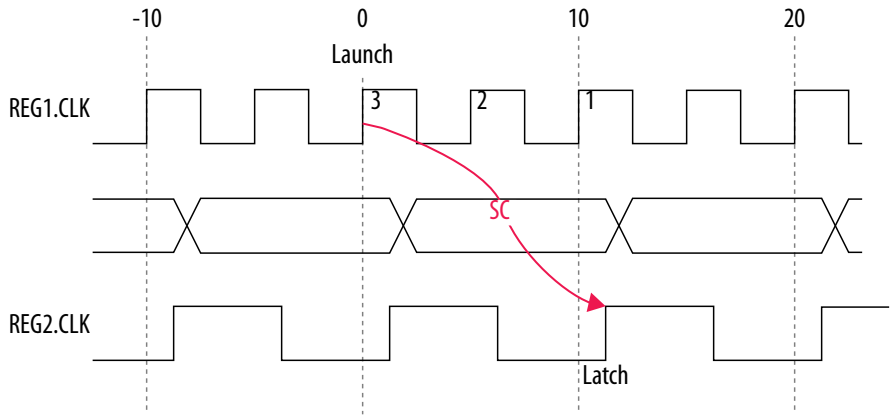
The following multicycle exception adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -start 3
```

The following timing diagram shows the preferred setup relationship for this example:

Figure 147. Preferred Setup Check Analysis



The Timing Analyzer performs the following calculation to determine the hold check:

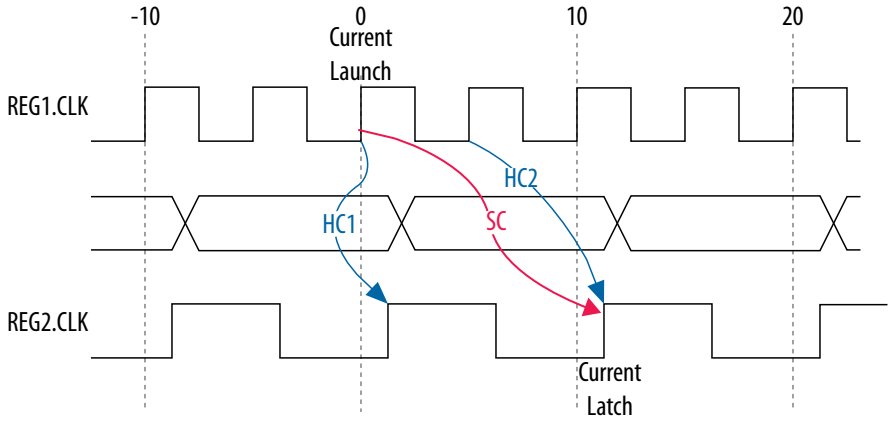
Figure 148. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 2\text{ ns} \\ &= -2\text{ ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 5\text{ ns} - 12\text{ ns} \\ &= -7\text{ ns} \end{aligned}$$

The following timing diagram shows the default hold check analysis the Timing Analyzer performs for a start multicycle setup value of three:

Figure 149. Default Hold Check Analysis



In this example, the hold check two is too restrictive. The data is launched next by the edge at 10 ns and must check against the data captured by the current latch edge at 12 ns, which does not occur in hold check two. To correct the default analysis, you must specify a multicycle hold exception of one.

2.4.5. Delay Annotation

To modify the default delay values used during timing analysis, use the `set_annotated_delay` and `set_timing_derate` commands. You must update the timing netlist to apply these commands.

To specify different operating conditions in a single `.sdc` file, rather than having multiple `.sdc` files that specify different operating conditions, use the `set_annotated_delay -operating_conditions` command.

Related Information

- set_timing_derate Command, Quartus Prime Help
- set_annotated_delay Command, Quartus Prime Help

2.5. Timing Report Descriptions

The Timing Analyzer generates only a subset of all available reports by default, including the Setup Summary and Timing Analyzer Summary reports. However, you can generate dozens of other detailed reports in the Timing Analyzer GUI, or with command-line commands to help pin-point timing issues. You can customize the display of information in the reports.

You can automatically generate reports that are useful to you, to display after opening a project in the stand-alone Timing Analyzer GUI, as The quartus_staw executable describes.

The following section describes a partial list of all the timing analysis reports that you can generate:

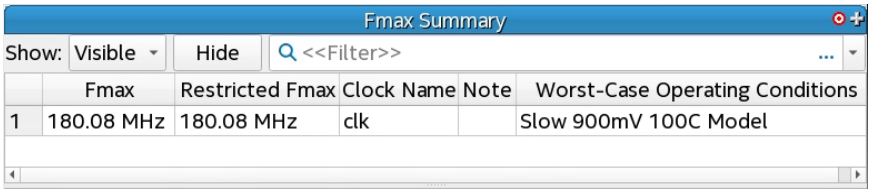
Related Information

Step 4: Analyze Timing Reports on page 40

2.5.1. Report Fmax Summary

The Timing Analyzer **Reports ➤ Datasheet ➤ Report Fmax Summary** command generates a report panel showing the potential maximum frequency for every clock in your design. The **Fmax** column reports the fastest frequency that your clock can run, and still pass `report_timing -setup -intra_clock` with a slack of `0`. The equivalent console command is `report_clock_fmax_summary`.

Figure 150. Fmax Summary Report



Note: The related `get_clock_fmax_info` command returns a Tcl list, which is useful for scripting and parsing. Refer to *get_clock_fmax_info (::quartus::sta)* in *Quartus Prime Pro Edition User Guide Scripting*.

Timing analysis only computes f_{MAX} for paths where the source and destination registers or ports are driven by the same clock. Timing analysis ignore paths of different clocks, including generated clocks. For paths between a clock and its inversion, the Timing Analyzer computes f_{MAX} as if the rising and falling edges of the clock scale along with f_{MAX}, such that duty cycle (by percentage) is maintained.

However, the **Fmax** report does not indicate whether your design meets timing for recovery, removal, nor setup or hold without the `intra_clock` option. For these reasons, always make sure to view the Setup, Hold, Recovery, Removal, and Min Pulse Width slack summaries to determine whether your design meets timing.

The **Restricted Fmax** column reports the lesser of the following values:

- The fastest frequency that your clock can run, and still pass `report_timing -hold -intra_clock` with a slack of `0` or `report_min_pulse_width` with a slack of `0`.
- The **Fmax** column value.(5)

Restricted Fmax considers hold timing in addition to setup timing, as well as minimum pulse and minimum period restrictions. Similar to unrestricted f_{MAX}, the analysis computes the restricted f_{MAX} as if the rising and falling edges of the clock scale along with f_{MAX}, such that the duty cycle (in terms of a percentage) is maintained.

The **Restricted Fmax** may display text indicating any of the following limiting factors:

- Limit due to hold check
- Limit due to minimum pulse width restriction
- Limit due to high minimum pulse width restriction
- Limit due to low minimum pulse width restriction

(5) The **Restricted Fmax** column never reports a value higher than the **Fmax** column.

Typically, hold checks do not limit the maximum frequency (f_{MAX}) because the checks are for same-edge relationships, and therefore independent of clock frequency. For example, when launch equals zero and latch equals zero. However, with an inverted clock transfer, or a multicycle transfer, the hold relationship is not a same-edge transfer and changes with the clock frequency.

Refer to the timing reports, such as those that you can generate using `report_timing`, or using minimum pulse width reports via the `report_min_pulse_width` command for details of specific paths, registers, or ports.

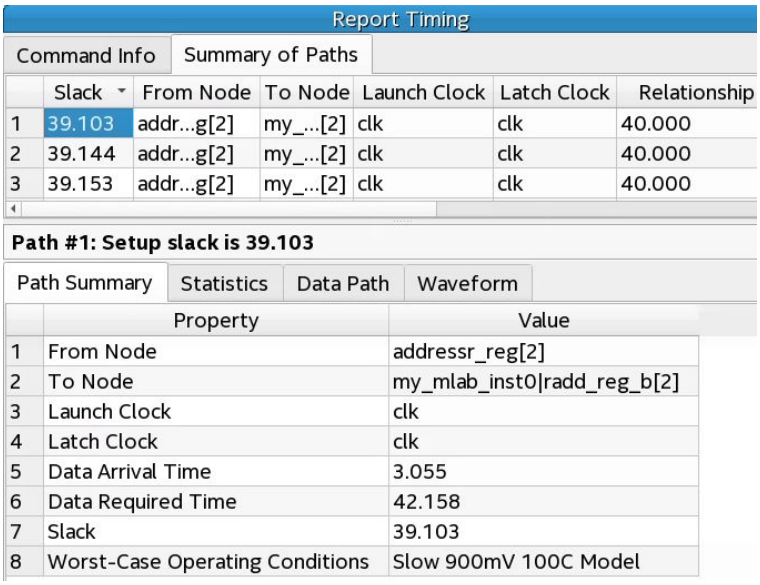
Related Information

Quartus Prime Pro Edition User Guide: Scripting

2.5.2. Report Timing

The Timing Analyzer's **Reports ➤ Timing Slack ➤ Report Timing...** command allows you to specify settings to report the timing of any path or clock domain in the design. The equivalent scripting command is `report_timing`.

Figure 151. Report Timing Report



You can specify diverse options to customize the reporting. You can specify the **Clocks** and **Targets** that the report displays, the **Analysis Type** to run, whether to display **Extra Info** in the report, and the **Output** options for the report. For example, you can increase the number of paths to report, add a **Target** filter, and add a **From Clock**.

Figure 152. Report Timing Dialog Box (Top Section)

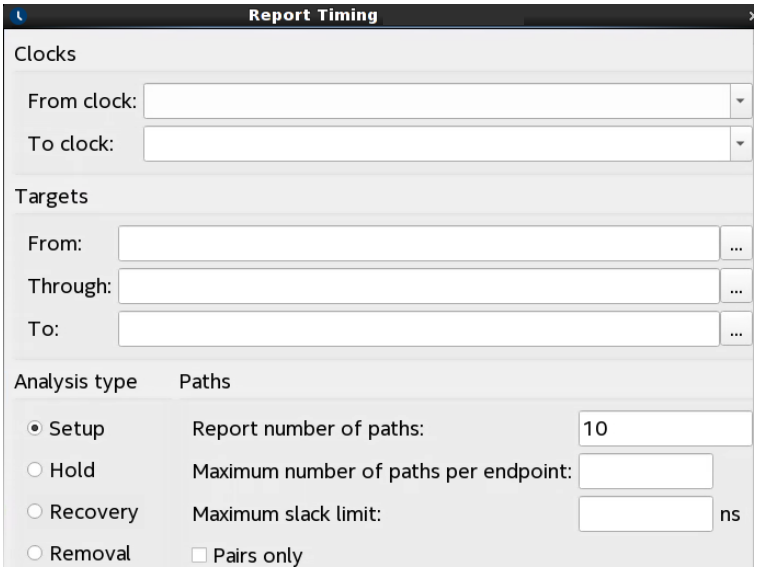


Figure 153. Report Timing Dialog Box (Bottom Section)

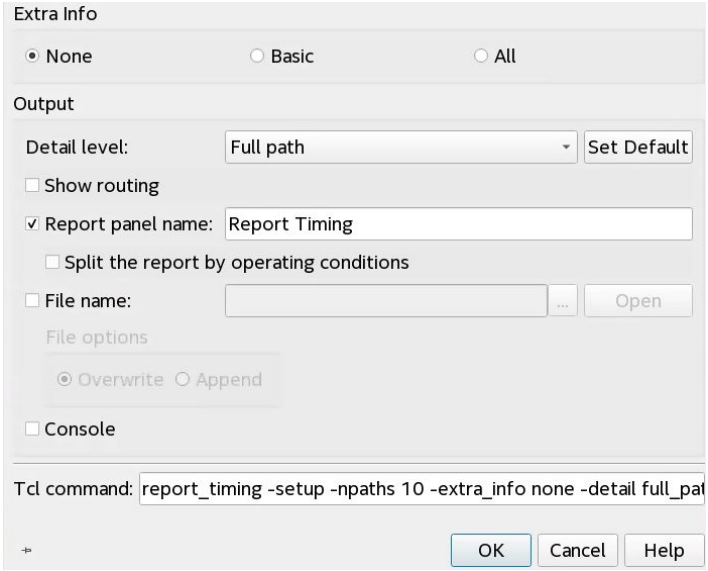


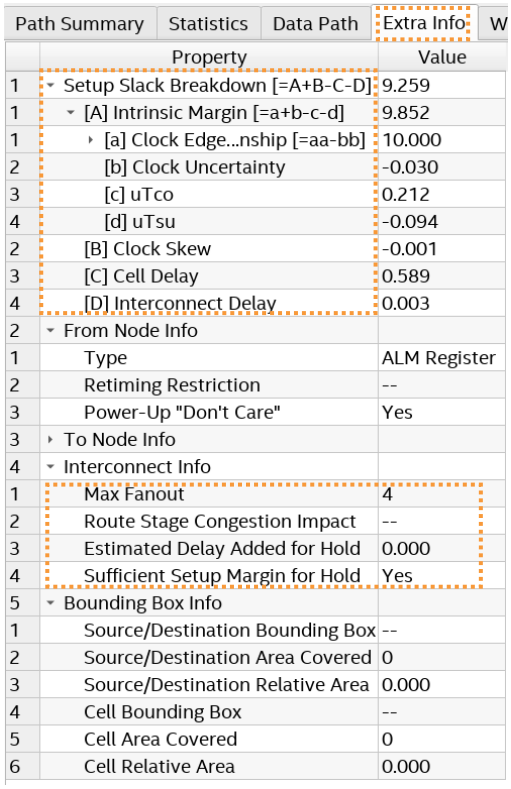
Table 23. Report Timing Settings

Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Targets	Specifies the target node for **From Clock** and **To Clock** to report paths with only those endpoints. Specify an I/O or register name or I/O port for this option. The field also supports wildcard characters. For example, to report only paths within a specific hierarchy: `report_timing -from *|egress:egress_inst|* \` `-to *|egress:egress_inst|* -(other options)`

continued...

Option	Description
	When the **From**, **To**, or **Through** boxes are empty, the Timing Analyzer assumes all possible targets in the device. The **Through** option limits the report to paths that pass through combinatorial logic, or a particular pin on a cell.
Analysis type	The **Analysis type** options are **Setup**, **Hold**, **Recovery**, or **Removal**. The Timing Analyzer reports the results for the type of analysis you select.
Paths	Specifies the number of paths to display by endpoint and slack level. The default value for **Report number of paths** is 10, otherwise, the report can be very long. Enable **Pairs only** to list only one path for each pair of source and destination. Limit further with **Maximum number of paths per endpoints**. You can also filter paths by entering a value in the **Maximum slack limit** field.
Extra Info	Provides additional data that is relevant for diagnosing timing failure root cause, such as setup slack breakdown, and unexpected routing detours caused by congestion and hold time fix-up. Specify whether to include **None**, **Basic**, or **All** extra information in the report. The **Extra Info** tab data can help you identify potential, unnecessary routing detours, as well as placement or circuit issues that restrict the path f_{MAX} performance. Refer to Setup Slack Breakdown On the Extra Info Tab on page 131. • **All**—report includes **Extra Info** tab that reports extra information for source timing endpoints that pass through the unregistered output of a RAM or DSP block, or for destination timing endpoints that pass through the unregistered input of a DSP block. The **Data Path** tab includes Estimated Delay Added for Hold and Route Stage Congestion Impact data. • **Basic**—report includes the **Extra Info** tab but no extra information on the **Data Path** tab. • **None**—report includes no **Extra Info** tab or other extra information on the **Data Path** tab.
Output	Specify the path types the analysis includes in output for **Detail level**: • **Summary**—level includes basic summary reports. Review the **Clock Skew** column in the **Summary** report. If the skew is less than +/-150ps, the clock tree is well balanced between source and destination. • **Path only**—displays all the detailed information, except the **Data Path** tab displays the clock tree as one line item. • **Path and Clock**—displays the same as **Path only** with respect to the clock. • **Full path**—when higher clock skew is present, enable the **Full path** option. This option breaks the clock tree into greater detail, showing every cell, including the input buffer, PLL, global buffer (called CLKCTRL_), and any logic. Review this data to determine the cause of clock skew in your design. Use the **Full path** option for I/O analysis because only the source clock or destination clock is inside the FPGA, and therefore the delay is a critical factor to meet timing.
Show routing	Shows routing data in the report.
Split the report by operating conditions	For the operating condition timing corners, subdivides the data by each operating condition.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

Figure 154. Extra Info Tab



Path Summary | Statistics | Data Path | Extra Info | W

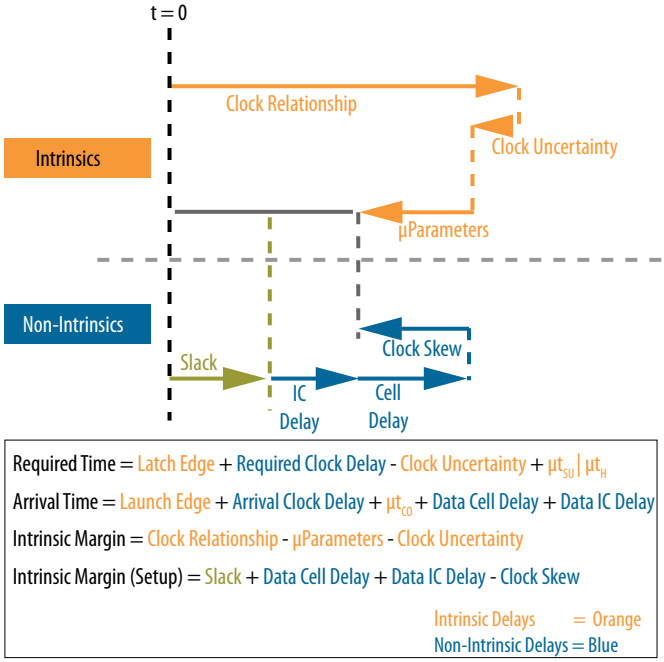
	Property	Value
1	Setup Slack Breakdown [=A+B-C-D]	9.259
1	[A] Intrinsic Margin [=a+b-c-d]	9.852
1	[a] Clock Edge...nship [=aa-bb]	10.000
2	[b] Clock Uncertainty	-0.030
3	[c] uTco	0.212
4	[d] uTsu	-0.094
2	[B] Clock Skew	-0.001
3	[C] Cell Delay	0.589
4	[D] Interconnect Delay	0.003
2	From Node Info	
1	Type	ALM Register
2	Retiming Restriction	--
3	Power-Up "Don't Care"	Yes
3	To Node Info	
4	Interconnect Info	
1	Max Fanout	4
2	Route Stage Congestion Impact	--
3	Estimated Delay Added for Hold	0.000
4	Sufficient Setup Margin for Hold	Yes
5	Bounding Box Info	
1	Source/Destination Bounding Box	--
2	Source/Destination Area Covered	0
3	Source/Destination Relative Area	0.000
4	Cell Bounding Box	--
5	Cell Area Covered	0
6	Cell Relative Area	0.000

Setup Slack Breakdown On the Extra Info Tab

The **Extra Info** tab contains other timing metrics to help you diagnose timing closure issues, including **Setup Slack Breakdown** for the path.

The slack of a path specifies the margin by which the path meets its timing requirement. The setup slack breakdown is a numeric value that the Timing Analyzer calculates from the following timing requirements and path element delays:

Figure 155. Setup Slack Breakdown Calculations



A path can fail timing requirements for many varied reasons. For example, the clock relationship can be impossibly tight, or there can be excessive routing delays that alone cause failure for the timing path. Calculating the intrinsic margin of a timing path, and then comparing that margin to other delays of the path, can help identify the specific reasons why a path fails its timing requirement.

The **Extra Info** tab can help you identify potential significant or unexpected routing detours caused by congestion and hold time fix-up. The **Extra Info** tab can also report extra information for source timing endpoints that pass through the unregistered output of a RAM or DSP block, or for destination timing endpoints that pass through the unregistered input of a DSP block.

You can review the **Extra Info** data and **Locate Path** or **Locate Chip Area** in Chip Planner, Technology Map Viewer, or Resource Property Viewer to determine whether to make changes to improve placement and routing.

Some delay elements are more sensitive to a path's placement and routing than others. Intrinsic delays that are part of **Setup Slack Breakdown** are less sensitive to placement and routing, and are inherent in the RTL and timing requirements. Non-intrinsic delays are the other delays that are sensitive to placement and routing.

Table 24. Extra Info Tab Data

Extra Info Data	Description
Intrinsic Margin	Reports the intrinsic and non-intrinsic timing elements that comprise the timing path slack value. Intrinsic margin is a numeric value that the Timing Analyzer calculates from the timing requirements and path element delays. The Timing Analyzer also derives the slack of the path from the same requirements and delays, but with a different calculation. Intrinsic delays are less sensitive to placement and routing, and are inherent in the RTL and timing requirements. Non-intrinsic delays are the other delays that are sensitive to placement and routing.
From Node Info	Specifies the node Type, any Retiming Restriction, and any Power-Up "Don't Care" attributes for the From Node. Consider removing the retiming restriction to allow retiming and improve performance for timing closure.
To Node Info	Specifies the node Type, any Retiming Restriction, and any Power-Up "Don't Care" attributes for the To Node. Consider removing the retiming restriction to allow retiming and improve performance for timing closure.
Max Fanout	Reports the maximum fan-out of register and combinational nodes in the path.
Route Stage Congestion Impact	Reports whether routing has a **Low**, **Medium**, or **High** impact on congestion. A **Low** value suggests timing issues are not congestion related. A **High** value suggests competition for scarce routing resources plays a role in poor timing.
Estimated Delay Added for Hold	Reports the estimated amount of delay added on to the fastest delay route to satisfy hold. This value can help you determine whether delays are routing congestion or Hold related.
Sufficient Setup Margin for Hold	Reports whether the setup margin is suitable for the hold timing. **Yes**, indicates that the setup margin is sufficient. **No** indicates that the setup margin is insufficient for hold timing.
Source/Destination Bounding Box	Reports the lower-left and upper-right coordinates for the boundary box enclosing the source and destination registers. In an ideal case, the **Source/Destination Bounding Box**, **Cell Bounding Box**, and **Interconnect Bounding Box** values are roughly the same, and the relative areas are approximately 1.0. If the cell bounding box size grows relative to the **Source/Destination Bounding Box**, that can indicate a potential unnecessary routing detour on the path.
Source/Destination Area Covered	Reports the total area covered in terms of LABs.
Source/Destination Relative Area	Reports the area for the source and destination, relative to the **Source/Destination Bounding Box**. The value is always 1.0, which equals the same size.
Cell Bounding Box	Reports the lower-left and upper-right coordinates for the boundary box enclosing the source and destination registers, and any cells in the path.
Cell Area Covered	Reports the area for the cell, relative to the **Source/Destination Bounding Box**. A value of 1.0 equals the same size. A value greater than 1.0 can indicate a path has a cell outside of the space between the registers in the path.

The following describe the interpretation of timing conditions indicated by the **Setup Slack Breakdown**:

- **When the Setup Slack Breakdown is less than 0**—the path has such a tight timing relationship, a significant difference in microparameters, or such significant clock source uncertainty, that the path fails before the addition of any delay. Review the SDC constraints to verify that the timing relationship is correct. An incorrect relationship can exist between unrelated clocks that lack the proper timing cut. Ensure that parameterizable hard blocks (such as 20K RAM and DSP blocks) are fully registered. Investigate clock sources to verify that the clocks use global signals for routing.
- **When the clock skew exceeds the Setup Slack Breakdown**—address the clock transfer to meet timing on the path. You may need to create clock region assignments. You might also need to redesign cross-clock transfers to switch from synchronous to asynchronous implementation, such as with a FIFO or other handshake.

- **When the cell delay is greater than its intrinsic margin**—reduce the cell delay, as the path would fail timing even if the clocks are perfect and use no routing wires. Rewrite RTL to reduce the logic depth, restructure logic to allow the Compiler to use faster LUT inputs, or unblock retiming optimizations. The Compiler can automatically retime registers to reduce logic depth, but only in ways that maintain functionality and that the device architecture supports. To unblock the Hyper-Retimer, remove asynchronous resets and initial conditions.
- **When the interconnect delay is greater than its intrinsic margin**—the path would fail timing even if the clocks are perfect, and there is no logic. This occurs if registers are too far apart, or a timing path detours around a congested chip area. Review the fan-in and fan-out of registers that are far apart. Apply Logic Lock regions so the Fitter places the registers closer together. Use Logic Lock regions only after determining why placement is initially poor.

Related Information

Hyperflex Architecture High-Performance Design Handbook

2.5.3. Report Timing By Source Files

The Timing Analyzer's **Reports ➤ Timing Slack ➤ Report Timing By Source Files** command allows you to specify settings to report the timing of any path or clock domain in the design, similar to Report Timing on page 128. However, **Report Timing By Source Files** groups the timing paths by the containing entity, and groups the entities by the source file that defines the entity.

This report allows you to attribute timing paths to exactly one instance in the design. The Path TNS column shows the sum of all negative slacks within a file or entity. The equivalent scripting command is `report_timing_by_source_files`.

From the report in the Timing Analyzer GUI, you can right-click on any source file to open it in a text editor to easily investigate timing paths within a particular source file.

2.5.4. Report Data Delay

You can run the Timing Analyzer's **Reports ➤ Other Slack Analyses ➤ Report Data Delay...** command to generate a custom report showing the worst-case slack for the datapath delay exception for a given path. The report shows only paths covered by data delay (`set_data_delay`) constraints, including paths whose non-datapath analysis is cut by a false path.

Note: Exclusive clock groups (set with `set_clock_groups -exclusive`) override `set_data_delays` constraints.

Related Information

Timing Exception Precedence on page 99

2.5.5. Report Net Delay

The Timing Analyzer's **Reports ➤ Other Slack Analyses ➤ Report Net Delay...** command to configure and display a customized report that details the results of net delay analysis, as defined by timing constraints and exceptions.

Each `set_net_delay` command is treated as a separate analysis and `report_net_delay` reports the results of all `set_net_delay` commands in a single report. The report contains each `set_net_delay` command with the worst case slack result followed by the results of each edge matching the criteria set by that `set_net_delay` command. These results are ordered based on the slack value.

2.5.6. Report Clocks and Clock Network

The Timing Analyzer's **Reports ➤ Clocks ➤ Report Clocks** command reports all clock signals in the design. The equivalent scripting command is `report_clocks`.

Report Clocks generates the Clock Summary report that lists details about all of the signals with clock setting constraints in the design.

Figure 156. Clock Summary Report Shows Properties of Clock Signals in Design

Clocks Summary

Show: Visible | Hide | <<Filter>>

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by	Phase	Offset
1	clk	Base	40.000	25.0 MHz	0.000	20.000					

Similarly, you can click the **Reports ➤ Clocks ➤ Report Clock Network...** command to generate a custom report that helps you identify and evaluate advanced clock structures, such as clock muxes, clock gates, and clock dividers.

Figure 157. Report Clock Network Dialog Box

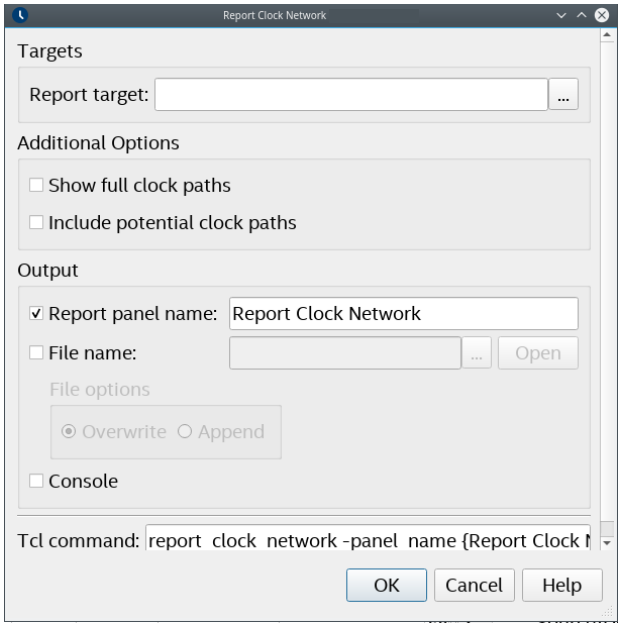
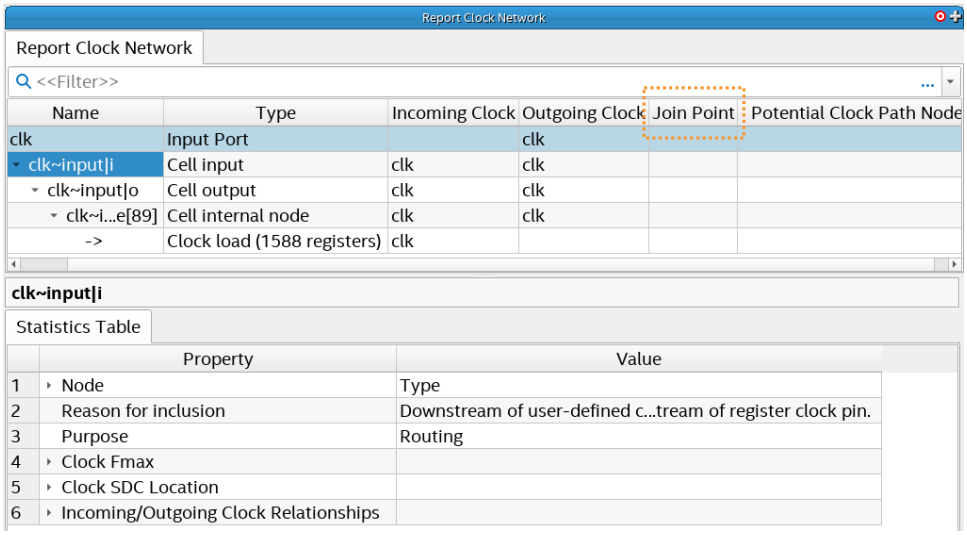


Table 25. Report Clock Network Dialog Box Settings

Option	Description
Report target	Specifies the collection of clocks and nodes that you want to analyze and report.
Expand clock path	Displays all subordinate nodes in expanded view. The default display collapses trivial nodes in the report.
Include potential clock paths	Includes nodes in the report that are not on a clock path, but are upstream of a register clock port.
Report panel name	Specifies the name that appears in the report panel title bar.
File name	Specifies the name of an optional output file to contain report data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

This report shows the nodes on the clock network hierarchically, starting from the input clock ports, followed by any other nodes that transform or route the clocks to the clock loads. The **Join Points** indicate whether the clock network has convergence, such as with clock muxes. The **Statistics Table** provides more details about the signals that you select in the report, such as the relationships between the incoming and outgoing clocks of this node.

Figure 158. Report Clock Network Report

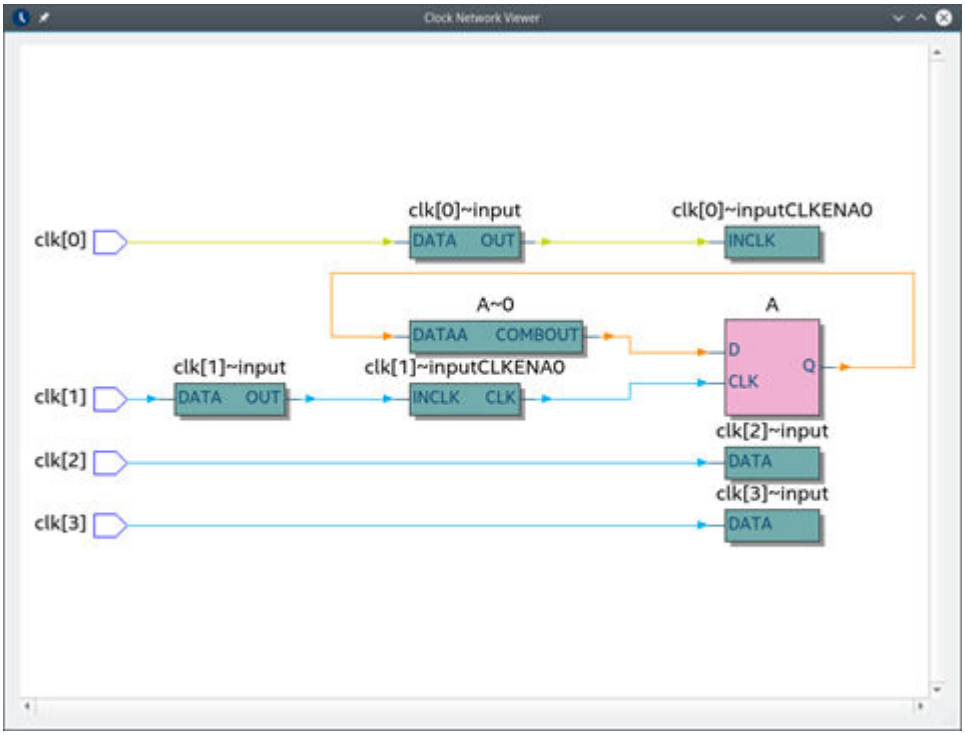


Right-click any of the nodes in the Clock Network report to click **Open Clock Network Viewer**. The Clock Network Viewer displays a graphical representation of the clock domains and constraints on the clock network to help you to see clock tree problems, such as signals entering and exiting globals. Use this graphical view to determine which clocks drive portions of the design. The Clock Network Viewer color codes the clock connection edge to indicate the clock types.

- Blue—the base clock
- Orange—a derived clock
- Green—a multicycle clock

The display shows a truncated signal name by default. Hover the mouse over the clock signals to display the full signal name. Right-click any signal to display the **Color Legend**. Click the **Zoom** controls to view more detail. You can export the schematic view as a PDF from the right-click menu.

Figure 159. Clock Network Viewer



You can also right-click and choose **Report Paths from Node**, **Report Paths Thru Node**, **Report Paths To Node**, or **Focus On Node** to rerun the report on the selected node.

2.5.7. Report Clock Transfers

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report Clock Transfers** command reports all clock-to-clock transfers in the design. The equivalent scripting command is `report_clock_transfers`.

Report Clock Transfers generates the Setup Transfers report and the Hold Transfers report that display data about the clock-to-clock transfers.

Figure 160. Setup Transfers Report Shows Clock-to-Clock Transfers

Setup Transfers

Show: Visible | Hide | <<Filter>>

	From Clock	To Clock	RR Paths	FR Paths	RF Paths	FF Paths	Clock Pair Classification	Worst-Case Slack	Worst-Case Operating Conditions
1	clock	clock	34	0	0	0	Intra-Clock (Timed Safe)	3.319	Slow vid2 100C Model

The Setup Transfers report and Hold Transfers report display all possible transfers, including rising clock edge to rising clock edge (RR), falling clock edge to rising clock edge (FR), rising clock edge to falling clock edge (RF), and falling clock edge to falling clock edge (FF) paths.

- If a path exists in the design, the report column cell is white and lists the number of paths.
- If a path is a false path, the report column cell is light gray and contains the text "false path."
- If a path does not exist in the design, then the report column cell is dark gray.

The Setup Transfers report and Hold Transfers report also lists the Worst-Case Slack for setup, the Worst-Case Operating Conditions, and the Clock Pair Classification for each clock path. The Clock Pair Classification includes the following:

Table 26. Clock Pair Classifications

Clock Pair Classification	Definition
Intra-Clock (Timed Safe)	• From Clock and To Clock are the same. • No timing constraint required.
Inter-Clock Synchronous (Timed Safe)	• From Clock and To Clock relate synchronously, and have a known phase and frequency relationship. • Multicycle path constraint may or may not exist.
Asynchronous (Timed Unsafe)	• From Clock and To Clock are asynchronous. • Timing constraints (false path, clock groups, set_max_skew) do not exist.
Ignored (Not Timed)	• From Clock and To Clock are asynchronous. • Timing constraints (false path, clock groups, set_max_skew) exist and setup and hold slack are not applicable.

2.5.8. Report Metastability

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report Metastability...** command generates a list of synchronization register chains found in the design, and can provide estimates of the Mean Time Between Failures (MTBF) of each chain. The equivalent scripting command is `report_metastability`.

Metastable registers have outputs hovering at a voltage between high and low for a length of time beyond the normal t_{CO} for the register, which may cause subsequent registers that use this metastable signal to latch different values. Synchronize register chains when transferring data between unrelated clock domains to reduce the probability of the captured data signal becoming metastable.

A synchronization register chain is a sequence of registers with the same clock, that is driven by a pin, or logic from an unrelated clock domain. All but the last register in the chain must connect only to the next register, but may do so through logic.

The Metastability Report displays the following for each synchronization chain the analysis discovers:

- Typical Mean Time Between Failures (MTBF) for the chain
- MTBF equation and method of synchronizer identification
- Available settling time of the synchronization register chain
- Number of synchronization registers in the chain

- Names of synchronization registers in the chain
- name of asynchronous source registers
- Data toggle rate used in the MTBF estimation
- Source clock domain names
- Synchronization clock domain names

To help you analyze synchronizer chains in more detail, you can report the neighbor paths for synchronization registers in a chain. From the list of synchronization registers in the report in the Timing Analyzer GUI, you can right-click any of the synchronization register names an click **Report Neighbor Paths** in the context menu.

To report the timing for the synchronization registers, you can right-click and then click **Report Timing** for the synchronization registers. For the asynchronous source registers, right-click and click **Report Path**.

When you do not specify a data toggle rate, the Timing Analyzer uses the value of 12.5% of the frequency of the clock domain that contains the synchronizer to calculate MTBF.

Related Information

- Metastability Analysis on page 16
- Step 1: Specify General Timing Analyzer Settings on page 29

2.5.9. Report CDC Viewer

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report CDC Viewer...** command allows you to configure and display a custom clock domain crossing report and the Clock Domain Crossing (CDC) Viewer. The CDC Viewer graphically displays the setup, hold, recovery, or removal analysis of all clock transfers in your design. The equivalent scripting command is `report_cdc_viewer`.

Table 27. Report Clock Domain Crossing Viewer Settings

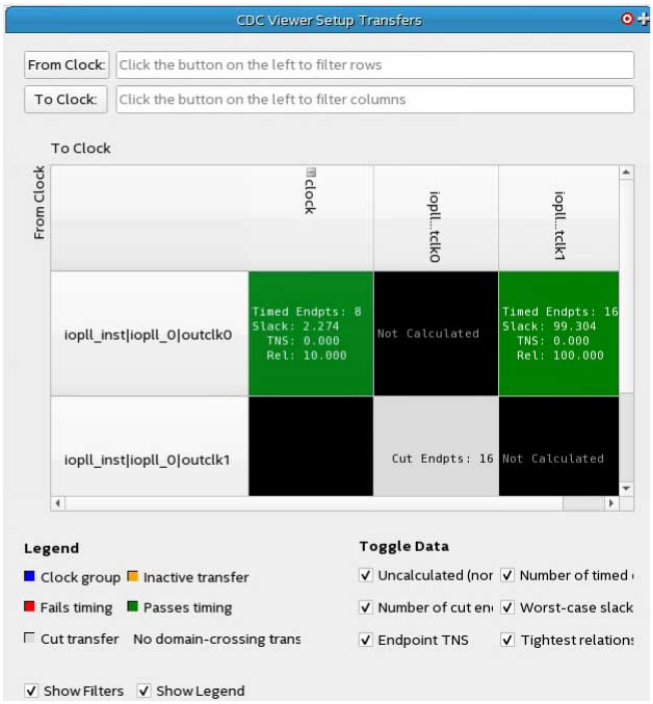
Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Analysis type	Options are **Setup**, **Hold**, **Recovery**, or **Removal**. The Timing Analyzer reports the results for the type of analysis you select.
Transfers	Specifies the type of clock transfers to include or exclude from the report, including **Timed transfers**, **Fully cut transfers**, **Clock groups, Inactive clocks**, and **Non-crossing transfers**. You can specify the **Maximum slack limit** and **Grid options** for the report.
Detail level	**Full** shows all details of the report and **Summary** filters the details and shows summary data.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data, and specify **Grid** or **List** format. *Note:* In grid format reports, clocks with non-crossing transfers always appear if they have transfers between other clocks.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

You can specify the following options to customize CDC Viewer reporting:

Table 28. CDC Viewer Report Controls

Control	Description
From Clock: and **To Clock:**	Filters the display according to the clock names you specify. Click **From Clock:** or **To Clock:** to search for specific clock names.
Legend	Defines the status colors. A color coded grid displays the clock transfer status. The clock headers list each clock with transfers in the design. The GUI truncates long clock names, but you can view the full name in a tool tip or by resizing the clock header cell. The GUI represents the generated clocks as children of the parent clock. A '+' icon next to a clock name indicates the presence of generated clocks. Clicking on the clock header displays the generated clocks associated with that clock.
Toggle Data	The text in each transfer cell contains data specific to each transfer. Turn on or off display of the following types of data: • **Number of timed endpoints** between clocks— the number of timed, endpoint-unique paths in the transfer. A path being "timed" means that analysis occurs on that path. Only paths with unique endpoints count towards this total. • **Number of cut endpoints** between clocks— the number of cut endpoint-unique paths, instead of timed paths. These paths are cut by either a false path or clock group assignment. Timing analysis skips such paths. • **Worst-case slack** between clocks— the worst-case slack among all endpoint-unique paths in the transfer. • **Total negative slack** between clocks— the sum of all negative slacks among all endpoint-unique paths in this transfer. • **Tightest relationship** between clocks— the lowest-value setup, hold, recovery, or removal relationship between the two clocks in this transfer.
Show Filters and **Show Legend**	Turns on or off Filters and **Legend**.

Figure 161. CDC Viewer Setup Transfers Report



Each block in the grid is a transfer cell. Each transfer cell uses color and text to display important details of the paths in the transfer. The color coding represents the following states:

Table 29. Transfer Cell Content

Cell Color	Color Legend
Black	Indicates no transfers. There are no paths crossing between the source and destination clock of this cell.
Green	Indicates passing timing. All timing paths in this transfer, that have not been cut, meet their timing requirements.
Red	Indicates failing timing. One or more of the timing paths in the transfer do not meet their timing requirements. If the transfer is between unrelated clocks, the paths likely require a synchronizer chain.
Blue	Indicates clock groups. The source and destination clocks of these transfers are cut by means of asynchronous clock groups.
Gray	Indicates a cut transfer. All paths in this transfer are cut by false paths. Therefore, timing analysis does not consider these paths.
Orange	Indicates inactive clocks. One of the clocks in the transfer is an inactive clock (with the `set_active_clocks` command). The Timing Analyzer ignores such transfers.

Right-click menus allow you to perform operations on transfer cells and clock headers. When the operation is a Timing Analyzer report or SDC command, a dialog box opens containing the contents of the transfer cell.

Table 30. Transfer Cell Right-Click Menus

Command	Description
Copy	Copies the contents of the transfer cell or clock header to the clipboard.
Report Timing	Reports timing. Not available for transfer cells with no valid paths (gray or black cells).
Report Endpoints	Reports endpoints. Not available for transfer cells with no cut paths (gray or black cells).
Report False Path	Reports false paths. Not available for transfer cells with no valid paths (black cells).
Report Exceptions	Reports exceptions. Only available for clock group transfers (blue cells).
Report Exceptions (with clock groups)	Reports exceptions with clock groups. Only available for clock group transfers (blue cells).
Set False Path	Sets a false path constraint.
Set Multicycle Path	Sets a multicycle path exception.
Set Min Delay	Sets a min delay constraint.
Set Max Delay	Sets a max delay constraint.
Set Clock Uncertainty	Sets a clock uncertainty constraint.

Table 31. Clock Header Right-Click Menus

Command	Description
Copy (include children)	Copies the name of the clock header, and the names of each of its derived clocks. This option only appears for clock headers with generated clocks.
Expand/Collapse All Rows/Columns	Shows or hides all derived clocks in the grid.

continued...

Command	Description
Create Slack Histogram	Generates a slack histogram report for the clock you select.
Report Timing From/To Clock	Generates a timing report for the clock you select. If you do not expand the clock to display derived clocks, the timing report includes all clocks that derive from the clock. To prevent this, expand the clock before right-clicking it.
Remove Clock(s)	Removes the clock you select from the design. If you do not expand the clock, timing analysis removes all clocks that derive from the clock.

You can view CDC Viewer output in any of the following formats:

- A report panel in the Timing Analyzer
- Output in the Timing Analyzer Tcl console
- A plain-text file
- An HTML file you can view in a web browser.

Related Information

- Report Asynchronous CDC on page 142
- Constraining CDC Paths on page 92

2.5.10. Report Asynchronous CDC

The Timing Analyzer's **Reports ➤ Clock Domain Crossings ➤ Report Asynchronous CDC...** command allows you to classify and report all asynchronous clock-domain-crossing (CDC) transfers in your design. Asynchronous CDCs include single-bit transfers, multibit transfers, and asynchronous reset CDCs. Designs often contain unintended CDCs or transfers. Use this report to ensure that the Timing Analyzer correctly detects all CDCs.

Table 32. Report Asynchronous CDC Information Settings

Option	Available Settings
Clocks	Filters the report to only show CDCs that originate from **From clock** and terminate at **To clock**.
Targets	Filters the report to only show CDCs that originate from **From** register and terminate at **To** register. Both the **From** and **To** must be registers. This option is optional with the **Clocks** option.
Entries	Limits the number of entries that are reported per CDC category
Detail	Chooses whether to show a summary of all CDC's or give detail on each individual CDC
CDC Categories	Specifies the CDC categories to be reported.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can overwrite or append the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

Figure 162. Report Asynchronous CDC Summary View

Asynchronous CDC Summary

<<Filter>>

	CDC Type	CDC Count	Total Transfer Width
1	Single-bit CDC		
1	Unsynchronized Transfer	5	5
2	Unconstrained Synchronizer	1	1
3	Synchronizer Dri...le Clock Domains	2	2
4	Intra-Clock False Path Synchronizer	2	2
5	Synchronizer Not...otected for MTBF	1	1
6	Compliant Synchronizer	11	11
7	Transfer Precede...binational Logic	0	0
2	Multi-bit CDC		
1	Unconstrained CE-type CDC Bus	2	5
2	Compliant CE-type CDC Bus	2	5
3	Unconstrained MUX-type CDC Bus	2	8
4	Compliant MUX-type CDC Bus	2	8
5	Unsynchronized Synchronizer Bus	2	8
6	Unconstrained Synchronizer Bus	1	2
7	Compliant Synchronizer Bus	1	4
8	Synchronizer Bu... Uneven Lengths	0	0
9	Synchronizer afte...th Control Signal	0	0
3	Asynchronous Reset CDC		
1	Unsynchronized Reset Synchronizer	1	1
2	Unconstrained Reset Synchronizer	1	1
3	Reset Synchroniz...Multiple Sources	1	1
4	Inactive Reset Synchronizer	1	1

The summary view of the Asynchronous CDC Report provides an overview of the number of CDC's that fall into each category in your design. CDC Count gives the number of topologies detected for a category. **Total Transfer Width** shows the total number of CDC crossings for a category. This value is different than **CDC Count** for multibit CDC's because each topology consists of multiple crossings.

The full view of the Asynchronous CDC report shows the source and destination registers and clocks for each detected CDC topology in your design. You may click on any row of this report. Clicking on a row that contains the name of a CDC category brings up the description for that category and associated Design Assistant rules that check for such topologies. Clicking on a row that contains a CDC displays detailed information on that CDC in the **CDC Statistics** table. The information available varies depending on the topology of the CDC.

Figure 163. Report Asynchronous CDC Full View

Asynchronous CDC Full Report

	CDC Type	Source Nodes	Destination Nodes
1	Single-bit CDC		
1	Unsynchronized Transfer (5)		
1	--	unsynch_chain|src_reg	unsynch_c...ynch_reg1
2	--	chain_with_...out|src_reg	chain_with...synch_reg1
3	--	logic_within...hain|src_reg	logic_withi...|synch_reg1
4	--	logic_within...hain|src_reg	logic_withi...|synch_reg1
5	--	synch_reg_o...ain|src_reg	synch_reg_...synch_reg1
2	Unconstrained Synchronizer (1)		
1	--	unconstrain...ain|src_reg	unconstrai...synch_reg1
3	Synchronizer Drive... Clock Domains (2)		
1	--	multi_clock_...ain|src_reg1	multi_cloc...synch_reg1
2	--	comb_logic_...ain|src_reg	comb_logic...synch_reg1
4	Intra-Clock False Path Synchronizer (2)		
1	--	intra_clock_...t_chain|rst1	intra_clock..._chain|rst2
2	--	intra_clock_...hain|src_reg	intra_clock...|synch_reg1

Figure 164. Statistics Table Showing Detailed Information on Each CDC

Unsynchronized Transfer #4

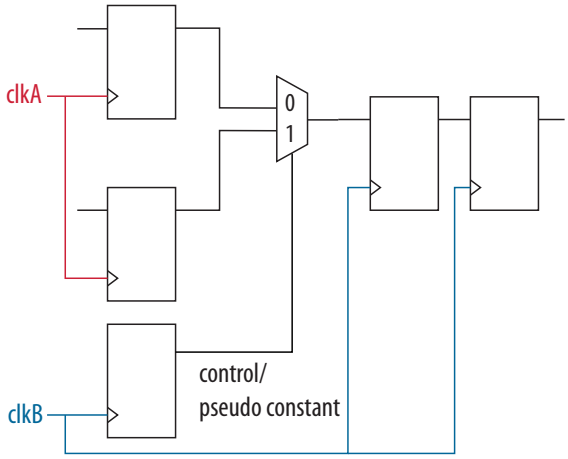
CDC Statistics

	Property	Value
1	Number of Source Registers	1
2	Source Registers	
1	logic_within_forced_chain|src_reg	
3	Number of Discovered Synchronization Registers	2
4	Discovered Synchronization Register	Synchr...Number
1	logic_within_forced_chain|synch_reg1	1
2	logic_within_forced_chain|synch_reg2	2
5	Method of Synchronizer Identification	User Specified
6	Chain Length Requested by User	3
1	User-requested chain le...rs in the synchronizer.	
7	The destination register ...ata synchronizer because:	
1	It has SYNCHRONIZER_I...nous reset connected.	
8	The synchronizer is termi...he last register because:	
1	The following nodes ca...ynchronizer registers:	
1	dataOut[91]	
9	Chain Length Protected	1
10	Protecting less synchron...than discovered because:	
1	The synchronizer regis...mode is set to FORCED:	
1	logic_within_forced_chain|synch_reg1	

In the Asynchronous CDC Full Report panel, the All Source Clocks (Synch/Asynch) column displays all clocks driving all source nodes of a synchronizer chain, regardless of whether the transfers are synchronous or asynchronous. There can be asynchronous and synchronous transfers into the head of a synchronizer chain if the paths go through combinational logic before the head of the synchronizer chain. For example, Figure xx shows a circuit with both asynchronous and synchronous transfers

into a synchronizer chain. For this example, the Asynchronous CDC Full Report lists both `clkA` and `clkB` as source clocks of the synchronizer chain, even though `clkB` is a synchronous transfer.

Figure 165. Circuit with Asynchronous and Synchronous Transfers to a Synchronizer Chain



Related Information

- Report CDC Viewer on page 139
- Constraining CDC Paths on page 92

2.5.11. Report Logic Depth

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Logic Depth...** command allows you to report the number of logic levels within a clock domain. This value typically corresponds to the number of look-up tables (LUTs) that a path passes through.

The equivalent scripting command is `report_design_metrics -logic_depth`. **Report Logic Depth** shows the distribution of logic depth among the critical paths, allowing you to identify areas where you can reduce logic levels in your RTL.

Figure 166. Report Logic Depth (Histogram)

Figure 167. Report Paths of Depth 3

Call report logic depth by topology for each clock, intraclock only.

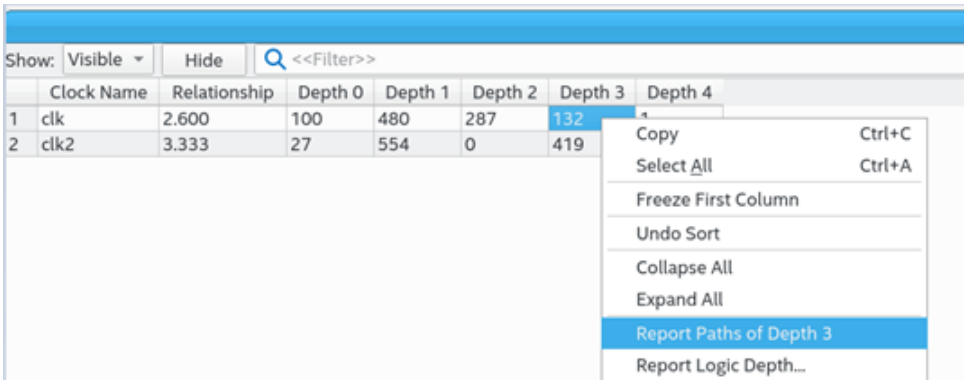
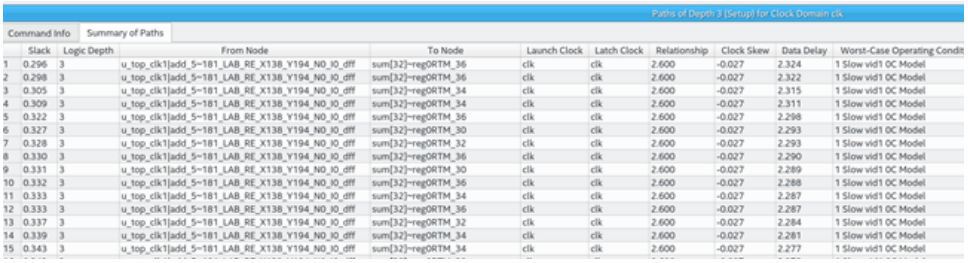


Figure 168. Summary of Paths

Close timing with accurate histogram cross probing.



You can specify various options to customize the reporting.

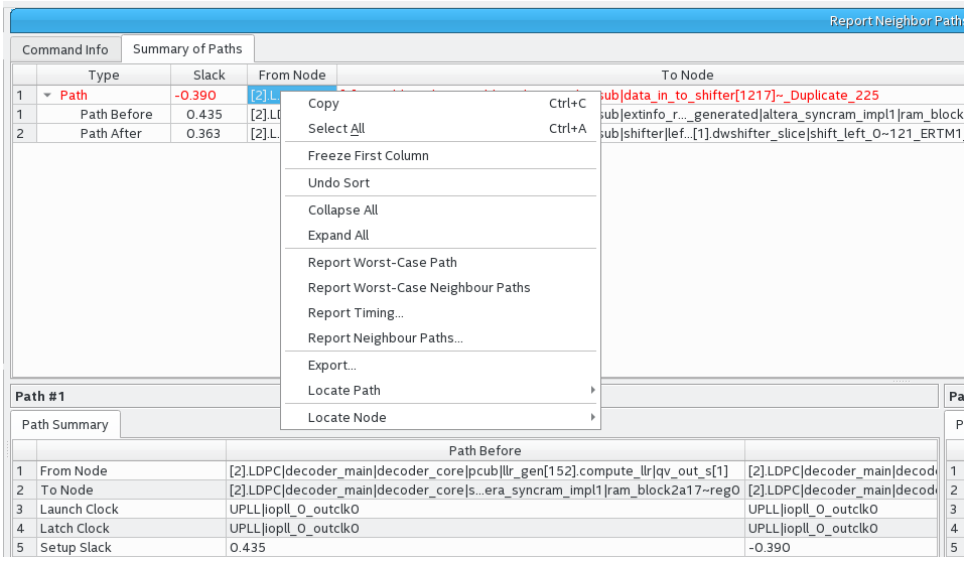
Table 33. Report Logic Depth Settings

Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Targets	Specifies the target node for **From Clock** and **To Clock** to report logic depth with only those endpoints. Specify an I/O or register name or I/O port for this option. The field also supports wildcard characters. When the **From**, **To**, or **Through** boxes are empty, the Timing Analyzer assumes all possible targets in the device. The **Through** option limits the report for paths that pass through combinatorial logic, or a particular pin on a cell.
Analysis type	The **Setup**, **Hold**, **Recovery**, and **Removal** analyses report the logic depths of the top *X* paths by slack. **Topology** analysis reports the logic depths of the top *X* paths by logic depth.
Paths	Specifies the number of paths to display by endpoint and slack level. The default value for **Report number of paths** is 10, otherwise, the report can be very long. Enable **Pairs only** to list only one path for each pair of source and destination. Limit further with **Maximum number of paths per endpoints**. You can also filter paths by entering a value in the **Maximum slack limit** field.
Detail	Specify whether to display on **Histogram** or full **Path** level of detail.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

2.5.12. Report Neighbor Paths

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Neighbor Paths...** command helps you to determine the root cause of critical paths (for example, high logic level, retiming limitation, sub-optimal placement, I/O column crossing, hold fix-up, time borrowing, or others). The equivalent scripting command is `report_design_metrics -neighbor_paths`.

Figure 169. Report Neighbor Paths Report



Report Neighbor Paths reports the most timing-critical paths in the design, including associated slack, additional path summary information, and path bounding boxes. **Report Neighbor Paths** shows the most timing-critical **Path Before** and **Path After** each critical **Path**. You can optionally view multiple before and after paths. Retiming or logic balancing of the **Path** can simplify timing closure if there is negative slack on the **Path**, but positive slack on the **Path Before** or **Path After**.

Table 34. Report Neighbor Path Dialog Box Settings

Option	Description
Clocks	**From Clock** and **To Clock** filter paths in the report to show only the launching or latching clocks you specify.
Targets	Specifies the target node for **From Clock** and **To Clock** to report neighbor paths with only those endpoints. Specify an I/O or register name or I/O port for this option. The field also supports wildcard characters. When the **From**, **To**, or **Through** boxes are empty, the Timing Analyzer assumes all possible targets in the device. The **Through** option limits the report for paths that pass through combinatorial logic, or a particular pin on a cell.
Analysis type	The **Analysis type** options are **Setup**, **Hold**, **Recovery**, or **Removal**. The Timing Analyzer reports the results for the type of analysis you select.
Paths	Specifies the number of paths to display by endpoint and slack level. The default value for **Report number of paths** is 10, otherwise, the report can be very long. Enable **Pairs only** to list only one path for each pair of source and destination. Limit further with **Maximum number of paths per endpoints**. You can also filter paths by entering a value in the **Maximum slack limit** field.

continued...

Option	Description
Report Number of Neighbor Paths	Specifies the number of neighbor paths to report, allowing you to view a number of the top adjacent paths entering the critical path, and the top paths exiting the critical path.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.
Extra Info	Specifies extra info.

2.5.13. Report Register Spread

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Register Spread...** command analyzes the final placement to identify registers with sinks pulling them in various directions. These registers are potential candidates for duplication. The equivalent scripting command is `report_register_spread`.

Registers that drive in opposite directions and connect to high fan-out can have placement-warping effects on the floorplan that impact f_{MAX}. The placement-warping may not cause timing failures. Therefore, you can view this report to identify such registers. Taking steps to address the registers listed in the report can make placement of the design easier and improve f_{MAX} performance.

You can automate duplication of registers with the `DUPLICATE_REGISTER` and `DUPLICATE_HIERARCHY_DEPTH` `.qsf` assignments, or you can manually modify RTL to duplicate registers or refactor logic. Refer to "Automatic Register Duplication: Hierarchical Proximity" in *Quartus Prime Pro Edition User Guide: Design Optimization*.

Figure 170. Report Register Spread Report

Report Register Spread

Show: Visible | Hide | <<Filter>>

Register Name	Register_Location	Number of Endpoints	Endpoint Centroid	Total Distance of Endpoints
counter_a[0]~ERTM	(69, 67)	78	(69, 67)	117.0
counter_a[1]~ERTM	(68, 67)	39	(68, 66)	21.4
counter_b[1]~ERTM	(70, 67)	39	(70, 67)	21.4
counter_b[2]~ERTM	(70, 67)	38	(70, 67)	20.9
counter_a[2]~ERTM	(68, 67)	38	(68, 66)	20.9
counter_b[3]~ERTM	(70, 67)	37	(70, 66)	20.4

You can specify various options to customize the report.

Table 35. Report Register Spread Settings

Option	Available Settings
Spread Type	Specifies the type of spread data in the report:

continued...

Option	Available Settings
	• **Tension**—reports the sum over each sink of the distance from it to the centroid of all the sinks. • **Angle**—reports how far around the source register the fan-outs wrap, expressed from 0 to 360 degrees. This value corresponds to 360 minus the maximum angle between any two angularly adjacent sinks. This metric complements **Tension** by identifying registers which are surrounded by their sinks in all directions, and not those registers only being pulled in a few directions. • **Span**—reports the maximum 1-dimensional delta between the left bottom-most sink and the right top-most sink. • **Area**—reports the coverage of the sinks by number of LABs on the FPGA device. This option multiplies the span of the sinks in both X- and Y- dimensions. This metric complements **Span** by incorporating both dimensional spans of the sinks, and not only the maximum sink. • **Count**—reports registers with the largest sink counts.
Sink Type	Specifies the type of sink in the report: • **Endpoint**—the nodes (usually registers) that terminate timing paths from a register. • **Immediate Fanout**—the immediately connected nodes of the register. For example, lookup tables, other registers, RAM, or DSP blocks.
From Clock	Filters paths in the report to show only the launching clocks you specify.
To Clock	Filters paths in the report to show only the latching clocks you specify, allowing you to debug one clock at a time.
Report number of registers	Specifies the number of registers to display in the report. The default value for **Report number of registers** is 10.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

Figure 171. Report Register Spread Types

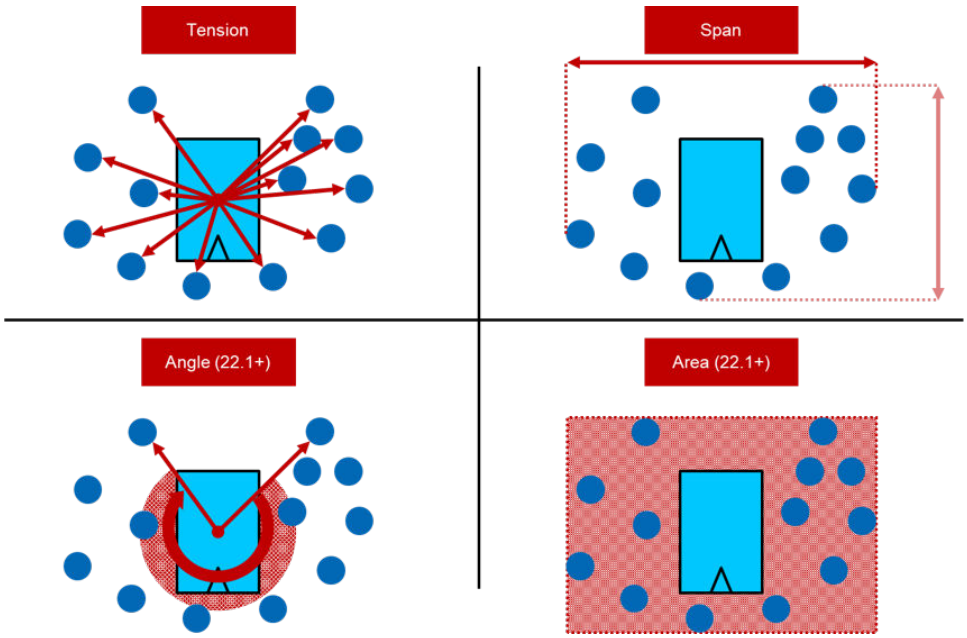
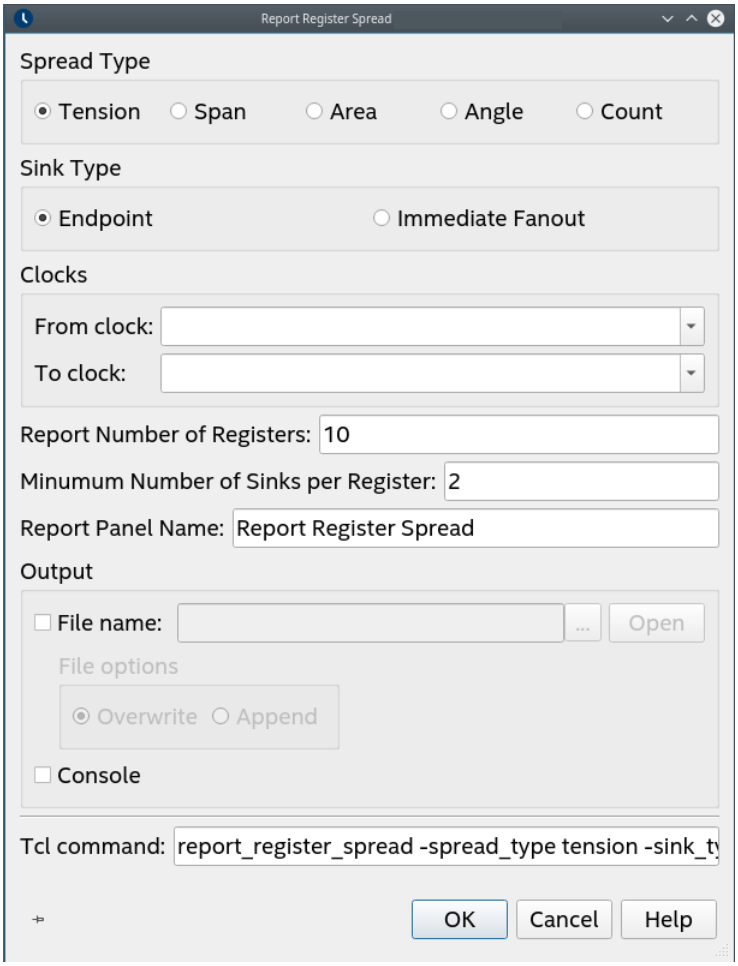


Figure 172. Report Register Spread Dialog Box



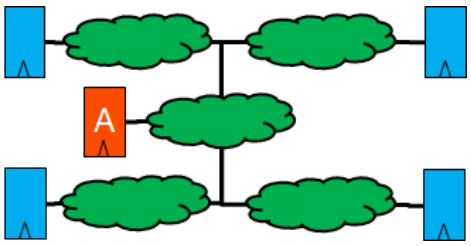
Related Information

Automatic Register Duplication: Hierarchical Proximity, Quartus Prime Pro Edition

2.5.13.1. Registers with High Timing Path Endpoint Tension

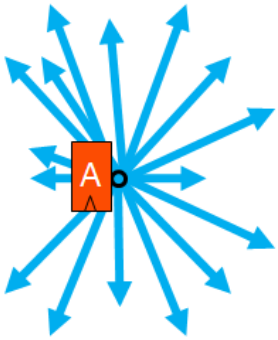
Timing path endpoints are the nodes (usually registers) that terminate timing paths from a register. The Timing path endpoint is equivalent to the nodes that the `get_fanouts` command returns, or the overall set of nodes that appear as a "From Node" after running the `report_timing` command. Register duplication is necessary, but not always sufficient, in helping to distribute the signal more efficiently. In addition, you may need to duplicate or restructure any intermediate logic before duplicating the register.

Figure 173. Register A has 4 Timing Path Endpoints



Tension is the sum over each sink of the distance from the sink to the centroid of all the sinks. The value of tension is therefore dependent on the number of sinks. Register duplication can help to break up these clouds, since they may be the result of the placement solution getting "warped" by the presence of the register.

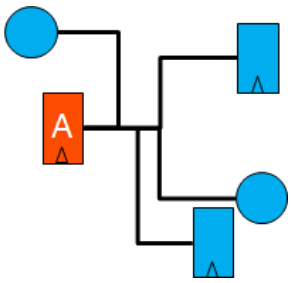
Figure 174. Register A has High Tension



2.5.13.2. Registers with High Immediate Fan-Out Tension

There are two **Sink Type** options: **Endpoint** and **Immediate Fanout**. The immediate fan-outs are the immediately connected nodes (lookup tables, other registers, RAM or DSP blocks, and others) of the register. This fan-out is equivalent to fan-outs that the Chip Planner displays, and in various high fan-out reports. Register duplication directly distributes the immediate fan-outs of a register among the duplicates.

Figure 175. Register A has High Immediate Fan-Out



2.5.14. Report Route Net of Interest

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Route Net of Interest...** command allows you to report the nets that require the most effort from the router. The report shows the percentage of total router effort for the nets reported. The equivalent scripting command is `report_route_net_of_interest`.

This report allows you to identify nets that should not require significant router effort. For example, you might expect that low speed management interface nets are not timing critical, and therefore not require much router effort. However, if **Report Route Net of Interest** reports that some nets in the low speed management interface require significant effort from the router, you can investigate that further. The investigation can determine whether the timing constraints are correct, whether the fan-out is significant and can reduce through driver duplication, or whether the net passes through congested areas.

Figure 176. Report Route Net of Interest Report

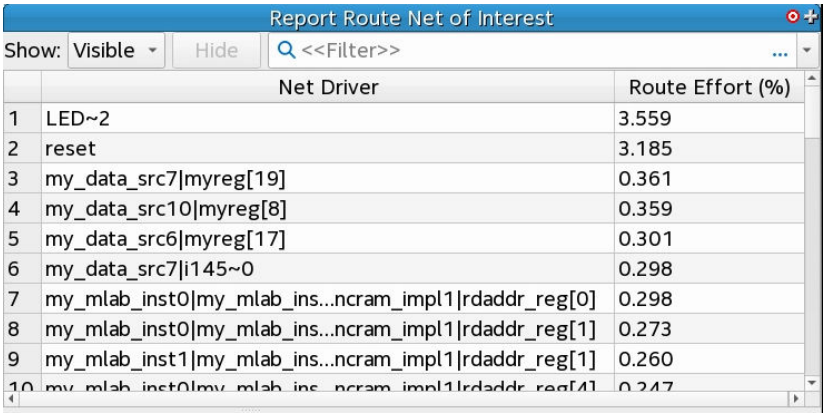
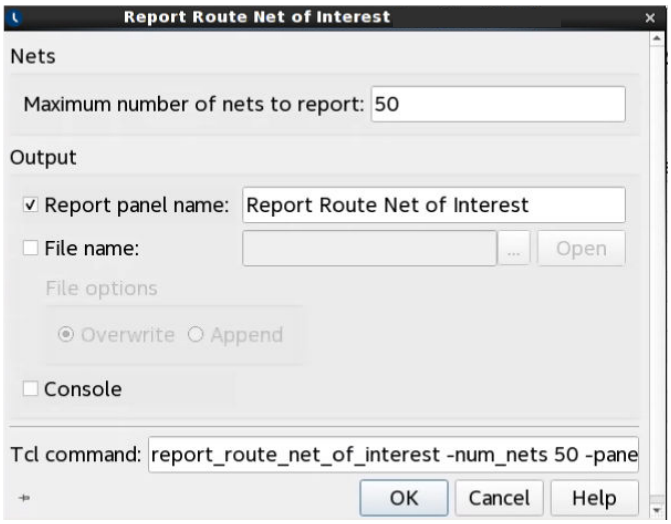


Figure 177. Report Route Net of Interest Dialog Box



From the Route Net of Interest Report in the Timing Analyzer GUI, you can right-click on any net and run Report Timing for more details about the net, its slack, and any of the net's paths.

Table 36. Report Route Net of Interest Settings

Option	Available Settings
Nets	Specifies the **Maximum number of nets to report**. The default value is 50.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

2.5.15. Report Retiming Restrictions

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Retiming Restrictions...** command allows you to report the occurrences of design conditions that restrict Hyper-Retiming, such as Power-up "Care" restrictions, and don't touch or preserve attributes for each port. You can refer to this report to improve the circuit and remove retiming restrictions that limit circuit performance. `report_retiming_restrictions` is the equivalent scripting command.

Figure 178. Report Retiming Restrictions Report

Report Retiming Restrictions

<<Filter>>

	Compilation Hierarchy Node	Register is part of a synchronization chain	Preserve assignn
1	|	114 (0)	292 (50)
1	|my_data_src7|	21 (21)	0 (0)
1	|myreg[*]	20 (20)	0 (0)
2	|enable_reg	1 (1)	0 (0)
2	|my_data_src8|	21 (21)	0 (0)
1	|myreg[*]	20 (20)	0 (0)
2	|enable_reg	1 (1)	0 (0)
3	|my_data_src5|	19 (19)	0 (0)
1	|myreg[*]	18 (18)	0 (0)
2	|enable_reg	1 (1)	0 (0)
4	|my_data_src6|	19 (19)	0 (0)

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of retiming restrictions in the specific entity alone. The numbers listed outside of parentheses indicate the number of retiming restrictions in the specific entity and all of its sub-entities in the hierarchy.

For table entries with two number values, the number in parentheses indicates the number of retiming restrictions in the specific entity alone. The number listed outside of parentheses indicates the number of retiming restrictions in the specific entity and all of its sub-entities in the `hierarchy`.

Related Information

Retiming Restrictions and Workarounds, Hyperflex Architecture High-Performance Design Handbook

2.5.16. Report Register Statistics

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Register Statistics** command allows you to report the number of synchronous and asynchronous resets, hyper registers, and registers with clock enables in the design. You can use this information, combined with timing slack, congestion, and other analysis reports, to identify timing-critical parts of your design that can have resets removed or control schemes changed to meet timing requirements more efficiently.

Figure 179. Report Register Statistics (Report Truncated)

Report Register Statistics

<<Filter>> (use !<string> to invert filter)

	Compilation Hierarchy Node	Register Count	Without a Clock	Unique Clocks	Hyper-Register Count	Synchronous Reset	Asynchronous Reset
1	|	3645 (0)	0 (0)	2 (0)	0 (0)	311 (0)	777 (0)
1	|auto_fab_0|	88 (0)	0 (0)	1 (0)	0 (0)	7 (0)	38 (0)
1	|alt_sld_fab_0|	88 (0)	0 (0)	1 (0)	0 (0)	7 (0)	38 (0)
2	|intel_niosv_m_0|	2789 (0)	0 (0)	2 (0)	0 (0)	235 (0)	658 (0)

Truncated Report Columns:

Clock Enable	Synchronous Load	Without a Control Signal	Full Hierarchy Name
2472 (0)	0 (0)	726 (0)	|
63 (0)	0 (0)	12 (0)	auto_fab_0
63 (0)	0 (0)	12 (0)	auto_fab...ld_fab_0
1961 (0)	0 (0)	454 (0)	intel_niosv_m_0

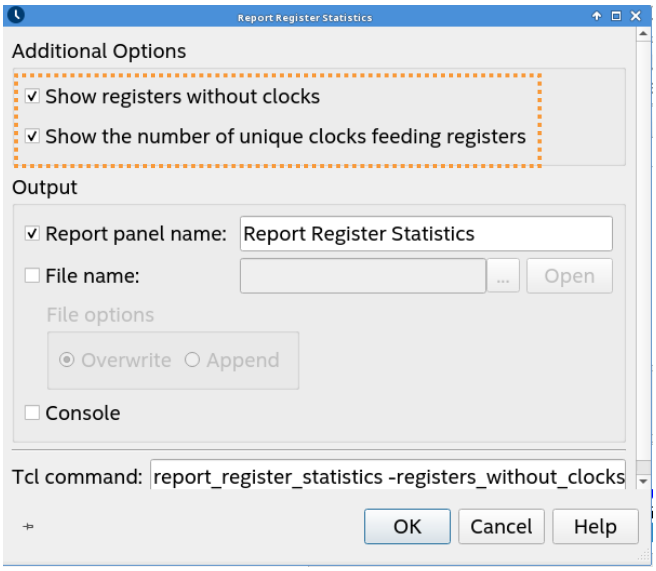
Note:

- This report works similarly in both post-synthesis Early Timing Analysis and post-fit timing analysis. However, the report's **Without a Clock** column is more helpful for Early Timing Analysis because you typically do not apply conventional SDCs for Early Timing Analysis.
- Clocks generated from `derive_clocks` commands do not count as user clocks.
- The report's **Without a Control Signal** column identifies registers that have no corresponding control signal.
- The report's **Synchronous Load** column identifies any synchronous load that can apply to Arria 10 devices only.

The **Without a Clock** column informs you of the number of registers where no defined clock feeds the registers in the hierarchy shown in the **Register Count** column. A value of 0 in this column suggests that your design has SDC-defined clocks feeding registers in the design. The **Unique Clocks** column indicates the number of unique SDC-defined clocks feeding registers in the hierarchy identified by the

Register Count. To view these columns, enable **Show registers without clocks** and **Show the number of unique clocks feeding registers** additional options in the dialog that displays when you run the report, as shown in the following image:

Figure 180. Report Register Statistics Additional Options Dialog

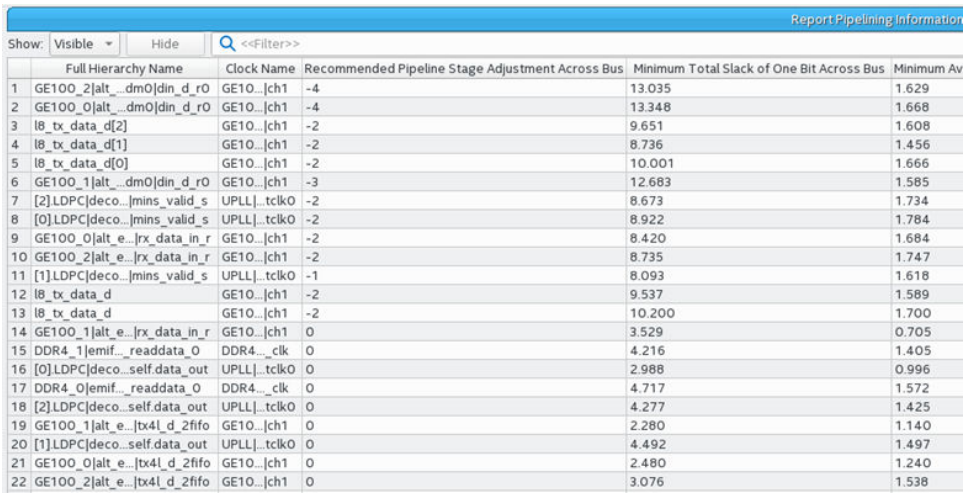


2.5.17. Report Pipelining Information

The Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Pipelining Information...** command allows you to generate a report that can help you to identify potential areas of over-pipelining in your design. Excessive pipelining unnecessarily consumes area. The equivalent scripting command is `report_pipelining_info`.

Report Pipelining Information... does not perform any functional analysis in making the recommended pipeline stage adjustment. You must be aware of any potential functional changes from removing pipeline stages. There may be circumstances when all the stages in a register pipeline are necessary for functional reasons. The report helps to identify location with more registers than necessary for covering distance.

Figure 181. Report Pipelining Information Report

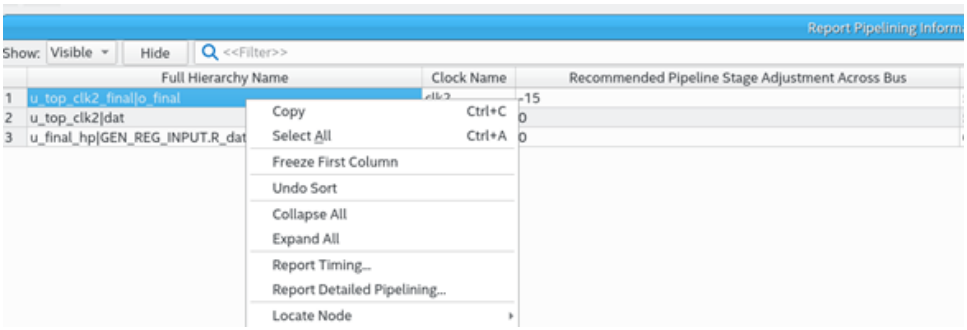


Report Pipelining Information

Show: Visible | Hide | <<Filter>>

	Full Hierarchy Name	Clock Name	Recommended Pipeline Stage Adjustment Across Bus	Minimum Total Slack of One Bit Across Bus	Minimum Av
1	GE100_2|alt_...dm0|din_d_r0	GE10...|ch1	-4	13.035	1.629
2	GE100_0|alt_...dm0|din_d_r0	GE10...|ch1	-4	13.348	1.668
3	l8_tx_data_d[2]	GE10...|ch1	-2	9.651	1.608
4	l8_tx_data_d[1]	GE10...|ch1	-2	8.736	1.456
5	l8_tx_data_d[0]	GE10...|ch1	-2	10.001	1.666
6	GE100_1|alt_...dm0|din_d_r0	GE10...|ch1	-3	12.683	1.585
7	[2].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-2	8.673	1.734
8	[0].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-2	8.922	1.784
9	GE100_0|alt_e...|rx_data_in_r	GE10...|ch1	-2	8.420	1.684
10	GE100_2|alt_e...|rx_data_in_r	GE10...|ch1	-2	8.735	1.747
11	[1].LDPC|deco...|mins_valid_s	UPLL|...tclk0	-1	8.093	1.618
12	l8_tx_data_d	GE10...|ch1	-2	9.537	1.589
13	l8_tx_data_d	GE10...|ch1	-2	10.200	1.700
14	GE100_1|alt_e...|rx_data_in_r	GE10...|ch1	0	3.529	0.705
15	DDR4_1|emif..._readdata_0	DDR4..._clk	0	4.216	1.405
16	[0].LDPC|deco...self.data_out	UPLL|...tclk0	0	2.988	0.996
17	DDR4_0|emif..._readdata_0	DDR4..._clk	0	4.717	1.572
18	[2].LDPC|deco...self.data_out	UPLL|...tclk0	0	4.277	1.425
19	GE100_1|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	2.280	1.140
20	[1].LDPC|deco...self.data_out	UPLL|...tclk0	0	4.492	1.497
21	GE100_0|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	2.480	1.240
22	GE100_2|alt_e...|tx4l_d_2fifo	GE10...|ch1	0	3.076	1.538

Figure 182. Report Detailed Pipelining



Report Pipelining Inform

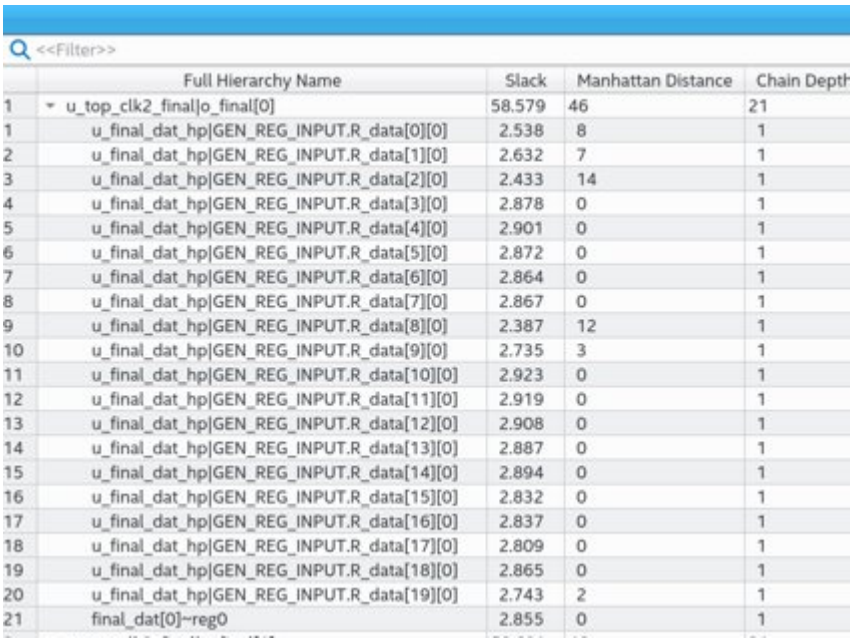
Show: Visible | Hide | <<Filter>>

	Full Hierarchy Name	Clock Name	Recommended Pipeline Stage Adjustment Across Bus
1	u_top_clk2_final|o_final		-15
2	u_top_clk2|dat		0
3	u_final_hp|GEN_REG_INPUT.R_dat		0

Copy Ctrl+C
Select All Ctrl+A
Freeze First Column
Undo Sort
Collapse All
Expand All
Report Timing...
Report Detailed Pipelining...
Locate Node

The detailed report shows every register in a tree structure. Over- or under-pipelining recommendations are in the main report. The following shows every single register inside the bus chain in a tree structure:

Figure 183. Detailed Pipelining Result



<<Filter>>

	Full Hierarchy Name	Slack	Manhattan Distance	Chain Depth
1	u_top_clk2_final|o_final[0]	58.579	46	21
1	u_final_dat_hp|GEN_REG_INPUT.R_data[0][0]	2.538	8	1
2	u_final_dat_hp|GEN_REG_INPUT.R_data[1][0]	2.632	7	1
3	u_final_dat_hp|GEN_REG_INPUT.R_data[2][0]	2.433	14	1
4	u_final_dat_hp|GEN_REG_INPUT.R_data[3][0]	2.878	0	1
5	u_final_dat_hp|GEN_REG_INPUT.R_data[4][0]	2.901	0	1
6	u_final_dat_hp|GEN_REG_INPUT.R_data[5][0]	2.872	0	1
7	u_final_dat_hp|GEN_REG_INPUT.R_data[6][0]	2.864	0	1
8	u_final_dat_hp|GEN_REG_INPUT.R_data[7][0]	2.867	0	1
9	u_final_dat_hp|GEN_REG_INPUT.R_data[8][0]	2.387	12	1
10	u_final_dat_hp|GEN_REG_INPUT.R_data[9][0]	2.735	3	1
11	u_final_dat_hp|GEN_REG_INPUT.R_data[10][0]	2.923	0	1
12	u_final_dat_hp|GEN_REG_INPUT.R_data[11][0]	2.919	0	1
13	u_final_dat_hp|GEN_REG_INPUT.R_data[12][0]	2.908	0	1
14	u_final_dat_hp|GEN_REG_INPUT.R_data[13][0]	2.887	0	1
15	u_final_dat_hp|GEN_REG_INPUT.R_data[14][0]	2.894	0	1
16	u_final_dat_hp|GEN_REG_INPUT.R_data[15][0]	2.832	0	1
17	u_final_dat_hp|GEN_REG_INPUT.R_data[16][0]	2.837	0	1
18	u_final_dat_hp|GEN_REG_INPUT.R_data[17][0]	2.809	0	1
19	u_final_dat_hp|GEN_REG_INPUT.R_data[18][0]	2.865	0	1
20	u_final_dat_hp|GEN_REG_INPUT.R_data[19][0]	2.743	2	1
21	final_dat[0]~reg0	2.855	0	1

To help identify potential over-pipelining, **Report Pipelining Information** reports:

- The recommended pipeline stage adjustment across bus
- The minimum total slack of one bit across bus
- The minimum average slack of one bit across bus
- The distance between the registers
- The width of buses in your design
- The number of sequential registers
- The number of registers on the bus

The Recommended Pipeline Stage Adjustment Across Bus reports the number of registers that you can remove from the bus for each bit. The Average Distance Per Stage, Max Distance Per Stage, and Min Distance Per Stage columns report the Manhattan distance measured in logic array blocks (LABs). The Bus Average Depth, Bus Max Depth, and Bus Min Depth columns report the number of sequential, single fan-out registers. For registers that have more than one clock source, the report lists the fastest one.

The **1+** sign under Recommended Pipeline Stage Adjustment Across Bus column means that the bus might need to add more registers to meet timing requirement. Refer to the Fast Forward Timing Closure Recommendations report.

If the report identifies a large register chain with multiple sequential registers, and the distance between registers is low, that condition can suggest over-pipelining. You may be able to remove some registers to recover some of the device area and reduce congestion.

The following options are available for this report:

Figure 184. Report Pipelining Information Dialog Box

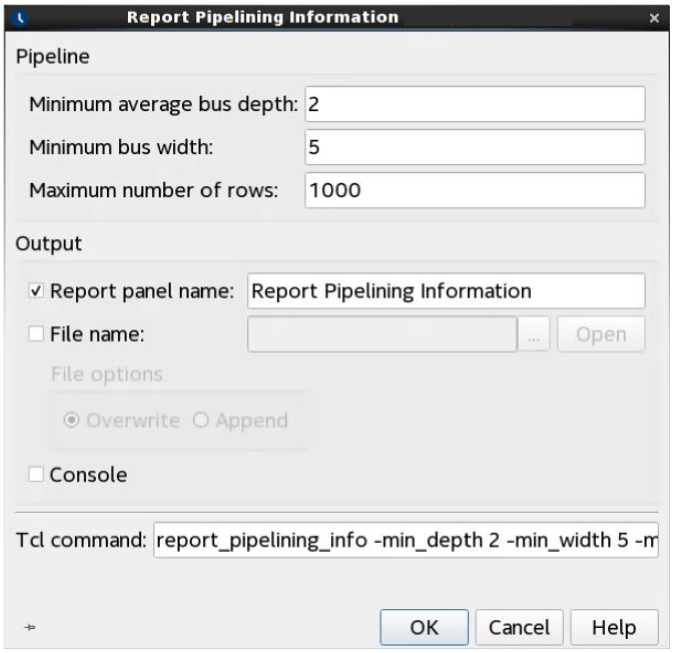


Table 37. Report Pipelining Information Settings

Option	Available Settings
Pipeline	Specifies the thresholds for reporting a register pipeline. You can define the **Minimum average bus depth**, the **Minimum bus width**, and the **Maximum number of rows** that the report includes.
Report panel name	Specifies the name of the report panel. You can optionally enable **File name** to write the information to a file. If you append `.htm` or `.html` as a suffix, the Timing Analyzer produces the report as HTML. If you enable **File name**, you can **Overwrite** or **Append** the file with latest data.
Tcl command	Displays the Tcl syntax that corresponds with the GUI options you select. You can copy the command from the **Console** into a Tcl file.

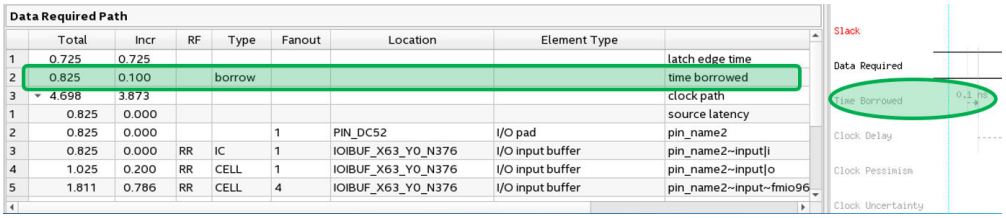
2.5.18. Report Time Borrowing Data

You can run the Timing Analyzer's **Reports ➤ Timing Slack ➤ Report Timing...** command to generate a report showing time borrowing data. The equivalent scripting command is `report_timing` (with specific arguments).

The Timing Analyzer reports time borrowing data for the **Data Arrival Path** or **Data Required Path**, according to whether borrowing occurs at the destination or the source.

Figure 185. Report for Time Borrowing At Destination

The following report shows 100ps borrowed on the Data Required path. The setup slack improves on incoming paths, at the expense of worse setup slack for outgoing paths.

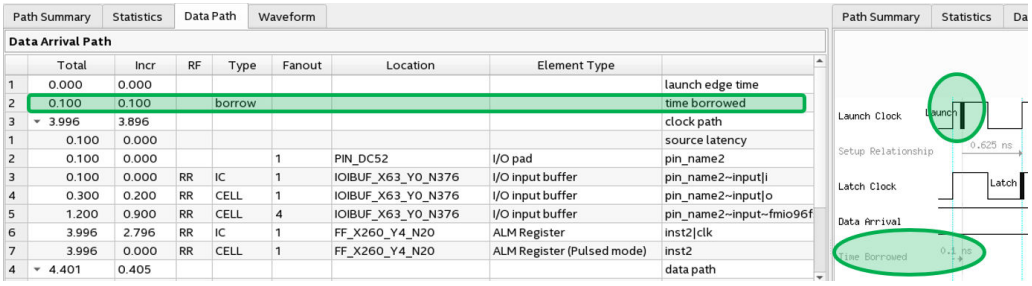


Data Required Path

	Total	Incr	RF	Type	Fanout	Location	Element Type	
1	0.725	0.725						latch edge time
2	0.825	0.100		borrow				time borrowed
3	4.698	3.873						clock path
1	0.825	0.000						source latency
2	0.825	0.000			1	PIN_DC52	I/O pad	pin_name2
3	0.825	0.000	RR	IC	1	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input|i
4	1.025	0.200	RR	CELL	1	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input|o
5	1.811	0.786	RR	CELL	4	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input~fmio96

Figure 186. Report for Time Borrowing At Source

The following report shows 100ps borrowed on the Data Arrival Path, worsening the setup relationship for the outgoing path.



Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	Element Type	
1	0.000	0.000						launch edge time
2	0.100	0.100		borrow				time borrowed
3	3.996	3.896						clock path
1	0.100	0.000						source latency
2	0.100	0.000			1	PIN_DC52	I/O pad	pin_name2
3	0.100	0.000	RR	IC	1	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input|i
4	0.300	0.200	RR	CELL	1	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input|o
5	1.200	0.900	RR	CELL	4	IOIBUF_X63_Y0_N376	I/O input buffer	pin_name2~input~fmio96f
6	3.996	2.796	RR	IC	1	FF_X260_Y4_N20	ALM Register	inst2|clk
7	3.996	0.000	RR	CELL	1	FF_X260_Y4_N20	ALM Register (Pulsed mode)	inst2
4	4.401	0.405						data path

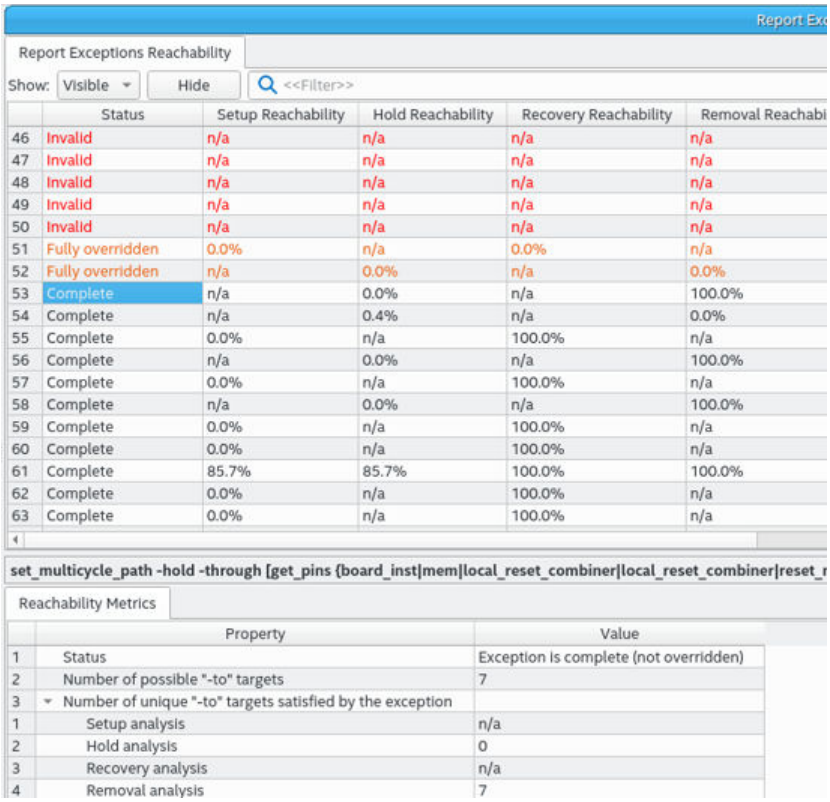
Related Information

Time Borrowing on page 19

2.5.19. Report Exceptions and Exceptions Reachability

The Timing Analyzer's **Reports ➤ Constraint Diagnostics ➤ Report Exceptions...** command allows you to report all exceptions to default timing analysis conditions, as specified by the **Set False Path**, **Set Multicycle Path**, **Set Minimum Delay**, or **Set Maximum Delay** commands (and the corresponding Tcl commands: `set_false_path`, `set_multicycle_path`, `set_min_delay`, and `set_max_delay`.)

Figure 187. Report Exceptions Reachability Report



Similarly, you can click on **Reports ➤ Constraint Diagnostics ➤ Report Exceptions Reachability...** to report the scope of exception constraints in your project. This report allows you to determine whether constraints apply fully or partially, are overridden, and whether the source and destination node are reachable for the constraint by providing a "reachability" percentage. Reachability is the percentage of paths to which the constraint applies. Low reachability indicates a constraint that may be too broad, potentially covering many unrelated items. High reachability indicates that the exception is very targeted.

2.5.20. Report Bottlenecks

You can run the Timing Analyzer's **Reports ➤ Design Metrics ➤ Report Bottlenecks...** command to list all nodes in a design ranked by specified criteria. The equivalent scripting command is `report_bottleneck`.

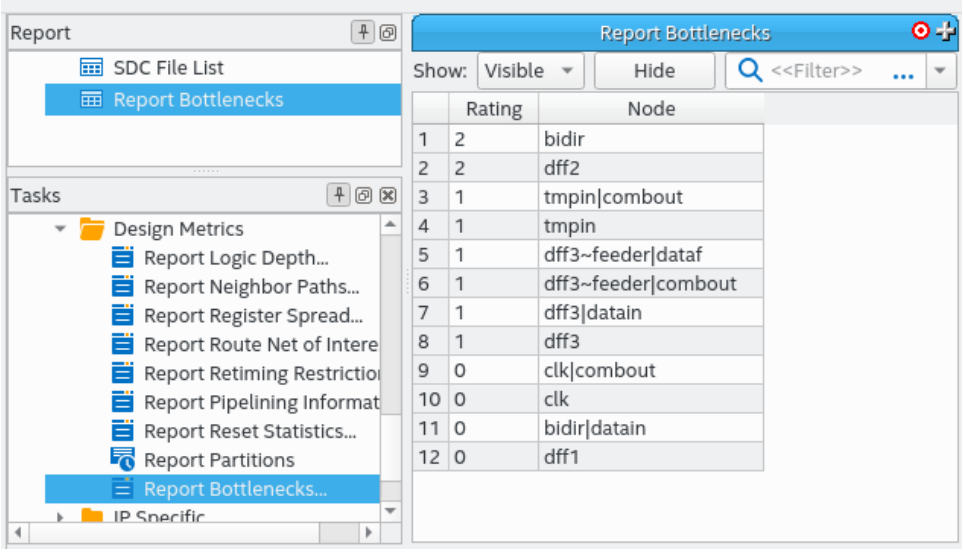
The following ranking criteria are pre-defined:

- `num_fpaths`—the number of paths that fail timing through a node.
- `num_fanins`—the number of fan-in edges from a node.
- `num_fanouts`—the number of fan-out edges from a node.
- `num_paths`—the number of paths through a node.
- `tns`—the total negative slack of all the paths through a node.

When using scripting, you can specify the paths for analysis by passing the result of any `get_timing_paths` call as the last argument to `report_bottleneck`. When using the GUI, the **Report Bottlenecks** dialog box handles this argument automatically. If you specify no paths, `report_bottleneck` analyzes the worst 1000 setup paths in the design by default.

You can direct the report output to the Tcl console (`-stdout`), the Timing Analyzer GUI (`-panel`), or to a combination of console and GUI.

Figure 188. Report Bottlenecks Rated on Number of Failing Paths Through a Node



2.5.20.1. Specifying Custom Bottleneck Criteria

You can optionally specify your own custom criteria for evaluating nodes based on the combination of the number of fan-outs, fan-ins, failing paths, and total paths.

To specify custom bottleneck criteria, follow these steps:

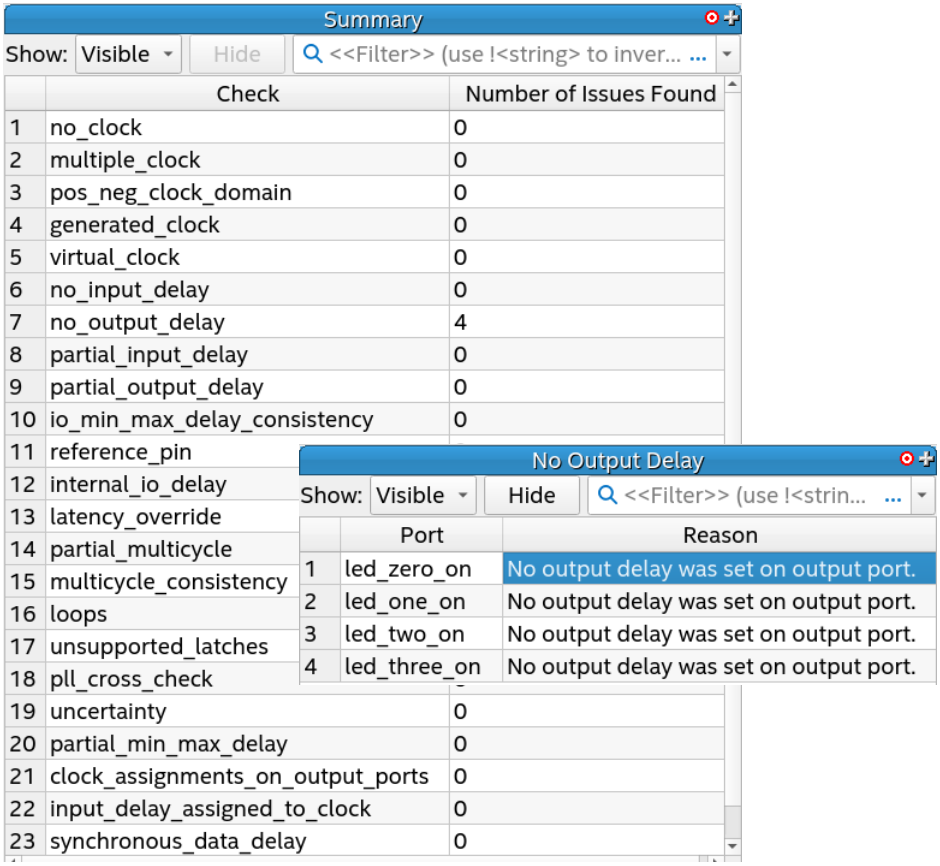
1. Create a Tcl procedure that takes one argument. For example, `arg`.
2. Use `upvar $arg metric` in the procedure.
3. Calculate the rating based on `$metric(tns)`, `$metric(num_fanouts)`, `$metric(num_fanins)`, and `$metric(num_fpaths)`.
4. Return the rating with `return $rating`.
5. Pass the name of your custom criteria procedure to `report_bottleneck` using the `-cmetric` option

2.5.21. Check Timing

The Timing Analyzer's **Reports ➤ Constraint Diagnostics ➤ Check Timing** command (`check_timing`) checks your design and constraint files for problems with design constraints.

Check Timing can perform a series of different checks based on the variables and options that you specify for the command. When using scripted methods, use the `-include` option to specify which checks to perform. You must run **Update Timing Netlist** (`update_timing_netlist`) before running **Check Timing** (`check_timing`).

Figure 189. Check Timing Report and No Output Delay Subreport



Check Timing can report the following data:

Table 38. Check Timing Report Data

Check Timing Data	Description
no_clock	Reports the registers that do not have at least one clock assignment at their clock pin, including any PLLs without a clock assignment.
multiple_clock	Reports the registers that have more than one clock at their clock pin. If multiple clocks reach a register clock pin, you must define which clock is used for analysis.
generated_clock	Reports the generated clocks that are invalid. Generated clocks must have a source that is triggered by a valid clock.
no_input_delay	Reports the input ports that are not clocks that have no input delay constraint.
no_output_delay	Reports the output ports that have no output delay constraint.

continued...

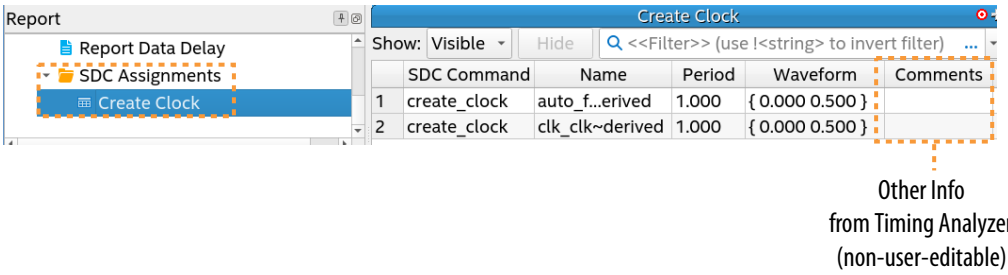
Check Timing Data	Description
partial_input_delay	Reports the input delays that lack a `rise-min`, `fall-min`, `rise-max`, and `fall-max` constraint set.
partial_output_delay	Reports the output delays that lack a `rise-min`, `fall-min`, `rise-max`, and `fall-max` constraint set.
io_min_max_delay_consistency	Reports the minimum delay values that you specify in `set_input_delay` or `set_output_delay` constraints that are not less than maximum delay values.
reference_pin	Reports the reference pins that you specify with `set_input_delay` and `set_output_delay` using the `-reference_pin` that are invalid. A `reference_pin` is only valid if the `-clock` option in the same `set_input_delay` or `set_output_delay` command matches the clock that is in the direct fan-in of the `reference_pin`. Being in the direct fan-in of the `reference_pin` means that there must be no keepers between the clock and the `reference_pin`.
latency_override	Reports the instances where the clock latency that you set on a port or pin overrides the more generic clock latency set on a clock. You can set clock latency on a clock, where the latency applies to all keepers clocked by the clock. You can also set clock latency on a port or pin, where the latency applies to registers in the fan-out of the port or pin.
loops	Reports the instances where there are strongly connected components in the netlist. These loops prevent a design from properly analysis. The loops check also reports whether loops exist but are marked so that they are not traversed by timing analysis.
latches	Reports the instances where latches are present in the design and warns that latches may not be analyzed properly. For best results, change your design to remove latches whenever possible.
pos_neg_clock_domain	Reports the instances where any register is clocked by both the rising and falling edges of the same clock. If this scenario is necessary, such as in a clock multiplexer, create two separate clocks that have similar settings and are assigned to the same node.
pll_cross_check	Reports the instances where clocks that are assigned to a PLL do not correspond properly with the PLL settings you define in design files. The subreport specifies the inconsistent settings, or an unmatched number of clocks associated with the PLL.
uncertainty	Reports the clock-to-clock transfers that do not have a clock uncertainty assignment set between the two clocks. If the target device family has `derive_clock_uncertainty` support, this report also includes the number of user-defined `set_clock_uncertainty` assignments that have less than recommended clock uncertainty value.
virtual_clock	Reports the unreferenced virtual clocks without constraint.
partial_multicycle	Reports the setup multicycle assignments without a corresponding hold multicycle assignment, and whether each hold muticycle assignment has a corresponding setup multicycle assignment.
multicycle_consistency	Reports the multicycle instances where a setup multicycle does not equal one less than the hold multicycle. Appropriate Hold multicycle assignments are usually one cycle less than setup multicycle assignments.
partial_min_max_delay	Reports the minimum delay assignments without a corresponding maximum delay assignment, and vice versa.
clock_assignments_on_output_ports	Reports the output ports that have clock assignments.
input_delay_assigned_to_clock	Reports the clocks with input delay values set. The Timing Analyzer ignores input delays set on clock ports because clock-as-data analysis takes precedence.
internal_io_delay	Reports the I/O delay constraints that have no specification for `-reference_pin` and `-source_latency_included`, and where `-clock` is a clock that is not assigned to a top level input or output port.

2.5.22. Report SDC

The Timing Analyzer's **Reports ➤ Constraint Diagnostics ➤ Report SDC** command generates the **SDC Assignments** report folder.

The **SDC Assignments** folder contains separately named reports for any SDC constraints found in the current project. For example, the Create Clock report appears in the **SDC Assignments** folder if the project includes any `create_clock` SDC constraint.

Figure 190. Create Clock Report Lists Clock Constraints in Current Project



In **SDC Assignment** reports, the **Comments** column may contain non-user-editable information reported by the Timing Analyzer.

Note: The **Comments** column in the **SDC Assignment** report has no relation to the SDC-on-RTL `-comment` argument that allows you to add comments to `.sdc` constraints. The SDC-on-RTL `-comment` does not appear in **SDC Assignment** reports.

The equivalent scripting command for **SDC Assignments** is `report_sdc`, for example:

```
report_sdc -panel_name sdc_report_panel
```

You can use the `-ignored` option to report any SDC constraints that the Timing Analyzer is ignoring and the reason for ignoring the constraint.

2.6. Scripting Timing Analysis

You can optionally use Tcl commands from the Quartus Prime software Tcl Application Programming Interface (API) to constrain, analyze, and collect timing information for your design. This section describes running the Timing Analyzer and setting constraints using Tcl commands. You can alternatively perform these same functions in the Timing Analyzer GUI. Tcl `.sdc` extensions provide additional methods for controlling timing analysis and reporting. The following Tcl packages support the Tcl timing analysis commands this chapter describes:

- `::quartus::sta`
- `::quartus::sdc`
- `::quartus::sdc_ext`

The Quartus Prime software provides the following two executables you can use for timing analysis and reporting:

- The `quartus_sta` executable—can perform timing analysis of your design without running the full Quartus Prime software GUI. If you use separate executables for various stages of design compilation (for example, `quartus_syn`, `quartus_fit`) you must use then use the `quartus_sta` executable to analyze timing for your placed and routed design. The `quartus_sta` executable also supports the interactive command-shell text-based timing reporting or scripted report generation.
- The `quartus_staw` executable—opens a GUI and supports interactive, graphical timing analysis and reporting.

Related Information

- ::quartus::sta
- ::quartus::staw
- ::quartus::sdc
- ::quartus::sdc_ext

2.6.1. The quartus_sta Executable

The `quartus_sta` executable allows you to run timing analysis without running the full Quartus Prime software GUI. The following methods are available:

- To run the Timing Analyzer in interactive command-shell mode, type the following at the command prompt:

```
quartus_sta -s
```

- To run timing analysis from a system command prompt, type the following command:

```
quartus_sta <options><project_name>
```

You can optionally use command line options available to perform iterative timing analysis on large designs. You can perform a less intensive analysis with `quartus_sta --mode=implement`. In this mode, the Quartus Prime software performs a reduced-corner timing analysis. When you achieve the desired result, you can use `quartus_sta --mode=finalize` to perform final Fitter optimizations and a full multi-corner timing analysis under all operating conditions.

Table 39. quartus_sta Command-Line Options

Command-Line Option	Description
`-h | --help`	Provides help information on `quartus_sta`.
`-t <script file> | --script=<script file>`	Sources the *<script file>*.
`-s | --shell`	Enters shell mode.
`--tcl_eval <tcl command>`	Evaluates the Tcl command *<tcl command>*.

continued...

Command-Line Option	Description
`--do_report_timing`	For all clocks in the design, run the following commands: `report_timing -npaths 1 -to_clock $clock` `report_timing -setup -npaths 1 -to_clock $clock` `report_timing -hold -npaths 1 -to_clock $clock` `report_timing -recovery -npaths 1 -to_clock $clock` `report_timing -removal -npaths 1 -to_clock $clock`
`--force_dat`	Forces an update of the project database with new delay information.
`--lower_priority`	Lowers the computing priority of the `quartus_sta` process.
`--post_map`	Uses the post-map database results.
`--sdc=<SDC file>`	Specifies the `.sdc` file to use.
`--report_script=<custom script>`	Specifies a custom report script to call.
`--speed=<value>`	Specifies the device speed grade used for timing analysis.
`-f <argument file>`	Specifies a file containing additional command-line arguments.
`-c <revision name> | --rev=<revision_name>`	Specifies which revision and its associated Quartus Prime Settings File (`.qsf`) to use.
`--multicorner`	Specifies that the Timing Analyzer generates all slack summary reports for both slow- and fast-corners.
`--multicorner[=on|off]`	Turns off multicorner timing analysis.
`--voltage=<value_in_mV>`	Specifies the device voltage, in mV used for timing analysis.
`--temperature=<value_in_C>`	Specifies the device temperature in degrees Celsius, used for timing analysis.
`--parallel [=<num_processors>]`	Specifies the number of computer processors to use on a multiprocessor system.
`--mode=implement|finalize`	Regulates whether Timing Analyzer performs a reduced-corner analysis for intermediate operations (`implement`),or a four-corner analysis for final Fitter optimization and placement (`finalize`).

2.6.2. The quartus_staw Executable

The `quartus_staw` executable opens a graphical interface to perform interactive timing reporting and analysis of your Quartus Prime project. The following methods are available:

- To run the Timing Analyzer as a stand-alone GUI application, type the following at the command prompt:

```
quartus_staw
```

`quartus_staw` can automatically open a project in the Timing Analyzer GUI if you specify the project name as an argument, as the following example shows. Specifying a project name as an argument saves you time because you do not have to browse to your project file after the GUI opens.

```
quartus_staw <project name>
```

You can specify a Tcl script that contains timing reporting and analysis commands to automatically run each time you open a project in the Timing Analyzer GUI. This feature allows you to easily define a uniform set of reports that the Timing Analyzer

GUI always generates, saving you time when you open a new project for interactive timing reporting. Use the `--report_script` option and specify a path to your report script:

```
quartus_staw --report_script=<custom script>
```

You must include only reporting and analysis commands in your report script. Do not include Tcl commands that open or close projects, create the timing netlist, read `.sdc` files, or update the timing netlist. The report script runs after the Timing Analyzer completely prepares the timing netlist for analysis.

You can optionally specify the directory and file names for the output of the reporting and analysis commands. The output of the reporting commands in the script are visible in the Timing Analyzer GUI, in the folder names you specify, when the reporting script completes. At that point, you can perform other reporting or analysis, as necessary.

2.6.3. Collection Commands

The Timing Analyzer supports collection commands that provide easy access to ports, pins, cells, or nodes in the design. Use collection commands with any constraints or Tcl commands specified in the Timing Analyzer.

Table 40. Collection Commands

Command	Collection Returned
`all_clocks`	All clocks in the design
`all_inputs`	All input ports in the design.
`all_outputs`	All output ports in the design.
`all_registers`	All registers in the design.
`get_cells`	Cells in the design. All cell names in the collection match the specified pattern. Wildcards can be used to select multiple cells at the same time.
`get_clocks`	Lists clocks in the design. When used as an argument to another command, such as the `-from` or `-to` of `set_multicycle_path`, each node in the clock represents all nodes clocked by the clocks in the collection. The default uses the specific node (even if the node is a clock) as the target of a command. The `-of_objects` option takes a node like a register and returns the clocks that drive it.
`get_nets`	Nets in the design. All net names in the collection match the specified pattern. You can use wildcards to select multiple nets at the same time.
`get_pins`	Pins in the design. All pin names in the collection match the specified pattern. You can use wildcards to select multiple pins at the same time.
`get_ports`	All ports (design inputs and outputs) in the design.
`get_registers`	Gets the specified registers in the design.
`get_keepers`	Gets the specified keepers in the design. Keepers are I/O ports or registers.

You can also examine collections and experiment with collections using wildcards in the Timing Analyzer by clicking **Name Finder** from the **View** menu.

2.6.3.1. Wildcard Characters

To apply constraints to many nodes in a design, use the "*" and "?" wildcard characters. The "*" wildcard character matches any string; the "?" wildcard character matches any single character.

If you apply a constraint to node `reg*`, the Timing Analyzer searches for and applies the constraint to all design nodes that match the prefix `reg` with any number of following characters, such as `reg`, `reg1`, `reg[2]`, `regbank`, and `reg12bank`.

If you apply a constraint to a node specified as `reg?`, the Timing Analyzer searches and applies the constraint to all design nodes that match the prefix `reg` and any single character following; for example, `reg1`, `rega`, and `reg4`.

2.6.3.2. Adding and Removing Collection Items

Wildcards that you use with collection commands define collection items that the command identifies. For example, if a design contains registers with the name `src0`, `src1`, `src2`, and `dst0`, the collection command `[get_registers src*]` identifies registers `src0`, `src1`, and `src2`, but not register `dst0`. To identify register `dst0`, you must use an additional command, `[get_registers dst*]`. To include `dst0`, you can also specify a collection command `[get_registers {src* dst*}]`.

To modify collections, use the `add_to_collection` and `remove_from_collection` commands. The `add_to_collection` command allows you to add additional items to an existing collection.

add_to_collection Command

add_to_collection *<first collection> <second collection>*

Note: The `add_to_collection` command creates a new collection that is the union of the two collections you specify.

The `remove_from_collection` command allows you to remove items from an existing collection.

remove_from_collection Command

remove_from_collection *<first collection> <second collection>*

The following example shows use of `add_to_collection` to add items to a collection.

Adding Items to a Collection

```
#Setting up initial collection of registers
set regs1 [get_registers a*]
#Setting up initial collection of keepers
set kprs1 [get_keepers b*]
#Creating a new set of registers of $regs1 and $kprs1
set regs_union [add_to_collection $kprs1 $regs1]
#OR
#Creating a new set of registers of $regs1 and b*
#Note that the new collection appends only registers with name b*
# not all keepers
set regs_union [add_to_collection $regs1 b*]
```

In the Quartus Prime software, keepers are I/O ports or registers. An `.sdc` file that includes `get_keepers` is incompatible with third-party timing analysis flows.

Related Information

- add_to_collection Command, Quartus Prime Help
- remove_from_collection Command, Quartus Prime Help

2.6.3.3. Query of Collections

You can display the contents of a collection with the `query_collection` command. Use the `-report_format` option to return the contents in a format of one element per line. The `-list_format` option returns the contents in a Tcl list.

```
query_collection -report_format -all $regs_union
```

Use the `get_collection_size` command to return the number of items the collection contains. If your collection is in a variable with the name `col`, use `set num_items [get_collection_size $col]` rather than `set num_items [llength [query_collection -list_format $col]]` for more efficiency.

2.6.3.4. Using the get_pins Command

The `get_pins` command supports options that control the matching behavior of the wildcard character (*). Depending on the combination of options you use, you can make the wildcard character (*) respect or ignore individual levels of hierarchy. The pipe character (|) indicates levels of hierarchy. By default, the wildcard character (*) matches only a single level of hierarchy.

These examples filter the following node and pin names to illustrate function:

- `lvl` (a hierarchy level with the name `lvl`)
- `lvl|dataa` (an input pin in the instance `lvl`)
- `lvl|datab` (an input pin in the instance `lvl`)
- `lvl|cnod` (a combinational node with the name `cnod` in the `lvl` instance)
- `lvl|cnod|datac` (an input pin to the combinational node with the name `cnod`)
- `lvl|cnod|datad` (an input pin to the combinational node `cnod`)

Table 41. Sample Search Strings and Search Results

Search String	Search Result
`get_pins *|dataa`	`lvl|dataa`
`get_pins *|datac`	*<empty>*[6]
`get_pins *|*|datac`	`lvl|cnod|datac`
`get_pins lvl*|*`	`lvl|dataa, lvl|datab`

continued...

(6) The search result is *<empty>* because the wildcard character (*) does not match more than one hierarchy level, that a pipe character (|) indicates, by default. This command matches any pin with the name `datac` in instances at the top level of the design.

Search String	Search Result
`get_pins -hierarchical *|*|datac`	*<empty>*[6]
`get_pins -hierarchical lvl|*`	`lvl|dataa, lvl|datab`
`get_pins -hierarchical *|datac`	`lvl|cnod|datac`
`get_pins -hierarchical lvl|*|datac`	*<empty>*[6]
`get_pins -compatibility_mode *|datac`	`lvl|cnod|datac` [7]
`get_pins -compatibility_mode *|*|datac`	`lvl|cnod|datac`

The default method separates hierarchy levels of instances from nodes and pins with the pipe character (|). A match occurs when the levels of hierarchy match, and the string values including wildcards match the instance or pin names. For example, the command `get_pins <instance_name>|*|datac` returns all the `datac` pins for registers in a given instance. However, the command `get_pins *|datac` returns an empty collection because the levels of hierarchy do not match.

Use the `-hierarchical` matching scheme to return a collection of cells or pins in all hierarchies of your design.

For example, the command `get_pins -hierarchical *|datac` returns all the `datac` pins for all registers in your design. However, the command `get_pins -hierarchical *|*|datac` returns an empty collection because more than one pipe character (|) is not supported.

The `-compatibility_mode` option returns collections matching wildcard strings through any number of hierarchy levels. For example, an asterisk can match a pipe character when using `-compatibility_mode`.

2.7. Using the Quartus Prime Timing Analyzer Document Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Added new *Using Timing Constraints throughout the Design Flow* topic. • Revised the *Step 1: Specify General Timing Analyzer Settings* topic to add link to SDC-on-RTL info. • Revised *Step 3: Run the Timing Analyzer* topic for Early Timing Analysis and SDC-on-RTL. • Revised *Applying Timing Constraints* section for Early Timing Analysis and SDC-on-RTL. • Revised *SDC File Precedence* topic for SDC-on-RTL. • Revised *Iteratively Modifying Constraints* topic for SDC-on-RTL. • Added new *Using Entity-based SDC-on-RTL Constraints* section. • Revised *Using Entity-bound SDC Files* section for SDC-on-RTL. • Renamed *Generate Timing Reports* topic to *Timing Report Descriptions* and relocated.

continued...

[7] When you use `-compatibility_mode`, the Timing Analyzer does not treat pipe characters (|) as special characters when you use the characters with wildcards.

Document Version	Quartus Prime Version	Changes
		• Revised the *Timing Report Descriptions* topic to reference quartus_staw method. • Revised *Report Timing By Source File* topic to mention ability to open the source file in a text editor from the report. • Revised *Report Metastability* topic to mention ability to run Report Neighbor Paths, default toggle rate, and more report details. • Revised *Report Route Net of Interest* topic to mention ability to run report_timing from the report. • Revised *Timing Analyzer Scripting* topic to reference quartus_sta and quartus_staw executables. • Added more details about derived_clock naming to *Automatic Clock Detection and Constraint Creation* topic. • Added new *The quartus_staw Executable* topic. • Enhanced *Using Fitter Overconstraints* topic for details specific to the Hyperflex architecture, new examples, and *Other Overconstraint Combinations* table of examples. • Added new *Report SDC* topic explaining report and Comments column. • Revised constraint precendence in *Maximum Skew* topic.
2023.12.04	23.4	• Revised the *Applying Timing Constraints* topic to add introductory information and context about typical constraints. • Added new *Check Timing* topic describing Check Timing command and report. • Added new cross-referencing links throughout *Applying Timing Constraints* section.
2023.10.02	23.3	• Updated compilation dashboard image in *Step 3: Run the Timing Analyzer*. • Revised the following topics entirely: — *Using Entity-bound SDC Files* — *Entity-bound Constraint Scope* — *Entity-bound Constraint Examples*
2023.08.03	23.1	• Corrected typo in command example in *Report Data Delay* topic. • Corrected typo in command example in *Constraining CDC Paths* topic. • Corrected typo in command example in *Maximum Skew* topic. • Added *Promoting Critical Warnings to Errors* topic.
2023.04.03	23.1	• Revised updated image and description in *Report Register Description* for new Without a Control Signal and Synchronous Load columns and data.
2023.01.31	22.4	• Revised outdated timing model descriptions in *Setting the Operating Conditions for Timing Analysis* topic.
2022.09.26	22.3	• Renamed the report title "Report Reset Statistics" as "Report Register Statistics." • Revised the *Report Register Statistics* topic to describe some new features of the report. • Revised *Report Fmax Summary* topic to refer to get_clock_fmax_info command and provide more detail. • Revised *Example SDC Constraints for External Clock Mux* to replace logically_exclusive with physically_exclusive.
2022.03.28	22.1	• Described new Clock Network Viewer in *Report Clocks and Clock Networks* topic. • Updated *Report Register Spread* topic for new angle and area spread types and -to_clock filtering. • Added new *Report Timing By Source Files* topic. • Added new *Report Metastability* topic. • Added new *Report Bottlenecks* topic.

continued...

Document Version	Quartus Prime Version	Changes
		• Added new *Specifying Custom Bottleneck Criteria* topic. • Revised Basic .sdc Constraints Example in *Example Circuit and SDC File* topic. • Added more detailed constraint example and diagram to *Set Clock Groups* topic. • Revised scripting examples and screenshots in *Correlating Constraints to the Timing Report* topic. • Revised scripting example in *Creating Base Clocks* topic. • Revised scripting example in *Clock Divider Example* topic. • Revised *Constraining CDC Paths* topic. • Added note about referenced SDCs within IP to *SDC File Precedence* topic.
2021.09.27	21.3	• Updated name of Report Hierarchical Retiming Restrictions command and report to Report Retiming Restrictions. • Added *Constraining CDC Paths* topic and linked to related topics. • Updated *Setting the Operating Conditions* with details about operating condition nomenclature. • Replaced Custom Reports, Device Specific, Diagnostic, and Slack report folder names throughout. • Added *Report Exceptions and Exceptions Reachability* topic describing new report. • Added *Report Clocks and Clock Networks* topic describing new report. • Added *Report Data Delay* topic describing report. • Mentioned option to view multiple before and after paths in *Report Neighbor Paths* topic. • Added `set_clock_groups` to *Timing Exception Precedence* • Updated content of Extra Info tab in *Report Timing* topic. • Corrected the *set_clock_groups -group A -group B* table in the *Creating Clock Groups* topic. • Removed *Report Custom CDC Viewer Command* topic. • Revised assignment examples in *Exclusive Clock Groups* topic.
2021.04.05	21.1	• Added: — "Report Reset Statistics" — "Report Asynchronous CDC" — Two new fields to "Report Pipelining Information". — New screenshots to "Report Logic Depth" and "Report Neighbor Paths" — `get_registers` and `get_keepers` to "Collection Commands". • Removed `-include` and `-exclude` options from "Maximum Skew"
2021.02.22	20.3	Added extra `SDC_ENTITY_FILE` info to "Using Entity-bound SDC Files"
2020.09.28	20.3	• Added "Cross Probing with Design Assistant" section. • Updated "Step 3: Run the Timing Analyzer" for multiple methods. • Updated "Step 1: Specify Timing Analyzer Settings for new tabbed dialog box and options. • Added new "Report Register Spread," "Report Route Net of Interest," "Report Hierarchical Retiming Restrictions," and "Report Pipelining Information" topics. • Updated "Report Clock Transfers" topic for new data columns. • Updated "Report Timing" topic for Extra Info tab data. • Updated "Report Fmax Summary," "Report Logic Depth," "Report Neighbor Paths," "Report CDC Viewer," and "Report Custom CDC Viewer" topics for latest GUI and consistency.

continued...

Document Version	Quartus Prime Version	Changes
2020.04.13	20.1	• Added details and Agilex 7 device examples to "Setting the Operating Conditions" topic. • Added "Report Logic Level Depth" topic. • Added "Report Neighbor Paths" topic. • Added "Enabling Time Borrowing Optimization" topic. • Added "Report Time Borrowing Data" topic.
2019.07.15	19.2	• Updated "Setting Operating Conditions" for SmartVID timing models. • Added step for setting operating conditions to "Step 4: Run Timing Analysis." • Added details about exclusive paths to "Maximum Skew" topic. • Added GUI steps for creating entity-bound SDC files to "Using Entity-bound SDC Files" topic.
2019.04.15	19.1	• Corrected typo in "Timing Constraint Precedence" topic. • Corrected typo in "Maximum Skew" topic. • Updated "Viewing Design Assistant Recommendations" for latest GUI changes.
2018.11.07	18.1	• Improved description and diagram for "Exclusive Clock Groups" topic.
2018.09.24	18.1	• Added "Using Entity-bound SDC Files" topic. • Added "Scoping Entity-bound Constraints" topic. • Added "Entity-bound Constraint Examples" topic. • Revised "Basic Timing Analysis Flow" section to add sequential step organization, update steps, and add supporting screenshots. • Added Timing Analyzer screenshot to "Using the Timing Analyzer" topic. • Removed "Creating a Constraint File from Templates with the Text Editor" topic due to limitations of this feature in this version of the software. • Retitled "SDC Constraint Creation Summary" to " Dual Clock SDC Example." • Retitled "Default Settings" to "Default Multicycle Analysis." • Retitled "SDC (Clock and Exception) Assignments on Blackbox Ports" to "Constraining Design Partition Ports." • Added "Viewing Design Assistant Recommendations" topic.
2018.05.07	18.0	• First release as part of the stand-alone *Timing Analyzer User Guide*
2017.11.27	17.1.0	• Removed outdated figure: Design Flow with the Timing Analyzer. • Updated Performing an Initial Analysis and Synthesis topic with Quartus Prime Pro Edition commands.
2017.11.06	17.1	• Updated *Using Fitter Overconstraints* topic for Stratix 10 support.
2017.05.08	17.0	• Added *Using Fitter Overconstraints* topic. • Added *Clock Domain Crossing report* topics
2016.10.31	16.1	• Implemented Intel rebranding. • Added support for `-blackbox` option with `set_input_delay`, `set_output_delay`, `remove_input_delay`, `remove_output_delay`.
2016.05.03	16.0	Added new topic: SCDS (Clock and Exception) Assignments on Blackbox Ports
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Added a description of running three- and four-corner analysis with `--mode=implement|finalize`. • Added description for new `set_operating_conditions` UI.

continued...

Document Version	Quartus Prime Version	Changes
2015.05.04	15.0.0	Added and updated contents in support of new timing algorithms for Arria 10: • Enhanced Timing Analysis for Arria 10 • Maximum Skew (`set_max_skew` command) • Net Delay (`set_net_delay` command) • Create Generated Clocks (clock-as-data example)
2014.12.15	14.1	Major reorganization. Revised and added content to the following topic areas: • Timing Constraints • Create Clocks and Clock Constraints • Creating Generated Clocks • Creating Clock Groups • Clock Uncertainty • Running the Timing Analyzer • Generating Timing Reports • Understanding Results • Constraining and Analyzing with Tcl Commands
August 2014	14.0a10.0	Added command line compilation requirements for Arria 10 devices.
June 2014	14.0	• Minor updates. • Updated format.
November 2013	13.1	• Removed HardCopy device information.
June 2012	12.0	• Reorganized chapter. • Added "Creating a Constraint File from Quartus Prime Templates with the Quartus Prime Text Editor" section on creating an SDC constraints file with the **Insert Template** dialog box. • Added "Identifying the Quartus Prime Software Executable from the SDC File" section. • Revised multicycle exceptions section.
November 2011	11.1	• Consolidated content from the Best Practices for the Quartus Prime Timing Analyzer chapter. • Changed to new document template.
May 2011	11.0	• Updated to improve flow. Minor editorial updates.
December 2010	10.1	• Changed to new document template. • Revised and reorganized entire chapter. • Linked to Quartus Prime Help.

continued...

Document Version	Quartus Prime Version	Changes
July 2010	10.0	Updated to link to content on SDC commands and the Timing Analyzer GUI in Quartus Prime Help.
November 2009	9.1	Updated for the Quartus Prime software version 9.1, including: • Added information about commands for adding and removing items from collections • Added information about the set_timing_derate and report_skew commands • Added information about worst-case timing reporting • Minor editorial updates
November 2008	8.1	Updated for the Quartus Prime software version 8.1, including: • Added the following sections: "set_net_delay" on page 7–42 "Annotated Delay" on page 7–49 "report_net_delay" on page 7–66 • Updated the descriptions of the `-append` and `-file` <*name*> options in tables throughout the chapter • Updated entire chapter using 8½" × 11" chapter template • Minor editorial updates

2.8. Quartus Prime Pro Edition User Guide: Timing Analyzer Archive

For the latest and previous versions of this user guide, refer to Quartus Prime Pro Edition User Guide: Timing Analyzer. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel® FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel® FPGA devices, and program CPLD and configuration devices, via connection with an Intel® FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel® FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Power Analysis and Optimization

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

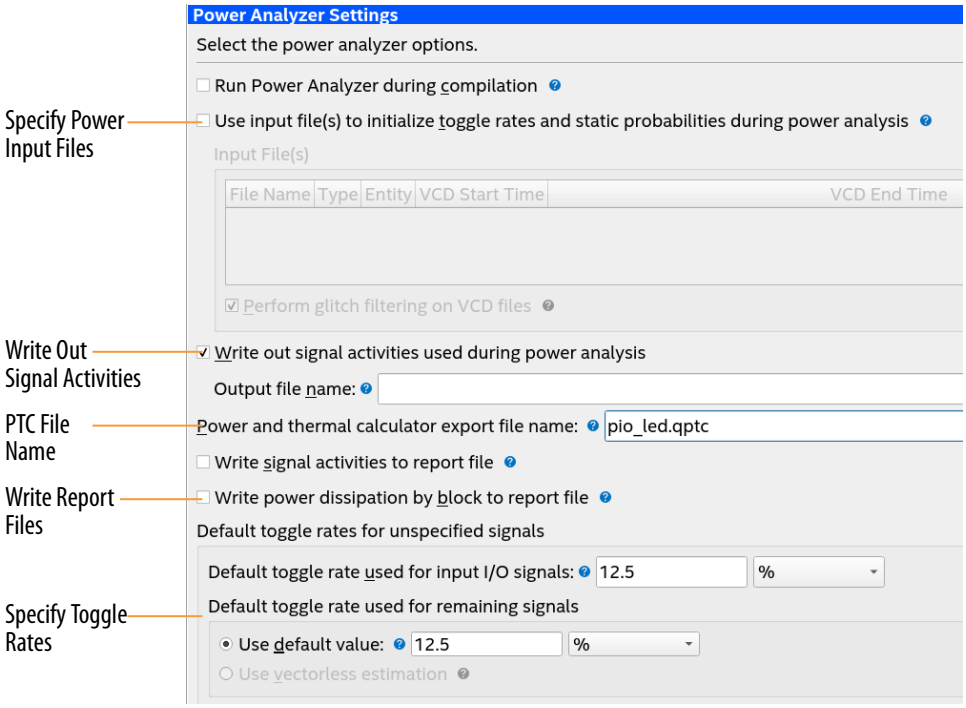
Contents

1. Power Analysis

Power consumption is a critical design consideration. When designing a PCB, you must determine the power consumption of the FPGA device to develop an accurate power budget, and to design the power supplies, voltage regulators, heat sink, and cooling system.

The Quartus® Prime software includes the Power Analyzer to help you to estimate the power consumption of your compiled design.

Figure 1. Power Analyzer Tool Settings



The Quartus Prime Design Suite also provides the Early Power Estimator (EPE) spreadsheet for Arria® 10 devices, and the Intel® FPGA Power and Thermal Calculator for Agilex™ FPGA portfolio and Stratix® 10 devices to estimate power consumption calculated from your predicted design characteristics.

Power estimation and analysis allows you to confirm that your design does not exceed thermal or power supply requirements throughout the design process:

- **Thermal**—Thermal power is the power that dissipates as heat from the FPGA. Devices use a heatsink or fan to act as a cooling solution. This cooling solution must be sufficient to dissipate the heat that the device generates. Additionally, the computed junction temperature must fall within normal device specifications.
- **Power supply**—Power supply is the power that the device needs to operate. Power supplies must provide adequate current to support device operation.

Note: Do not use the results of the Power Analyzer as design specifications. You must also verify the actual power during device operation to account for actual environmental operating conditions.

Related Information

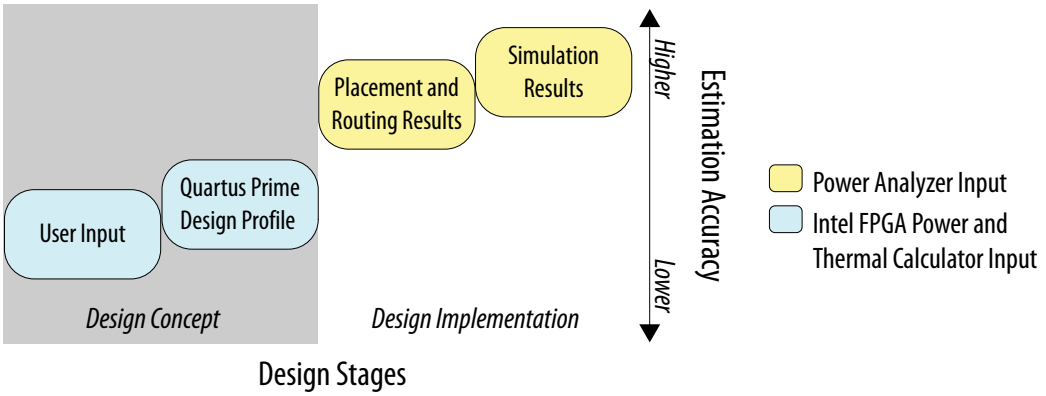
- Intel FPGA Power and Thermal Calculator (PTC) User Guide
- Power Analyzer Support Resources

1.1. Power Analysis Tools

The Quartus Prime Design Suite provides tools to analyze the power consumption of your FPGA design at different stages of the design process.

- Intel FPGA Power and Thermal Calculator (PTC)—estimates power supply and system thermal requirements before compiling the design, or anytime during the design phase. Supports Agilex FPGA portfolio and Stratix 10 devices.
- Quartus Prime Power Analyzer (QPA)—estimates power consumption for a post-fit design, allowing you establish guidelines for the power budget.
- Early Power Estimator (EPE) spreadsheet—estimates power consumption for power supply planning before compiling the design. Supports Arria 10 and Stratix 10 devices. (For versions of the Quartus Prime software later than version 19.4, Stratix 10 devices are supported in the Intel FPGA Power and Thermal Calculator.)

Figure 2. Estimation Accuracy for Different Inputs and Power Analysis Tools



The accuracy of the power model is determined on a per-power-rail basis for the Quartus Prime Power Analyzer.

- For most Stratix 10 designs, the Quartus Prime Power Analyzer has the following accuracy, assuming final power models: Within 10% of silicon for the majority of power rails with higher power, assuming accurate inputs and toggle rates.
- For most Agilex FPGA portfolio designs, the Quartus Prime Power Analyzer has the following accuracy, assuming final power models: Within 10% of silicon for all power rails, assuming accurate inputs and toggle rates.

Table 1. Comparison of EPE/Intel FPGA PTC and Quartus Prime Power Analyzer Capabilities

Characteristic	EPE / PTC	Quartus Prime Power Analyzer
When to use	Any time *Note:* For post-fit power analysis, you get better results with the Quartus Prime Power Analyzer.	Post-fit
Software requirements	EPE: Spreadsheet program. Intel FPGA PTC: Integrated into the Quartus Prime software, and is also available as a standalone tool.	The Quartus Prime software
Accuracy	Medium	Medium to very high
Data inputs	• Resource usage estimates • Clock requirements • Environmental conditions • Toggle rate	• Post-fit design • Clock requirements • Signal activity defaults • Environmental conditions • Register transfer level (RTL) simulation results (optional) • Post-fit simulation results (optional) • Signal activities per node or entity (optional)
Data outputs *Note:* The EPE and Power Analyzer outputs vary by device family.	• Total thermal power dissipation • Thermal static power • Thermal dynamic power • Off-chip power dissipation • Current drawn from voltage supplies	• Total thermal power • Thermal static power • Thermal dynamic power • Thermal I/O power • Thermal power by design hierarchy • Thermal power by block type • Thermal power dissipation by clock domain • Device supply currents
Estimation of transceiver power for dynamic reconfiguration features	Includes an estimation of the incremental power consumption by these features.	Not included

Note: The Quartus Prime Power Analyzer does not support power analysis of the following Intel FPGA IP:

- Stratix 10 HBM2 IP
- Stratix 10 HPS IP
- Arria 10 HPS IP

In versions of the Quartus Prime software later than 19.4, you can obtain power estimations for the Stratix 10 HBM2 IP and Stratix 10 HPS IP using the Intel FPGA Power and Thermal Calculator (PTC).

For power estimation of Arria 10 HPS IP, and for power estimation in the Quartus Prime software version 19.4 or earlier, you can obtain power estimations using the Early Power Estimator spreadsheet (EPE).

1.2. Running the Power Analyzer

Before running the Power Analyzer you must run full compilation of your design to generate the post-fit netlist. In addition, you must either provide timing assignments for all clocks in the design, or specify signal activity data for power analysis. You must specify the I/O standard on each device input and output, and the board trace model on each output in the design.

To run the Power Analyzer:

1. To specify device power characteristics, operating voltage, and temperature conditions for power analysis, click **Assignments ➤ Settings ➤ Operating Settings and Conditions**, as Settings for Power Analysis on page 8 describes.
2. To run full compilation of your design, click **Processing ➤ Start Compilation**.
3. Click **Assignments ➤ Settings ➤ Power Analyzer**.
4. Specify the source of signal activity data, as Generating Signal Activity Data for Power Analysis on page 12 describes.
5. To generate a Signal Activity (`.saf`) file during analysis, turn on **Write out signal activities used during power analysis**, and specify the file name.
6. You can customize the generated **Power and Thermal Calculator export file name**. This file summarizes the resource utilization and allows you to perform what-if analyses in PTC
7. Specify the **Default toggle rates for unspecified signals**, as Specifying the Default Toggle Rate on page 17 describes.
8. To specify temperature range and cooling options, click **Cooling Solution and Temperature**.
9. To run full compilation of your design, click **Processing ➤ Start ➤ Start Power Analyzer**.
10. When power analysis is complete, click **Report** to open the Power Analyzer reports that Viewing Power Analysis Reports on page 19 describes.

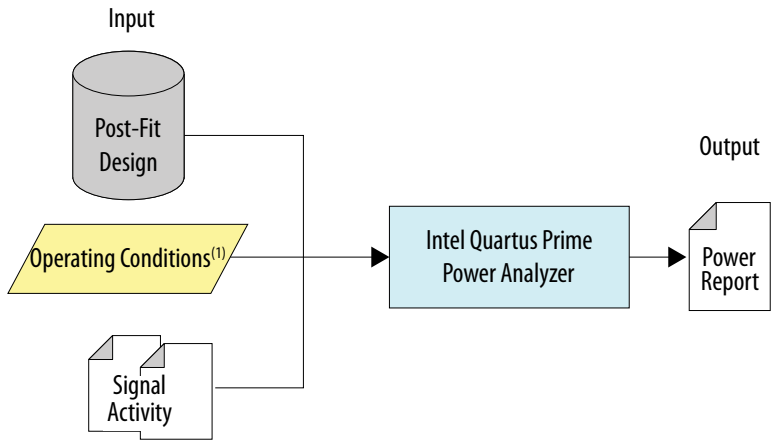
Related Information

Specifying Power Analyzer Input on page 8

1.3. Specifying Power Analyzer Input

The Power Analyzer accuracy is driven by design factors, operating conditions, and signal activity data that affect power consumption. The following figure shows how the Power Analyzer interprets these inputs and generates results in the Power Analysis report:

Figure 3. Power Analyzer High-Level Flow



(1) Operating condition specifications are available for only some device families

To obtain accurate I/O power estimates, the Power Analyzer requires full compilation of your design, in addition to specifying the following settings:

- The electrical standard on each I/O cell.
- The board trace model on each I/O standard in the design.
- Timing assignments for all the clocks in your design, or use a simulation-based flow to generate activity data.

Note: For accurate results, ensure that any .VCD file used with the Power Analyzer is the result of gate-level simulation.

1.3.1. Settings for Power Analysis

You can specify device power characteristics, operating voltage conditions, operating temperature conditions, Power Analyzer settings and thermal settings, in the **Operating Settings and Conditions**, **Power Analyzer Settings**, and **Thermal** pages of the **Settings** dialog box.

Table 2. Operating Settings and Conditions

Option	Settings
Device power characteristics	• **Maximum**—specifies maximum power consumed by the worst-case device. This is the default value for Stratix 10 and Agilex devices. • **Typical**—specifies average power consumed by typical silicon at nominal operating conditions.
Voltage tab	Specifies the operating voltage conditions for each power rail in the device, and the supply voltages for power rails with selectable supply voltages.
Temperature tab	Specifies the minimum and maximum junction temperature range.

Figure 4. Operating Settings and Conditions

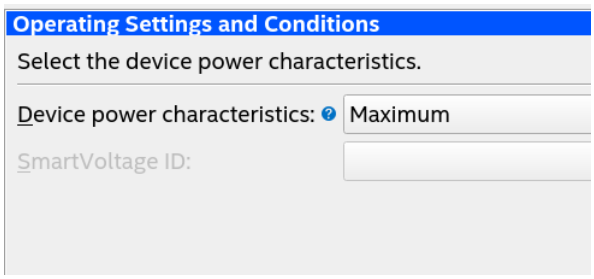


Figure 5. Power Analyzer Settings

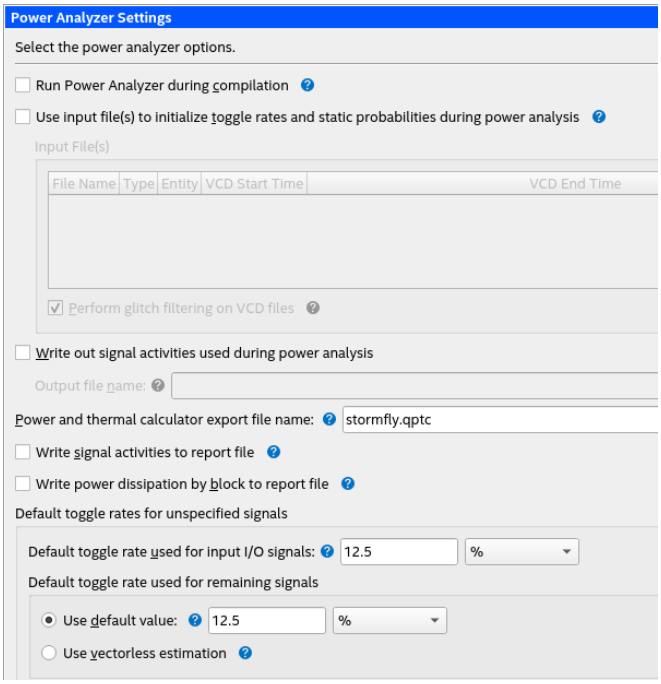


Table 3. Power Analyzer Settings

Option	Settings
Power Analyzer Settings	Specifies the Power Analyzer options, including: • **Run Power Analyzer during compilation**—Check this box to turn on power analysis during compilation. • **Use input file(s) to initialize toggle rates and static probabilities during power analysis**—Check this box to use Signal Activity Files or VCD files to initialize toggle rates and static probabilities for power estimation. • **Write out signal activities used during power analysis**—Check this box to write the toggle rates and static probabilities used during power estimation to a file. • **Power and thermal calculator export file name**—specifies the export file name (`.qptc`) of the design summary that you can import into the Intel FPGA Power and Thermal Calculator. You can customize this name. • **Write signal activities to report file**—Check this box to have the Power Analyzer write a report file containing the signal activities used during power analysis.

Option	Settings
	• **Write power dissipation by block to report file**—Check this box to have the Power Analyzer report the thermal power dissipation calculated during power analysis, in the **Thermal Power Dissipation by Block** report panel. • **Default toggle rate used for input I/O signals**—Specify a default toggle rate for use on input I/O pins during power estimation. Can be expressed as a percentage or in transitions/second. • **Default toggle rate used for remaining signals** — **Use default value**—Specify a default toggle rate for use during power estimation on all nodes except I/O pins. This value is used only if no toggle rate is specified through a Signal Activity File, VCD file, or user assignment. Can be expressed as a percentage or in transitions/second. — **Use vectorless estimation**—Turn on this control to use vectorless estimation to fill in undefined toggle rates and static probabilities. If this option is not available, the device family does not support vectorless estimation.

Figure 6. Thermal Settings

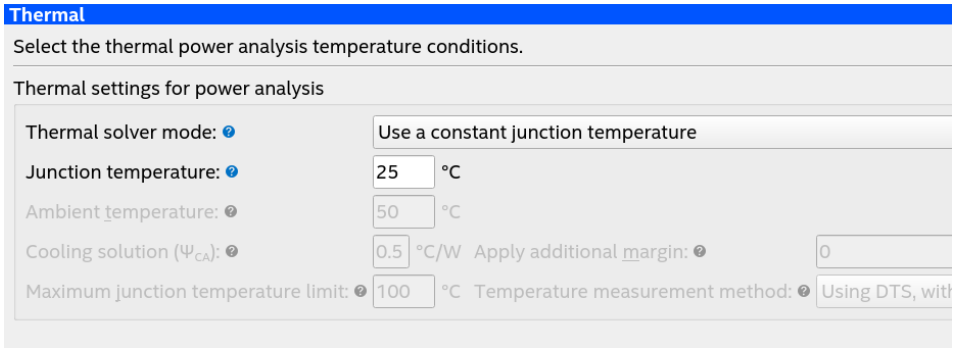


Table 4. Thermal Settings

Option	Settings
Thermal Settings	Specifies the thermal power analysis temperature conditions, including: • **Thermal Solver Mode**—Select the thermal solver mode to use during power estimation. • **Junction temperature**—Specifies the junction temperature, in °C, used during power estimation. • **Ambient temperature**—Specifies the ambient temperature, in °C, used during power estimation. • **Cooling solution**—Specifies the cooling solution case-to-ambient thermal resistance, in °C per watt. • **Maximum junction temperature limit**—Specifies the maximum junction temperature limit that no part of any die in the package should exceed. • **Apply additional margin**—Specifies, as a percentage, the amount of additional margin to apply to detailed thermal analysis results. Valid values are 0–25%. The default value is 0%. The recommended margin for Agilex FPGA portfolio devices is 10%, and for Stratix 10 devices, 25%. *Note:* For a design compiled in an earlier version of the Quartus Prime software with the **Apply Recommended Margin** parameter set to *Yes*, the current version of Power Analyzer interprets this as an **Apply Additional Margin** setting of 25%. • **Temperature measurement method**—Select the method to use for reporting temperature sensors for thermal analysis.

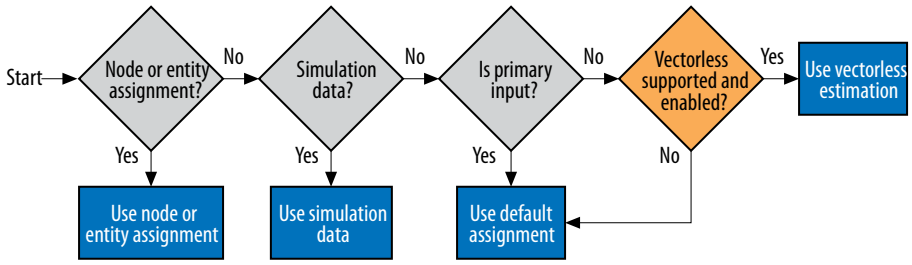
1.3.2. Specifying Signal Activity Data

The accuracy of the power estimation depends on how representative signal activity data is during power analysis. The Power Analyzer allows you to specify signal activity data from the following sources:

- `.vcd` files from supported simulators
- User-entered node, entity, clock, and toggle rate assignments
- Vectorless estimation (selected devices)

You can mix and match the signal activity data sources on a signal-by-signal basis.

Figure 7. Priority Scheme Applied to Each Signal

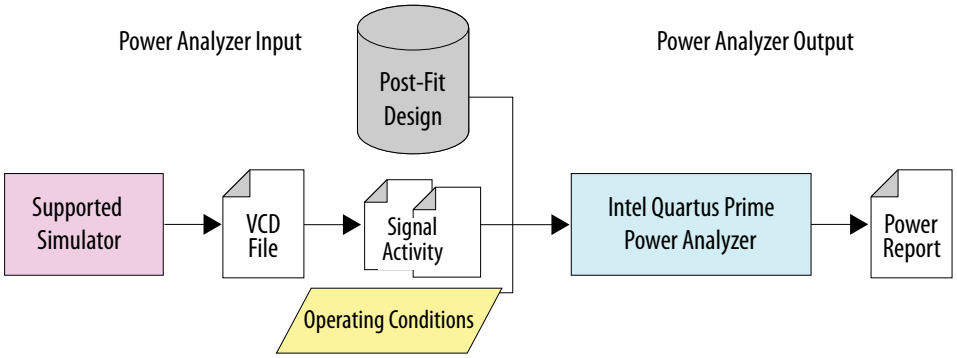


1.3.2.1. Using Simulation Signal Activity Data in Power Analysis

You can specify a Verilog Value Change Dump File (`.vcd`) generated by simulating a placed and routed gate-level netlist in a supported simulator as the source of signal activity data for power analysis. Third-party simulators can output a `.vcd` that contains signal activity and static probability information for power analysis. The `.vcd` includes all routing resources and the logic array resource usage.

To improve the accuracy of power analysis, you can generate a Standard Delay Output (`.sdo`) file that includes back-annotated delay estimates of the instances of core atoms for ModelSim* simulation. ModelSim simulation can then output a more accurate `.vcd` for use as power analysis input. You must run **Fitter (Finalize)** before generating the `.sdo`.

Figure 8. Using Simulation Signal Activity Data in Power Analysis

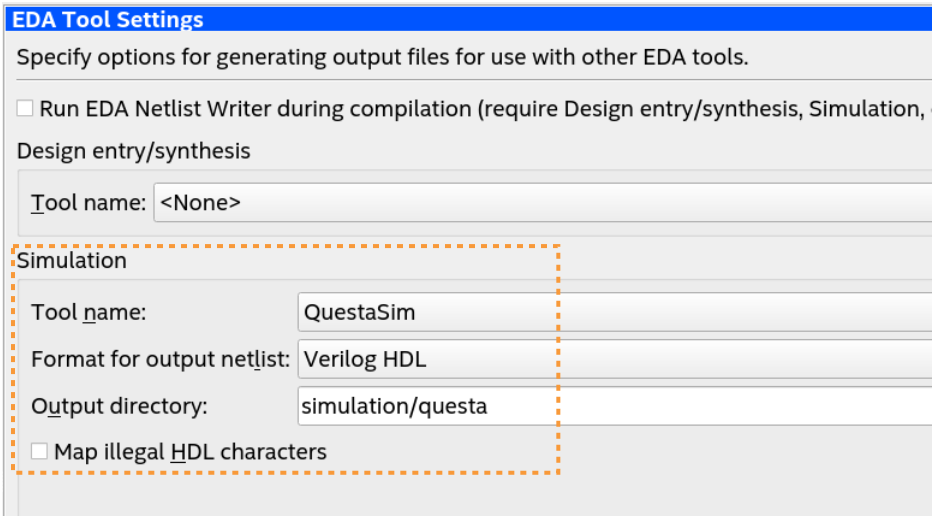


1.3.2.1.1. Generating Signal Activity Data for Power Analysis

To generate and use simulation signal activity data for power analysis:

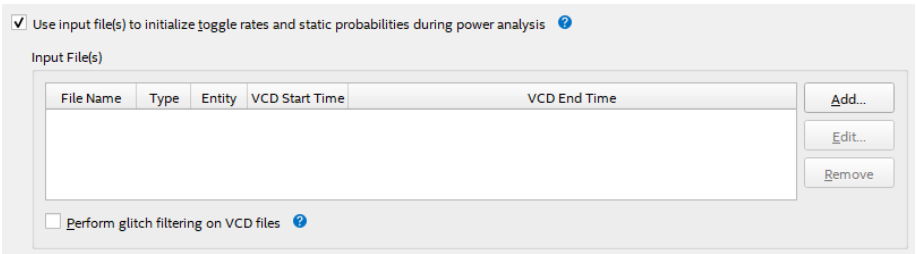
1. To run full compilation on your design, click **Processing ➤ Start Compilation**.
2. To specify settings for simulation output, click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ Simulation**. Select your simulator in **Tool name** and the **Format for output netlist** and **Output directory**.

Figure 9. EDA Tool Settings for Simulation



3. Turn on **Map illegal HDL characters**. This setting directs the EDA Netlist Writer to map illegal characters for VHDL or Verilog HDL, and results in more accurate data for power analysis.
4. Click the **Power Analyzer Settings** page.
5. For Stratix 10 designs, to generate a Standard Delay Output (`.sdo`) file that includes back-annotation of delays for power analysis, refer to Generating Standard Delay Output for Power Analysis on page 13.
6. Under **Input file**, turn on **Use input files to initialize toggle rates and static probabilities during power analysis**.

Figure 10. Specifying Power Analysis Input Files



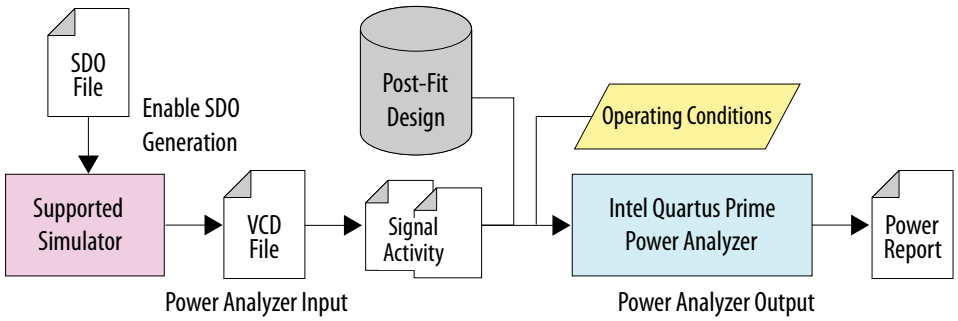
7. To specify a `.vcd` for power analysis, click **Add** and specify the **File name**, **Entity**, and **Simulation period** for the `.vcd`, and click **OK**.
8. To enable glitch filtering during power analysis with the `.vcd` you generate, turn on **Perform glitch filtering on VCD files**.
9. To run the power analysis, click **Start** on the **Power Analysis** step in the Compilation Dashboard. View the toggle rates in the power analysis results.

Note: To improve accuracy of power analysis, the Quartus Prime EDA Netlist writer can generate a Standard Delay Output (`.sdo`) file that includes back-annotation of delays for a design's netlist for use during simulation in QuestaSim. Although the `.sdo` only contains delay estimates and imprecise timing information, including the `.sdo` in simulation results in a more accurate output `.vcd` for power analysis. The EDA Netlist Writer currently supports `.sdo` file generation only for Verilog `.vo` simulation in the QuestaSim simulator (not ModelSim - Intel FPGA Edition) for Stratix 10 designs. The EDA Netlist Writer does not currently support `.sdo` file generation for any other simulator or device family.

1.3.2.1.2. Generating Standard Delay Output for Power Analysis

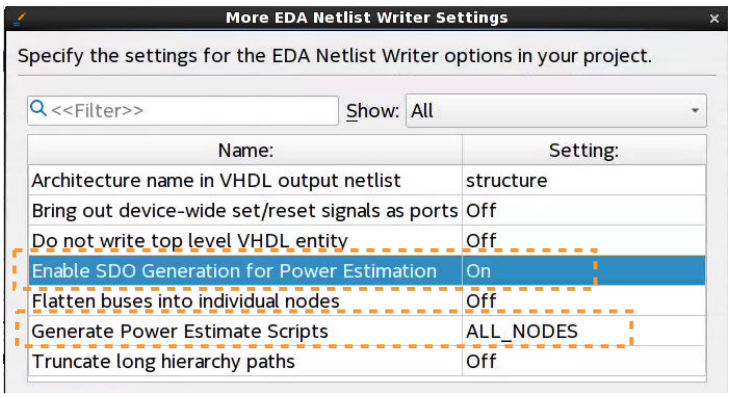
To improve accuracy of power analysis, you can generate a Standard Delay Output (`.sdo`) file that includes back-annotated delay estimates for QuestaSim simulation. QuestaSim simulation can then output a more accurate `.vcd` for use as power analysis input. You must run **Fitter (Finalize)** before generating the `.sdo`.

Figure 11. Using an SDO in Power Analysis



1. Click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ Simulation**. In **Tool name** select **QuestaSim** and **Verilog** for **Format for output netlist**.

Figure 12. More EDA Netlist Writer Settings



2. Click **More EDA Netlist Writer Settings**. Set **Enable SDO Generation for Power Estimation** to **On**. Set **Generate Power Estimate Scripts** to **ALL_NODES**.
3. To run the Fitter, click **Processing ➤ Start ➤ Start Fitter (Finalize)**.
4. Create a representative testbench (`.vt`) that exercises the design functions appropriately.
5. To specify the appropriate hierarchy level for signals in the output `.vcd`, add the following line to the project `.qsf` file: (1)

```
set_global_assignment -name EDA_TEST_BENCH_DESIGN_INSTANCE_NAME
     <DUT instance path> -section_id eda_simulation
```

6. After Fitter processing is complete, click **Processing ➤ Start ➤ Start EDA Netlist Writer**. EDA Netlist Writer generates the following files in `/<project>/simulation/questa/power/`:
   - `<project>.vo` (contains a reference to the `.sdo` file by default)
   - `<project>_dump_all_vcd_nodes.tcl`—specifies nodes to save in `.vcd`
   - `<project>_v.sdo`—back-annotated delay estimates
7. Create a QuestaSim script (`.do`) to load the design and testbench, start QuestaSim, and then source the `.do` script.
8. To specify the signals QuestaSim includes in the `.vcd` file, source `*_dump_all_vcd_nodes.tcl` in QuestaSim.
9. To generate the `.vcd` file, simulate the test bench and netlist in QuestaSim. The `.vcd` file generates according to your specifications.
10. Specify the `.vcd` as an input to power analysis, as Generating Signal Activity Data for Power Analysis describes.

(1) Specify the full hierarchical path in the testbench, not just the instance name. For example, specify a|b|c, not just c.

Note: The EDA Netlist Writer currently supports `.sdo` file generation only for Verilog `.vo` simulation in the QuestaSim simulator (not ModelSim - Intel FPGA Edition) for Stratix 10 designs. The EDA Netlist Writer does not currently support `.sdo` file generation for any other simulator or device family.

1.3.2.1.3. Simulation Glitch Filtering

The Power Analyzer defines a glitch as two signal transitions so closely spaced in time that the pulse, or glitch, occurs faster than the logic and routing circuitry can respond. The output of a transport delay model simulator contains glitches for some signals. The logic and routing structures of the device form a low-pass filter that filters out glitches that are tens to hundreds of picoseconds long, depending on the device family.

Some third-party simulators use different models than the transport delay model as the default model. Different models cause differences in signal activity and power estimation. The inertial delay model, which is the ModelSim default model, filters out more glitches than the transport delay model and usually yields a lower power estimate.

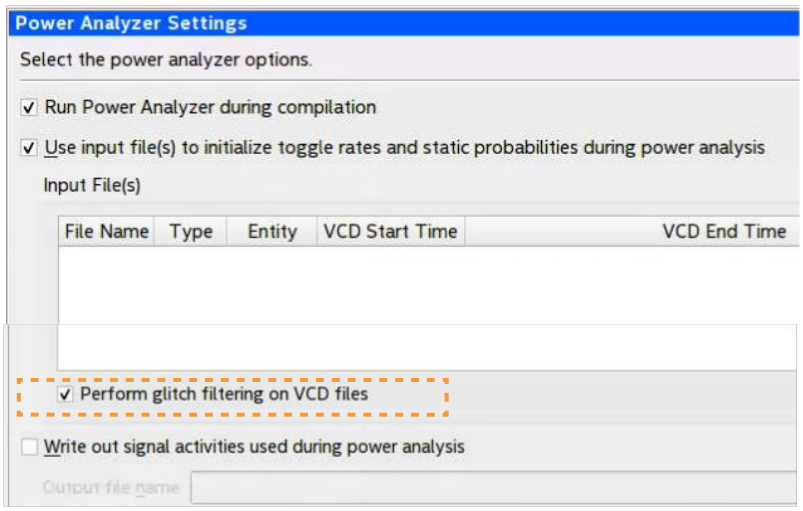
Note: Intel FPGA recommends that you use the transport simulation model when using the Quartus Prime software glitch filtering support with third-party simulators. Simulation glitch filtering has little effect if you use the inertial simulation model.

Glitch filtering in a simulator can also filter a glitch on one logic element (LE) (or other circuit element) output from propagating to downstream circuit elements to ensure that the glitch does not affect simulated results. Glitch filtering prevents a glitch on one signal from producing non-physical glitches on all downstream logic, which can result in a signal toggle rate and a power estimate that are too high. Circuit elements in which every input transition produces an output transition, including multipliers and logic cells configured to implement XOR functions, are especially prone to glitches. Therefore, circuits with such functions can have power estimates that are too high when glitch filtering is not used.

Note: Intel FPGA recommends that you use the glitch filtering feature to obtain the most accurate power estimates. For `.vcd` files, the Power Analyzer flows support two levels of glitch filtering.

Enable glitch filtering in the `.vcd` that you generate for use in power analysis by turning on **Perform glitch filtering on VCD files**.

Figure 13. Enabling Glitch Filtering for VCD



The `.vcd` file reader performs glitch filtering that is complementary to simulation glitch filtering, but is often less precise. While the `.vcd` file reader has the ability to remove glitches on logic blocks, the file reader cannot determine how a given glitch potentially affects downstream logic and routing. Filtering the glitches during simulation avoids switching downstream routing and logic automatically.

Note: When running simulation for design verification (rather than to produce input to the Power Analyzer), Intel recommends that you turn off the glitch filtering option to produce the most rigorous and conservative simulation from a functionality viewpoint. When performing simulation to produce input for the Power Analyzer, Intel FPGA recommends that you turn on the glitch filtering to produce the most accurate power estimates.

1.3.2.2. Signal Activities from RTL (Functional) Simulation, Supplemented by Vectorless Estimation

In the functional simulation flow, simulation provides toggle rates and static probabilities for all pins and registers in your design. Vectorless estimation fills in the values for all the combinational nodes between pins and registers, giving good results. This flow usually provides a compilation time benefit when you use the third-party RTL simulator.

1.3.2.2.1. RTL Simulation Limitation

RTL simulation may not provide signal activities for all registers in the post-fitting netlist because synthesis loses some register names. For example, synthesis might automatically transform state machines and counters, thus changing the names of registers in those structures. As a result, a large number of nodes in the `.vcd` file may not match the nodes in your design netlist, which can result in the power analysis results being less accurate or of lower confidence.

1.3.2.3. Signal Activities from Vectorless Estimation and User-Supplied Input Pin Activities

The vectorless estimation flow provides a low level of accuracy, because vectorless estimation for registers is not entirely accurate.

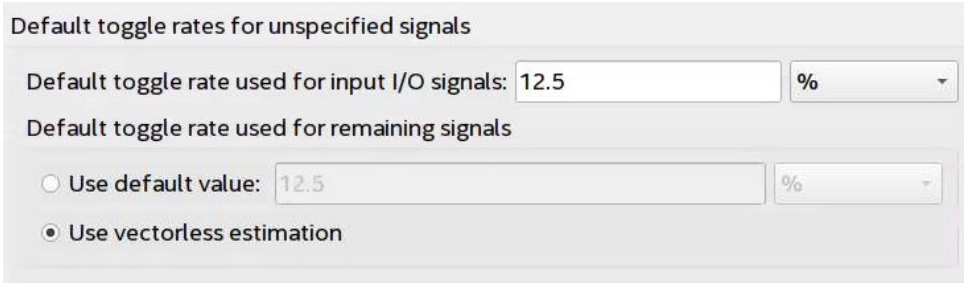
1.3.2.4. Signal Activities from User Defaults Only

The user defaults only flow provides the lowest degree of accuracy.

1.3.3. Specifying the Default Toggle Rate

You can specify the **Default toggle rates for unspecified signals** in your design for power analysis. The Power Analyzer uses the default toggle rate when no other method specifies the signal activity data.

Figure 14. Specifying the Default Toggle Rate



You specify the toggle rate in absolute terms (transitions per second), or as a fraction of the clock rate in effect for each node. The toggle rate for a clock derives from the timing settings for the clock. For example, if the Power Analyzer specifies a clock with an f_{MAX} constraint of 100 MHz and a default relative toggle rate of 20%, nodes in this clock domain transition in 20% of the clock periods, or 20 million transitions occur per second.

In some cases, the Power Analyzer cannot determine the clock domain for a node because the clock domain is ambiguous. For example, the Power Analyzer cannot determine a clock domain for a node unless you specify sufficient timing constraints for the clock domains. If the Power Analyzer cannot determine the clock domain for a node, the Power Analyzer substitutes and reports a toggle rate of zero.

Note: The transceiver I/O toggle rate is determined by the XCVR data rate value specified in your IP catalog settings. Do not include transceiver I/O toggle rate in the default toggle rates that you specify in the Power Analyzer.

Related Information

Toggle Rate on page 34

1.3.4. Specifying Toggle Rates for Specific Nodes

You can assign toggle rates and static probabilities to individual nodes in the design. These assignments have the highest priority, overriding data from all other signal activity sources.

You must use the Assignment Editor or Tcl commands to create the **Power Toggle Rate** and **Power Static Probability** assignments. You can specify the power toggle rate as an absolute toggle rate in transitions per second using the **Power Toggle Rate** assignment, or you can use the **Power Toggle Rate Percentage** assignment to specify a toggle rate relative to the clock domain of the assigned node for a more specific assignment made in terms of hierarchy level.

Note: If you use the **Power Toggle Rate Percentage** assignment, and the node does not have a clock domain, the Quartus Prime software issues a warning and ignores the assignment.

Assigning toggle rates and static probabilities to individual nodes is appropriate for signals in which you have knowledge of the signal being analyzed. For example, if you know that a 100 MHz data bus or memory output produces data that is essentially random (uncorrelated in time), you can directly enter a 0.5 static probability and a toggle rate of 50 million transitions per second.

The Power Analyzer treats bidirectional I/O pins differently. The combinational input port and the output pad for a pin share the same name. However, those ports might not share the same signal activities. For reading signal activity assignments, the Power Analyzer creates a distinct name *<node_name~output>* when configuring the bidirectional signal as an output and *<node_name~result>* when configuring the signal as an input. For example, if a design has a bidirectional pin named `MYPIN`, assignments for the combinational input use the name `MYPIN~result`, and the assignments for the output pad use the name `MYPIN~output`.

Note: When you create the logic assignment in the Assignment Editor, you cannot find the `MYPIN~result` and `MYPIN~output` node names in the Node Finder. Therefore, to create the logic assignment, you must manually enter the two differentiating node names to create the assignment for the input and output port of the bidirectional pin.

1.3.4.1. Clock Node Toggle Rates

For clock nodes, the Power Analyzer uses timing requirements to derive the toggle rate when neither simulation data nor user-entered signal activity data is available. f_{MAX} requirements specify full cycles per second, but each cycle represents a rising transition and a falling transition. For example, a clock f_{MAX} requirement of 100 MHz corresponds to 200 million transitions per second for the clock node.

1.3.5. Avoiding Simulation Node Name Match

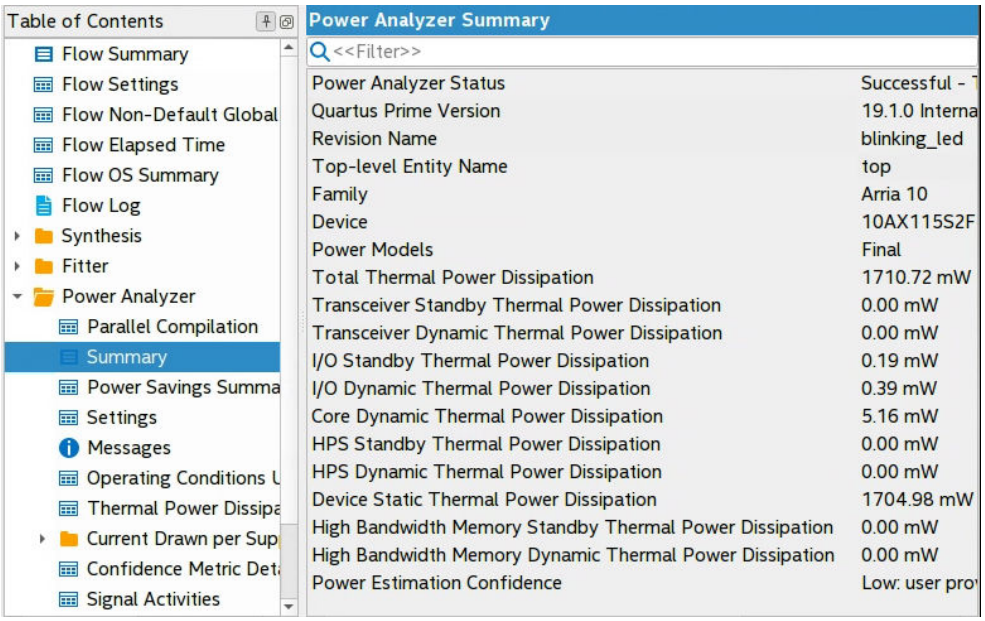
Node name mismatches happen when you have `.vcd` applied to entities other than the top-level entity. In a modular design flow, the gate-level simulation files created in different Quartus Prime projects might not match their node names with the current Quartus Prime project.

For example, you may have a file named `8b10b_enc.vcd`, which the Quartus Prime software generates in a separate project called `8b10b_enc` while simulating the `8b10b` encoder. If you import the `.vcd` into another project called `Top`, you might encounter name mismatches when applying the `.vcd` to the `8b10b_enc` module in the `Top` project. This mismatch happens because the Quartus Prime software might name all the combinational nodes in the `8b10b_enc.vcd` differently than in the `Top` project. To avoid such mismatches, Intel recommends using `.vcd` files generated from simulation of your top level project.

1.4. Viewing Power Analysis Reports

Following successful power analysis, click the **Power Analyzer** pulldown in the **Table of Contents** of the Compilation Report, to view the Power Analysis section of the report.

Figure 15. Power Analysis Reports



The Power Analysis reports contains the following sections:

Summary

The Summary section of the report shows the estimated total thermal power consumption of your design. This includes dynamic, static, and I/O thermal power consumption. The I/O thermal power includes the total I/O power drawn from the V_{CCIO} and V_{CCPD} power supplies and the power drawn from V_{CCINT} in the I/O subsystem including I/O buffers and I/O registers. The report also includes a confidence metric that reflects the overall quality of the data sources for the signal activities. For example, a **Low** power estimation confidence value reflects that you have provided insufficient toggle rate data, or most of the signal activity information used for power estimation is from default or vectorless estimation settings. For more information about the input data, refer to the Power Analyzer Confidence Metric report.

Power Savings Summary

Lists any savings (in mW) and the type of savings method, such as SmartVID Power Savings.

Parallel Compilation

When you enable parallel compilation, the Parallel Compilation report list the number of processors you use during Power Analysis

Settings

The Settings section of the report shows the Power Analyzer settings information of your design, including the default input toggle rates, operating conditions, and other relevant setting information.

Simulation Files Read

The Simulation Files Read section of the report lists the simulation output file that the `.vcd` used for power estimation. This section also includes the file ID, file type, entity, VCD start time, VCD end time, the unknown percentage, and the toggle percentage. The unknown percentage indicates the portion of the design module unused by the simulation vectors.

Operating Conditions Used

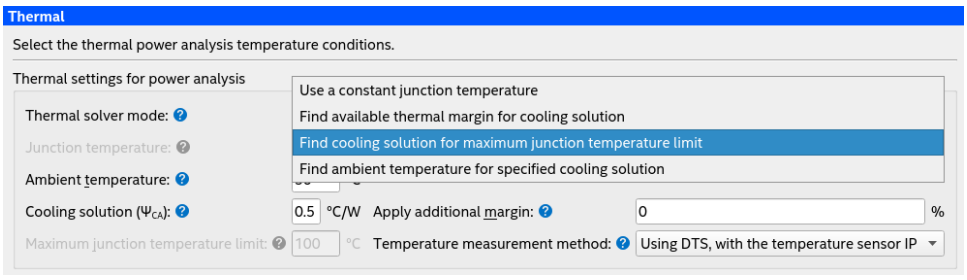
The Operating Conditions Used section of the report shows device characteristics, voltages, temperature, and cooling solution, if any, during the power estimation. This section also shows the entered junction temperature or auto-computed junction temperature during the power analysis.

Thermal Map Visualization

For Agilex FPGA portfolio designs, the Power Analyzer provides a visualization of the expected thermal distribution on the core die and the transceiver dies. This data is available when you run the Power Analyzer on your compiled Agilex FPGA portfolio design and **you have enabled the Thermal Map visualization by selecting one of the following options under the Thermal solver mode selection in the Thermal Settings dialog:**

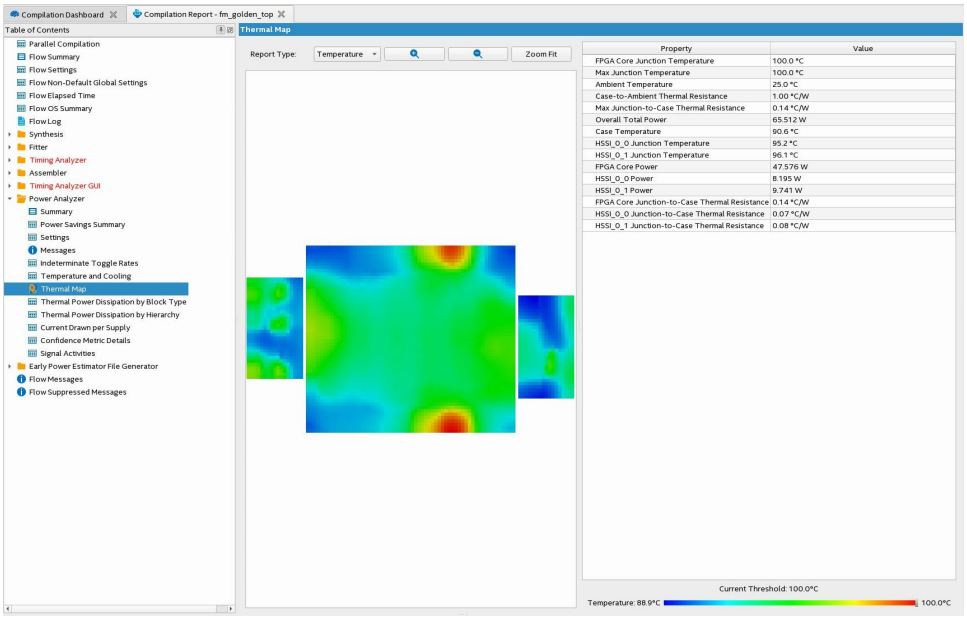
- **Find available thermal margin for cooling solution**
- **Find cooling solution for maximum junction temperature limit**
- **Find ambient temperature for specified cooling solution**

Figure 16. Thermal Settings to Enable Thermal Map Visualization



After you run the Power Analyzer, select the **Thermal Map** section in the Power Analyzer report. You can set the threshold temperature you want to use, which is useful if you are making any what-if analyses based on your thermal design. You can adjust the threshold temperature in increments of 5°C, between the ambient temperature (or 50°C, whichever is lower), and an upper limit of 100°C.

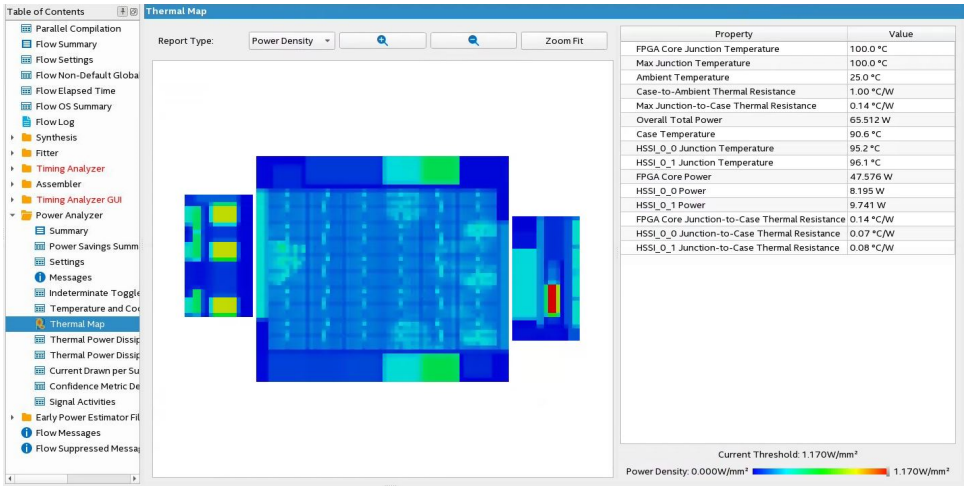
Figure 17. Temperature View of the Thermal Map for an Agilex 7 FPGA Design



Knowing the locations of hot spots in your design can help you make modifications as necessary for proper operation of the system.

The thermal map report can show two different views—a temperature view and a power density view. You can choose the view from a pulldown selection in the GUI, when you open the thermal map in the Power Analyzer.

Figure 18. Power Density View of the Thermal Map for an Agilex 7 FPGA Design



Thermal Power Dissipated by Block

The Thermal Power Dissipated by Block section of the report shows estimated thermal dynamic power and thermal static power consumption categorized by atoms. This information provides you with estimated power consumption for each atom in your design.

By default, this section does not contain any data, but you can turn on the report with the **Write power dissipation by block to report file** option on the **Power Analyzer Settings** page.

On-Chip Power Dissipation by Block Type

The On-Chip Power Dissipation by Block Type section of the report shows the estimated total on-chip power consumption by block type, and estimated on-chip dynamic power and estimated on-chip static power, by block type.

Figure 19. On-Chip Power Dissipation by Block Type

On-Chip Power Dissipation by Block Type

Show: All | Hide | <<Filter>>

	Block Type	Total On-Chip Power by Block Type (W)	Block On-Chip Dynamic Power (W)	Block On-Chip Static Power (W)
1	Clock	0.077	0.005	0.072
2	Crypto	0.035	0.000	0.035
3	DSP	0.026	0.004	0.022
4	IO	0.887	0.754	0.132
5	Logic	1.257	5.51E-04	1.257
6	Miscellaneous	2.520	0.709	1.811
7	PLL	0.010	0.000	0.010
8	RAM	0.521	0.092	0.429
9	Transceiver	1.963	0.397	1.566

On-Chip Power Dissipation by Hierarchy

This On-Chip Power Dissipation by Hierarchy section of the report shows estimated cumulative and current-hierarchy-level on-chip dynamic power consumption by hierarchy node. This information is useful when locating modules with high power consumption in your design. (Available for Agilex FPGA portfolio devices.)

Core Dynamic Thermal Power Dissipation by Clock Domain

The Core Dynamic Thermal Power Dissipation by Clock Domain section of the report shows the estimated total core dynamic power dissipation by each clock domain, which provides designs with estimated power consumption for each clock domain in the design. If the clock frequency for a domain is unspecified by a constraint, the clock frequency is listed as "unspecified." For all the combinational logic, the clock domain is listed as no clock with zero MHz.

Current Drawn per Supply

The Current Drawn per Supply section of the report lists the current drawn from each voltage supply. The V_{CCIO} and V_{CCPD} voltage supplies are further categorized by I/O bank and by voltage. This section also lists the minimum safe power supply size (current supply ability) for each supply voltage. Minimum current requirement can be higher than user mode current requirement in cases in which the supply has a specific power up current requirement that goes beyond user mode requirement.

The I/O thermal power dissipation on the summary page does not correlate directly to the power drawn from the V_{CCIO} and V_{CCPD} voltage supplies listed in this report. This is because the I/O thermal power dissipation value also includes portions of the V_{CCINT} power, such as the I/O element (IOE) registers, which are modeled as I/O power, but do not draw from the V_{CCIO} and V_{CCPD} supplies.

The reported current drawn from the I/O Voltage Supplies (ICCIO and ICCPD) as reported in the Power Analyzer report includes any current drawn through the I/O into off-chip termination resistors. This can result in ICCIO and ICCPD values that are higher than the reported I/O thermal power, because this off-chip current dissipates as heat elsewhere and does not factor in the calculation of device temperature. Therefore, total I/O thermal power does not equal the sum of current drawn from each V_{CCIO} and V_{CCPD} supply multiplied by V_{CCIO} and V_{CCPD} voltage.

For SoC devices, there is no standalone ICC_AUX_SHARED current drawn information. The ICC_AUX_SHARED is reported together with ICC_AUX.

Confidence Metric Details

The Confidence Metric is defined in terms of the total weight of signal activity data sources for both combinational and registered signals. Each signal has two data sources allocated to it; a toggle rate source and a static probability source.

The Confidence Metric Details section also indicates the quality of the signal toggle rate data to compute a power estimate. The confidence metric is low if the signal toggle rate data comes from poor predictors of real signal toggle rates in the device during an operation. Toggle rate data that comes from simulation, user-entered assignments on specific signals or entities are reliable. Toggle rate data from default toggle rates (for example, 12.5% of the clock period) or vectorless estimation are relatively inaccurate. This section gives an overall confidence rating in the toggle rate data, from low to high. This section also summarizes how many pins, registers, and combinational nodes obtained their toggle rates from each of simulation, user entry, vectorless estimation, or default toggle rate estimations. This detailed information helps you understand how to increase the confidence metric, letting you determine your own confidence in the toggle rate data.

Signal Activities

The Signal Activities section lists toggle rates and static probabilities assumed by power analysis for all signals with fan-out and pins. This section also lists the signal type (pin, registered, or combinational) and the data source for the toggle rate and static probability. By default, this section does not contain any data, but you can turn on the report with the **Write signal activities to report file** option on the **Power Analyzer Settings** page.

Intel recommends that you keep the **Write signal activities to report file** option turned off for a large design because of the large number of signals present. You can use the Assignment Editor to specify that activities for individual nodes or entities are reported by assigning an on value to those nodes for the **Power Report Signal Activities** assignment.

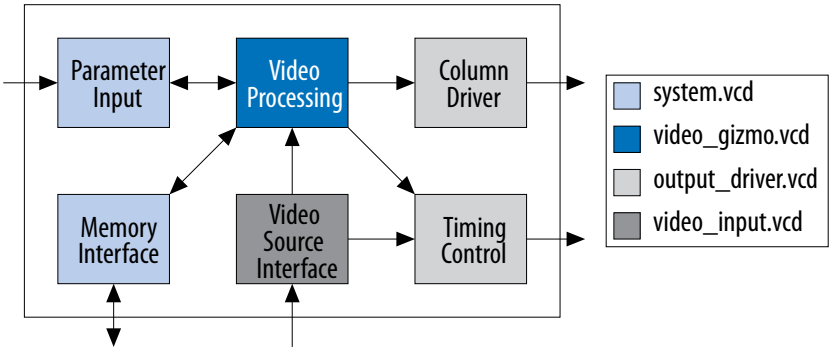
Messages

The Messages section lists the messages that the Quartus Prime software generates during the analysis.

1.5. Power Analysis in Modular Design Flows

In modular or hierarchical design flows you develop each design block separately, and then instantiate these blocks into a higher-level design to form a complete design. The Intel Quartus Prime software supports simulation and power analysis of the top-level design or individual blocks with the design.

Figure 20. Modular Simulation Flow

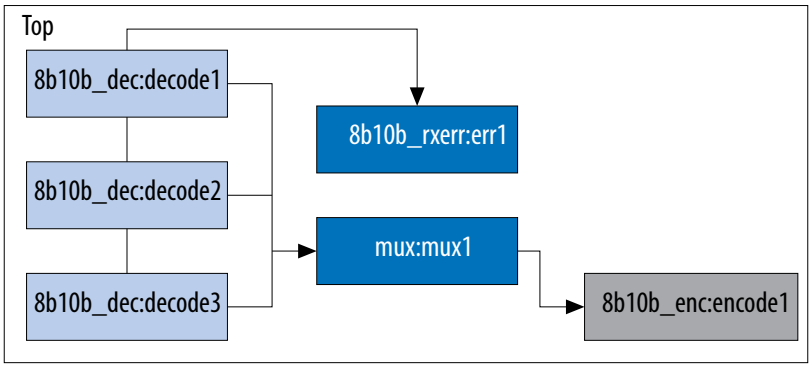


You can associate multiple `.vcd` simulation output files with specific node names, enabling the integration of partial design simulations into a complete design power analysis. When specifying multiple `.vcd` files for a node, more than one simulation file can contain signal activity information for the same signal. In those cases, the Power Analyzer follows these rules:

- When you apply multiple `.vcd` files to the same design node, the Power Analyzer calculates the signal activity as the equal-weight arithmetic average of each `.vcd`.
- When you apply multiple simulation files to design nodes at different levels in the design hierarchy, the signal activity in the power analysis derives from the simulation file that applies to the most specific design node.

The following figure shows an example of a hierarchical design:

Figure 21. Example Hierarchical Design



The top-level module of the design, called `Top`, consists of three 8b/10b decoders, followed by a `mux`. The software encodes the output of the `mux` to produce the final output of the top-level module. An error-handling module handles any 8b/10b decoding errors. The Top module contains the top-level entity of the design and any logic not defined as part of another module. The design file for the top-level module can be a wrapper for the hierarchical entities or can contain its own logic.

The following usage scenarios show common ways that you can simulate the design and import the `.vcd` into the Power Analyzer:

1.5.1. Complete Design Simulation Power Analysis Flow

You can simulate the entire gate-level design and generate a `.vcd` from a third-party simulator. The Power Analyzer can then import the `.vcd` (specifying the top-level design). The resulting power analysis uses the signal activities information from the generated `.vcd`, including those that apply to submodules, such as `decode [1-3]`, `err1`, `mux1`, and `encode1`.

1.5.2. Modular Design Simulation Power Analysis Flow

You can independently simulate the top-level design, and then import all the resulting `.vcd` files into the Power Analyzer. For example, you can simulate the `8b10b_dec` independent of the entire design and `mux`, `8b10b_rxerr`, and `8b10b_enc`. You can then import the `.vcd` files generated from each simulation by specifying the appropriate instance name. For example, if the files produced by the simulations are `8b10b_dec.vcd`, `8b10b_enc.vcd`, `8b10b_rxerr.vcd`, and `mux.vcd`, you can use the import specifications in the following table:

Table 5. Import Specifications

File Name	Entity
`8b10b_dec.vcd`	`Top|8b10b_dec:decode1`
`8b10b_dec.vcd`	`Top|8b10b_dec:decode2`
`8b10b_dec.vcd`	`Top|8b10b_dec:decode3`
`8b10b_rxerr.vcd`	`Top|8b10b_rxerr:err1`
`8b10b_enc.vcd`	`Top|8b10b_enc:encode1`
`mux.vcd`	`Top|mux:mux1`

The resulting power analysis applies the simulation vectors in each file to the assigned instance. Simulation provides signal activities for the pins and for the outputs of functional blocks. If the inputs to an instance are input pins for the entire design, the simulation file associated with that instance does not provide signal activities for the inputs of that instance. For example, an input to an instance such as `mux1` has its signal activity specified at the output of one of the decode instances.

1.5.3. Multiple Simulation Power Analysis Flow

You can perform multiple simulations of an entire design or specific modules of a design. For example, in the process of verifying the top-level design, you can have three different simulation testbenches: one for normal operation, and two for corner cases. Each of these simulations produces a separate `.vcd`. In this case, apply the different `.vcd` file names to the same top-level entity, as shown in the following table.

Table 6. Multiple Simulation File Names and Entities

File Name	Entity
normal.vcd	Top
corner1.vcd	Top
corner2.vcd	Top

The resulting power analysis uses an arithmetic average of the signal activities calculated from each simulation file to obtain the final signal activities used. If a signal `err_out` has a toggle rate of zero transition per second in `normal.vcd`, 50 transitions per second in `corner1.vcd`, and 70 transitions per second in `corner2.vcd`, the final toggle rate in the power analysis is 40 transitions per second.

If you do not want the Power Analyzer to read information from multiple instances and take an arithmetic average of the signal activities, use a `.vcd` that includes only signals from the instance that you care about.

1.5.4. Overlapping Simulation Power Analysis Flow

You can perform a simulation on the entire design, and more exhaustive simulations on a submodule, such as `8b10b_rxerr`. The following table lists the import specification for overlapping simulations:

Table 7. Overlapping Simulation Import Specifications

File Name	Entity
full_design.vcd	Top
error_cases.vcd	Top|8b10b_rxerr:err1

In this case, the software uses signal activities from `error_cases.vcd` for all the nodes in the generated `.vcd` and uses signal activities from `full_design.vcd` for only those nodes that do not overlap with nodes in `error_cases.vcd`. In general, the more specific hierarchy (the most bottom-level module) derives signal activities for overlapping nodes.

1.5.5. Partial Design Simulation Power Analysis Flow

You can perform a simulation in which the entire simulation time is not applicable to signal activity calculation. For example, if you run a simulation for 10,000 clock cycles and reset the chip for the first 2,000 clock cycles. If the Power Analyzer performs the signal activity calculation over all 10,000 cycles, the toggle rates are only 80% of their steady state value (because the chip is in reset for the first 20% of the simulation). In this case, you must specify the useful parts of the `.vcd` for power analysis. The **Limit VCD Period** option enables you to specify a start and end time when performing signal activity calculations.

1.5.5.1. Specifying Start and End Time for Signal Activity Calculations

To specify a start and end time for signal activity calculations using the **Limit VCD period** option, follow these steps:

1. In the Quartus Prime software, click **Assignments ➤ Settings**.
2. Under the **Category** list, click **Power Analyzer Settings**.
3. Turn on the **Use input file(s) to initialize toggle rates and static probabilities during power analysis** option.
4. Click **Add**.
5. In the **File name** and **Entity** fields, browse to the necessary files.
6. Under **Simulation period**, turn on **VCD file** and **Limit VCD period** options.
7. In the **Start time** and **End time** fields, specify the desired start and end time.
8. Click **OK**.

You can also use the following Tcl or `.qsf` assignment to specify `.vcd` files:

```
set_global_assignment -name POWER_INPUT_FILE_NAME "test.vcd" -section_id test.vcd
set_global_assignment -name POWER_VCD_FILE_START_TIME "10 ns" -section_id
test.vcd
set_global_assignment -name POWER_VCD_FILE_END_TIME "1000 ns" -section_id
test.vcd
set_instance_assignment -name POWER_READ_INPUT_FILE test.vcd -to test_design
```

1.5.6. Vectorless Estimation Power Analysis Flow

For some device families, the Power Analyzer automatically derives estimates for signal activity on nodes with no simulation or user-entered signal activity data.

Vectorless estimation statistically estimates the signal activity of a node based on the signal activities of nodes feeding that node, and on the actual logic function that the node implements. Vectorless estimation cannot derive signal activities for primary inputs. Vectorless estimation is accurate for combinational nodes, but not for registered nodes. Therefore, the Power Analyzer requires simulation data for at least the registered nodes and I/O nodes for accuracy.

1.6. Scripting Support

You can run procedures and create settings described in this chapter in a Tcl script. Alternatively, you can run procedures at a command prompt. For more information about scripting command options, refer to the Quartus Prime Command-Line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp
```

Related Information

Quartus Prime Pro Edition Settings File Reference Manual

1.6.1. Running the Power Analyzer from the Command–Line

The executable to run the Power Analyzer is `quartus_pow`. For a complete listing of all command–line options supported by `quartus_pow`, type the following command at a system command prompt:

```
quartus_pow --help
```

or

```
quartus_sh --qhelp
```

The following lists the examples of using the `quartus_pow` executable. Type the command listed in the following section at a system command prompt:

Note: These examples assume that operations are performed on Quartus Prime project called *sample*.

- To customize the name of the generated EPE File:

  ```
  quartus_pow sample --output_epe=sample.csv ←
  ```

- To customize the name of the generated Power and Thermal Calculator file:

  ```
  quartus_pow sample --output_ptc=sample.qptc  ←
  ```

- To instruct the Power Analyzer to use a `.vcd` as input (`sample.vcd`):

  ```
  quartus_pow sample --input_vcd=sample.vcd ←
  ```

- To instruct the Power Analyzer to use two `.vcd` files as input files (`sample1.vcd` and `sample2.vcd`), perform glitch filtering on the `.vcd` and use a default input I/O toggle rate of 10,000 transitions per second:

  ```
  quartus_pow sample --input_vcd=sample1.vcd --input_vcd=sample2.vcd \
  --vcd_filter_glitches=on --\
  default_input_io_toggle_rate=10000transitions/s
  ```

- To instruct the Power Analyzer not to use an input file, specify a default input I/O toggle rate of 60%, with vectorless estimation off, and a default toggle rate of 20% on all remaining signals:

  ```
  quartus_pow sample --no_input_file --default_input_io_toggle_rate=60% \
  --use_vectorless_estimation=off --default_toggle_rate=20%
  ```

Note: No command–line options are available to specify the information found on the **Operating Settings and Conditions** and **Power Analyzer Settings ➤ Thermal** pages. Use the Quartus Prime GUI to specify these options.

The `quartus_pow` executable creates a report file, *<revision name>*`.pow.rpt`. You can locate the report file in the main project directory. The report file contains the same information that the Power Analyzer Compilation Report.

Related Information

Viewing Power Analysis Reports on page 19

1.7. Power Analysis Revision History

The following revision history applies to this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	Updated Agilex 7 to Agilex FPGA portfolio in several topics to include support for Agilex 5 devices.
2023.12.04	23.4	• Revised the information for Device power characteristics in the *Settings for Power Analysis*.
2023.10.02	23.3	• Updated the section "Thermal Map Visualization" with additional information in *Viewing Power Analysis Reports*. • Updated the Power Analyzer Settings image and relevant description in *Settings for Power Analysis*. • Updated the product family name to "Intel Agilex 7."
2022.12.12	22.4	• Updated screenshot in *Power Analysis* topic. • Revised description of On-Chip Power Dissipation by Hierarchy report in *Viewing Power Analysis Reports* topic. • Removed Write out Power and Thermal Calculator file option description from *Settings for Power Analysis* topic. This setting is replaced by Power and Thermal Calculator File Name option. • Revised *Running the Power Analyzer* topic for Power and Thermal Calculator File Name option. • Revised *Running the Power Analyzer from the Command Line* for new GUI option and removal of obsolete commands.
2022.06.22	21.4	In the *Power Analysis Tools* topic, added statements about Quartus Prime Power Analyzer accuracy for Stratix 10 and Intel Agilex designs.
2021.12.13	21.4	• In the *Settings for Power Analysis* topic, modified a *Thermal Settings* entry in the *Power Analyzer and Thermal Settings* table. Specifically, changed *Apply Recommended Margin* to *Apply Additional Margin*, and modified the description accordingly. • In the *Viewing Power Analysis Reports* topic, added the *Crypto* block type to the *On-Chip Power Dissipation by Block Type* section.
2021.10.04	21.3	Recast the note in the *Power Analysis Tools* topic for greater clarity.
2021.03.29	21.1	• In the *Power Analysis* topic, updated the figure. • In the *Running the Power Analyzer* topic, modified step 3 and removed the figure following step 9. • Changed the title of the *Device Operating Condition Settings for Power Analysis* topic to *Settings for Power Analysis*, updated the existing figure and added an additional figure, recast the existing table and added an additional table. • In the *Generating Signal Activity Data for Power Analysis* topic, modified step 5 and the figure within step 6. • In the *Viewing Power Analysis Reports* topic, made minor changes to the first paragraph of the *Thermal Map Visualization* section. • In the *Running the Power Analyzer from the Command–Line* topic, modified the note at the bottom of the topic.
2020.12.07	20.3.0	Added note to the *Specifying the Default Toggle Rate* topic.
2020.10.05	20.3.0	• Added mention of gate-level simulation to the *Specifying Power Analyzer Input*, *Specifying Signal Activity Data*, *Using Simulation Signal Activity Data in Power Analysis*, and *Complete Design Simulation Power Analysis Flow* topics, • Added a sentence to the *RTL Simulation Limitation* and *Avoiding Simulation Node Name Match* topics. • Added a *Thermal Map Visualization* section to the *Viewing Power Analysis Reports* topic.

continued...

Document Version	Quartus Prime Version	Changes
2020.04.13	20.1.0	Added information about the Intel FPGA Power and Thermal Calculator to the following topics: • *Power Analysis* • *Power Analysis Tools* • *Running the Power Analyzer from the Command–Line*
2019.12.04	19.1.0	• Removed references to entity-specific toggle rates in "Specifying Toggle Rates for Specific Nodes." Toggle rates must be either global or node specific.
2019.08.02	19.1.0	• Clarified wording of statements about .vcd files in "Simulation Glitch Filtering" topic. • Corrected typo in "Specifying the Default Toggle Rate" topic. • Corrected typo in "Running the Power Analyzer from the Command Line" topic. • Improved explanation in "Generating Standard Delay Output for Power Analysis" topic.
2019.07.03	19.1.0	• Corrected broken links to Help.
2019.04.01	19.1.0	• Described new support for generation of SDO for use in power analysis. • Retitled some topic headings for greater clarity. • Changed the order of some topics for improved flow of information. • Added descriptions of Power Savings Summary and Parallel Compilation power analysis reports. • Added new Power Analysis flow diagrams.
2018.09.24	18.1.0	• General chapter reorganization. • Moved *Factors Affecting Power Consumption* to chapter: *Power Optimization*. • Updated figure: *Power Analyzer High-level Flow*. • Divided topic: *Types of Power Analysis* into two topics: *Power Estimations and Design Requirements* and *Design Activity and Power Analysis*. • Updated figure: *Power Analysis Tools from Design Concept through Design Implementation* and renamed to: *Estimation Accuracy for Different Inputs and Power Analysis Tools* • Removed content referring to device families not supported in *Intel Quartus Prime Pro Edition*.
2018.06.11	18.0.0	• In *Comparison of the EPE and the Intel Quartus Prime Power Analyzer*, updated the data output types that the Power Analyzer supports. • In *Comparison of the EPE and the Intel Quartus Prime Power Analyzer*, added row about estimation of transceiver power for features that you enable only through dynamic reconfiguration. • Specified features not supported by the Power Analyzer.
2017.05.08	17.0.0	Removed references to PowerPlay name. Power analysis occurs in the Intel Quartus Prime Power Analyzer.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Removed support for `.vcd` generation by the Compiler. Generate `.vcd` files for power estimation in your EDA simulator.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2014.12.15	14.1.0	• Removed Signal Activities from Full Post-fit Netlist (Timing) Simulation and Signal Activities from Full Post-fit Netlist (Zero Delay) Simulation sections as these are no longer supported. • Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings.

continued...

Document Version	Quartus Prime Version	Changes
2014.08.18	14.0a10.0	Updated "Current Drawn from Voltage Supplies" to clarify that for SoC devices or for Arria V SoC and Cyclone V SoC devices, there is no standalone ICC_AUX_SHARED current drawn information. The ICC_AUX_SHARED is reported together with ICC_AUX.
November 2012	12.1.0	• Updated "Types of Power Analyses" on page 8–2, and "Confidence Metric Details" on page 8–23. • Added "Importance of .vcd" on page 8–20, and "Avoiding Power Estimation and Hardware Measurement Mismatch" on page 8–24
June 2012	12.0.0	• Updated "Current Drawn from Voltage Supplies" on page 8–22. • Added "Using the HPS Power Calculator" on page 8–7.
November 2011	10.1.1	• Template update. • Minor editorial updates.
December 2010	10.1.0	• Added links to Quartus II Help, removed redundant material. • Moved "Creating PowerPlay EPE Spreadsheets" to page 8–6. • Minor edits.
July 2010	10.0.0	• Removed references to the Quartus II Simulator. • Updated Table 8–1 on page 8–6, Table 8–2 on page 8–13, and Table 8–3 on page 8–14. • Updated Figure 8–3 on page 8–9, Figure 8–4 on page 8–10, and Figure 8–5 on page 8–12.
November 2009	9.1.0	• Updated "Creating PowerPlay EPE Spreadsheets" on page 8–6 and "Simulation Results" on page 8–10. • Added "Signal Activities from Full Post-fit Netlist (Zero Delay) Simulation" on page 8–19 and "Generating a .vcd from Full Post-fit Netlist (Zero Delay) Simulation" on page 8–21. • Minor changes to "Generating a .vcd from ModelSim Software" on page 8–21. • Updated Figure 11–8 on page 11–24.
March 2009	9.0.0	• This chapter was chapter 11 in version 8.1. • Removed Figures 11-10, 11-11, 11-13, 11-14, and 11-17 from 8.1 version.
November 2008	8.1.0	• Updated for the Quartus II software version 8.1. • Replaced Figure 11-3. • Replaced Figure 11-14.
May 2008	8.0.0	• Updated Figure 11–5. • Updated "Types of Power Analyses" on page 11–5. • Updated "Operating Conditions" on page 11–9. • Updated "PowerPlay Power Analyzer Compilation Report" on page 11–31. • Updated "Current Drawn from Voltage Supplies" on page 11–32.

2. Power Optimization

The Quartus Prime software offers power-driven compilation to fully optimize device power consumption. Power-driven compilation focuses on reducing the design's total power consumption in synthesis and place-and-route stages.

This chapter focuses on design optimization options and techniques that help reduce core dynamic power and I/O power. In addition to these techniques, there are additional power optimization techniques available for specific devices, including Programmable Power Technology and Device Speed Grade Selection.

Related Information

- Power Analysis on page 4
- AN 711: Power Reduction Features in Arria 10 Devices
- Intel FPGA Literature and Technical Documentation

2.1. Factors Affecting Power Consumption

Understanding the following factors that affect power consumption allows you to use the Power Analyzer and interpret its results effectively:

2.1.1. Design Activity and Power Analysis

Power consumption of a device also depends on the design's activity over time. Static power (P_{STATIC}) is the thermal power that a chip dissipates independent of user clocks. P_{STATIC} includes leakage power from all FPGA functional blocks, except for I/O DC bias power and transceiver DC bias power, which are accounted for in the I/O and transceiver sections. Dynamic power is the additional power consumption of a device due to signal activity or switching.

2.1.2. Device Selection

Device families have different power characteristics. Many parameters affect the device family power consumption, including choice of process technology, supply voltage, electrical design, and device architecture.

Power consumption also varies in a single device family. A larger device with more transistors consumes more static power than a smaller device in the same family. In devices that employ global routing architectures, dynamic power can also increase with device size.

The choice of device package also affects the ability of the device to dissipate heat, and you may need to use a different cooling solution to comply with junction temperature constraints.

Process variation can affect power consumption. Process variation primarily impacts static power, because sub-threshold leakage current varies exponentially with changes in transistor threshold voltage. Therefore, you must consult device specifications for static power, and not rely on empirical observation. Process variation has a weak effect on dynamic power.

2.1.3. Environmental Conditions

The main environmental parameters affecting junction temperature are operating temperature and the cooling solution. Operating temperature primarily affects device static power consumption. Higher junction temperatures result in higher static power consumption. The device thermal power and cooling solution that you use must keep the device junction temperature within the maximum operating range for the device.

The following table lists the environmental conditions that influence power consumption.

Table 8. Environmental Conditions that Affect Power Consumption

Environmental Condition	Description
Airflow	Measures how quickly the device replaces heated air from the vicinity of the device with air at ambient temperature. You can either specify airflow as "still air" when you are not using a fan, or as the linear feet per minute rating of the fan in the system. Higher airflow decreases thermal resistance.
Heat Sink and Thermal Compound	A heat sink allows more efficient heat transfer from the device to the surrounding area because of its large surface area exposed to the air. The thermal compound that interfaces the heat sink to the device also influences the rate of heat dissipation. The case-to-ambient thermal resistance (θ_{CA}) parameter describes the cooling capacity of the heat sink and thermal compound employed at a given airflow. Larger heat sinks and more effective thermal compounds reduce θ_{CA}.
Junction Temperature	The junction temperature of a device is equal to: $T_{Junction}=T_{Ambient}+P_{Thermal}\cdot\theta_{JA}$ in which θ_{JA} is the total thermal resistance from the device transistors to the environment, in degrees Celsius per watt. The value θ_{JA} is equal to the sum of the junction-to-case (package) thermal resistance (θ_{JC}), and the case-to-ambient thermal resistance (θ_{CA}) of the cooling solution.
Board Thermal Model	The junction-to-board thermal resistance (θ_{JB}) is the thermal resistance of the path through the board, in degrees Celsius per watt. To compute junction temperature, you can use this board thermal model along with the board temperature, the top-of-chip θ_{JA} and ambient temperatures.

2.1.4. Device Resource Usage

Power consumption depends on the number and types of device resources that a design uses.

2.1.4.1. Number, Type, and Loading of I/O Pins

Output pins drive off-chip components, resulting in high-load capacitance that leads to a high-dynamic power per transition. Terminated I/O standards require external resistors that draw constant (static) power from the output pin.

2.1.4.2. Number and Type of Hard Logic Blocks

A design with more logic elements (LEs), multiplier elements, memory blocks, transceiver blocks, or HPS system tends to consume more power than a design with fewer circuit elements. The operating mode of each circuit element also affects its power consumption.

For example, a DSP block performing 18 × 18 multiplications and a DSP block performing multiply-accumulate operations consume different amounts of dynamic power, because of different amounts of charging internal capacitance on each transition. The operating mode of a circuit element also affects static power.

2.1.4.3. Number and Type of Global Signals

Global signal networks span large portions of the device and have high capacitance, resulting in significant dynamic power consumption. The type of global signal is important as well. Global clocks cover the entire device, whereas quadrant clocks only span one-fourth of the device. Clock networks that span smaller regions have lower capacitance and tend to consume less power. The location of the logic array blocks (LABs) driven by the clock network can also have an impact because the Quartus Prime software automatically disables unused branches of a clock.

2.1.5. Signal Activity

The behavior of each signal in the design is an important factor in estimating power consumption. To get accurate results from the power analysis, the signal activity must represent the actual operating behavior of the design.

The two most important behaviors of a signal are toggle rate and static probability.

2.1.5.1. Toggle Rate

The toggle rate of a signal is the average number of times that the signal changes value per unit of time. The units for toggle rate are transitions per second, and a transition is a change from `1` to `0`, or `0` to `1`.

Note: Inaccurate signal toggle rate data is the largest source of power estimation error.

Dynamic power increases linearly with the toggle rate as you charge the board trace model more frequently for logic and routing. The Quartus Prime software models full rail-to-rail switching. For high toggle rates, especially on circuit output I/O pins, the circuit can transition before fully charging the downstream capacitance. The result is a slightly conservative prediction of power by the Power Analyzer.

Note: The transceiver I/O toggle rate is determined by the XCVR data rate value specified in your IP catalog settings. Do not include transceiver I/O toggle rate in the default toggle rates that you specify in the Power Analyzer.

Related Information

Specifying the Default Toggle Rate on page 17

2.1.5.2. Static Probability

The static probability of a signal is the fraction of time that the signal is logic 1 during device operation. Static probability ranges from 0 (always at ground) to 1 (always at logic-high).

The static probability of input signals impacts the design's static power consumption, due to state-dependent leakage in routing and logic. This effect becomes more important for smaller geometries. In output I/O standards that drive termination resistors, the static power also depends on the static probability on I/O pins.

2.2. Design Space Explorer II for Power-Driven Optimization

The Design Space Explorer II (DSE II) tool allows you to find and implement the project settings that result in best power behavior.

The DSE II offers two options in **Exploration mode** that target power optimization: **Power (High Effort)** and **Power (Aggressive)**. In both cases, the target is an overall improvement in the design's power; specifically, reducing the total thermal power in the design.

When the optimization targets power, the DSE II runs the Quartus Prime Power Analyzer for every group of settings. The resultant reports help you debug the design and determine trade-offs between power requirements and performance optimization.

Related Information

- Design Space Explorer II
  In *Quartus Prime Pro Edition User Guide: Design Optimization*
- Launch Design Space Explorer Command (Tools Menu)
  In *Quartus Prime Help*

2.3. Power-Driven Compilation

The standard Quartus Prime compilation flow consists of Analysis and Synthesis, placement and routing, Assembly, and Timing Analysis. Power-driven compilation takes place at the Analysis and Synthesis and Place-and-Route stages.

Quartus Prime software settings that control power-driven compilation are located in the **Power optimization during synthesis** list in the **Advanced Settings (Synthesis)** dialog box, and the **Power optimization during fitting** list on the **Advanced Fitter Settings** dialog box. The following sections describes these power optimization options at the Analysis and Synthesis and Fitter levels.

2.3.1. Power-Driven Synthesis

Synthesis netlist optimization occurs during the synthesis stage of the compilation flow. You can apply these settings on a project or entity level.

The **Power Optimization During Synthesis** logic option determines how aggressively Analysis & Synthesis optimizes the design for power. To access this option at a project level, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.

Table 9. Power Optimization During Synthesis Options

Settings	Description	Optimization Techniques Included
Off	The Compiler does not perform netlist, placement, or routing optimizations to minimize power.	-
Normal compilation (Default)	The Compiler applies low compute effort algorithms to minimize power through netlist optimizations that do not reduce design performance.	• Memory block optimization • Power-aware logic mapping
Extra effort	Besides the techniques in the **Normal compilation** setting, the Compiler applies high-compute-effort algorithms to minimize power through netlist optimizations. Selecting this option might impact performance.	• Memory block optimization • Power-aware logic mapping • Power-aware memory balance

You can also control memory optimization options from the Quartus Prime **Settings** dialog box. The **Default Parameters** page allows you to edit the **Low_Power_Mode** parameter. The settings for this parameter are equivalent to the values of the **Power Optimization During Synthesis** logic options. The **Low_Power_Mode** parameter always takes precedence over the **Optimize Power for Synthesis** option for power optimization on memory.

Table 10. Low Power Mode Parameter Options

Parameter Value	Equivalent Setting in Power Optimization During Synthesis Logic Option
None	**Off**
Auto	**Normal compilation**
All	**Extra effort**

Related Information

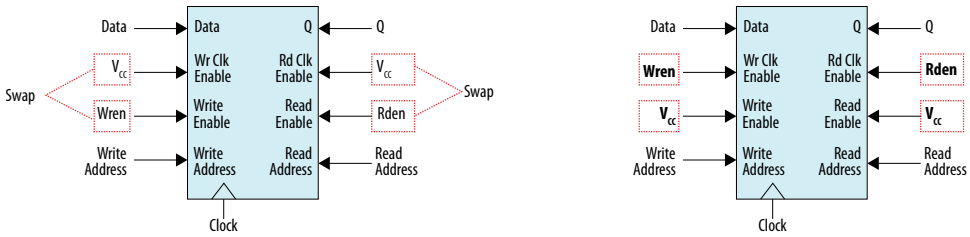
- Clock Enable in Memory Blocks on page 42
- Quartus Prime Compiler Settings on page 39

2.3.1.1. Memory Block Optimization

Memory optimization involves moving user-defined read/write enable signals to associated read-and-write clock enable signals for all memory types.

Memory blocks can represent a large fraction of total design dynamic power. Minimizing the number of memory blocks accessed during each clock cycle can significantly reduce memory power.

Figure 22. Memory Block Transformation



In the default implementation of a simple dual-port memory block, write-clock enable signals and read-clock enable signals connect to V_{CC}, making both read and write memory ports active during each clock cycle.

Memory transformation moves the read-enable and write-enable signals to the respective read-clock enable and write-clock enable signals. This technique reduces the design's memory power consumption, because memory ports are shut down when they are not accessed.

2.3.1.2. Power-Aware Logic Mapping

Power-aware logic mapping reduces power by rearranging the logic during synthesis to eliminate nets with high switching rates.

2.3.1.3. Power-Aware Memory Balancing

Power-aware memory balancing chooses the best configuration for a memory implementation and provides optimal power saving by determining the required number of memory blocks, decoder, and multiplexer circuits. When the design does not specify target-embedded memory blocks for the design's memory functions, the power-aware balancer automatically selects them during memory implementation.

The Compiler includes this optimization technique when the **Power Optimization During Synthesis** logic option is set to **Extra effort**.

There is a trade-off between power saved by accessing fewer memories and power consumed by the extra decoder and multiplexor logic. The Quartus Prime software automatically balances the power savings against the costs to choose the lowest power configuration for each logical RAM. The benchmark data shows that the power-driven synthesis can reduce memory power consumption by as much as 60% in Stratix devices.

You can also set the **MAXIMUM_DEPTH** parameter manually to configure the memory for low power optimization. This technique is the same as the power-aware memory balancer, but it is manual rather than automatic like the **Extra effort** setting in the **Power optimization** list. The **MAXIMUM_DEPTH** parameter always takes precedence over the **Optimize Power for Synthesis** options for power optimization on memory optimization. You can set the **MAXIMUM_DEPTH** parameter for memory modules manually in the Intel FPGA IP instantiation or in the IP Catalog.

Related Information

- RAM and ROM Parameter Settings
  In *Stratix 10 Embedded Memory User Guide*

- Maximum Block Depth Configuration
  In *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*

2.3.2. Power-Driven Fitter

The Quartus Prime software allows you to control the power-driven compilation setting of the Fitter on a project-wide basis. The **Advanced Fitter Settings** dialog box page provides the **Power optimization during Fitting** logic option, that determines how aggressively the Fitter optimizes the design for power.

Table 11. Power-Driven Fitter Option

Option	Description
Off	The Fitter does not perform optimizations to minimize power.
Normal compilation (Default)	The Fitter applies low compute effort algorithms to minimize power through placement and routing optimizations. These techniques do not reduce design performance. Includes DSP optimizations that create power-efficient DSP block configurations for DSP functions.
Extra effort	Besides the optimization techniques of the **Normal Compilation** option, the Fitter applies high compute effort algorithms to minimize power through placement and routing optimizations. These techniques might impact performance. The **Extra effort** setting for the Fitter requires extensive effort to optimize the design for power and can increase compilation time.

The **Extra effort** setting the Fitter works to minimize power even after the design meets timing requirements by moving the logic closer during placement to localize high-toggling nets and choosing routes with low capacitance.

The **Extra effort** setting uses a Value Change Dump (`.vcd`) file that guides the Fitter to fully optimize the design for power, based on the signal activity of the design. The best power optimization during fitting results from using the most accurate signal activity information. If there is no `.vcd` file, the Quartus Prime software estimates the signal activities from the settings in the **Power Analyzer Settings** page in the **Settings** dialog box, such as assignments, clock assignments, and vectorless estimation values.

Related Information

Assignment Editor Options on page 40

2.3.3. Area-Driven Synthesis

Using area optimization rather than timing or delay optimization during synthesis saves power because you use fewer logic blocks. Using less logic usually means less switching activity.

The Quartus Prime software provides **Speed**, **Balanced**, or **Area** for the **Optimization Technique** option. You can also specify this logic option for specific modules in your design with the Assignment Editor in cases where you want to reduce area using the **Area** setting (potentially at the expense of register-to-register timing performance) while leaving the default **Optimization Technique** setting at **Balanced** (for the best trade-off between area and speed for certain device families). The **Speed Optimization Technique** can increase the resource usage of your design if the constraints are too aggressive and can also result in increased power consumption.

Related Information

Assignment Editor Options on page 40

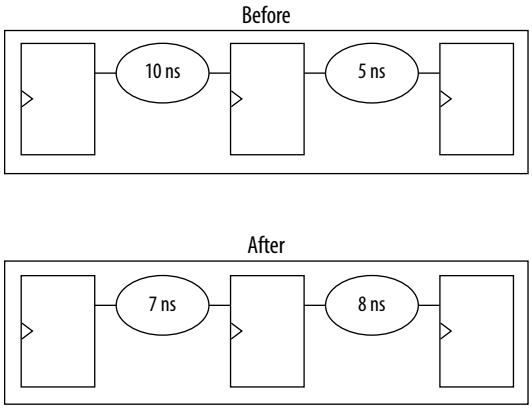
2.3.4. Gate-Level Register Retiming

You can also use gate-level register retiming to reduce circuit switching activity. Retiming shuffles registers across combinational blocks without changing design functionality.

The **Perform gate-level register retiming** option in the Quartus Prime software enables the movement of registers across combinational logic to balance timing, allowing the software to trade off the delay between critical and noncritical paths.

Retiming uses fewer registers than pipelining. In this example of gate-level register retiming, the 10 ns critical delay is reduced by moving the register relative to the combinational logic, resulting in the reduction of data depth and switching activity.

Figure 23. Gate-Level Register Retiming



Gate-level register retiming makes changes at the gate level. If you are using an atom netlist from a third-party synthesis tool, you must also select the **Perform WYSIWYG primitive resynthesis** option to undo the atom primitives to gates mapping (so that register retiming can be performed), and then to remap gates to Intel primitives.

Related Information

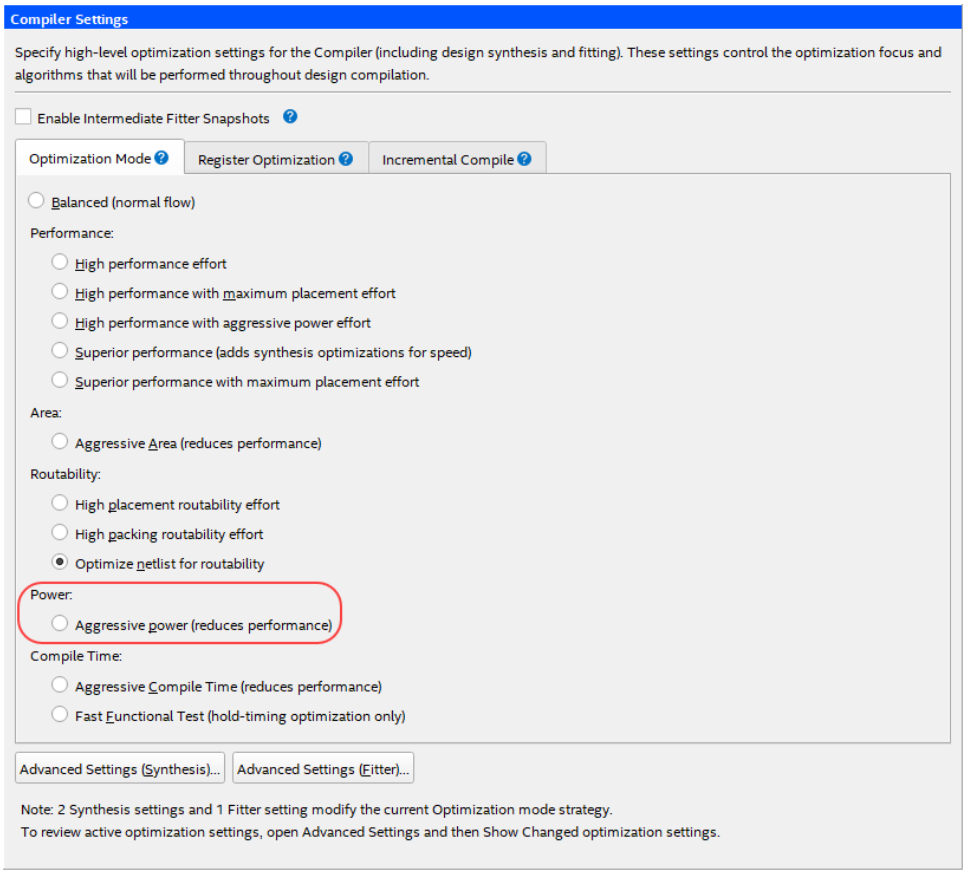
Netlist Optimizations and Physical Synthesis
In *Quartus Prime Pro Edition User Guide: Design Optimization*

2.3.5. Quartus Prime Compiler Settings

The Quartus Prime software provides settings that optimize power for the full design.

To set the optimization mode on the Quartus Prime software, click **Assignments ➤ Settings ➤ Compiler Settings**.

Figure 24. Compiler Settings



Aggressive Power (reduces performance)

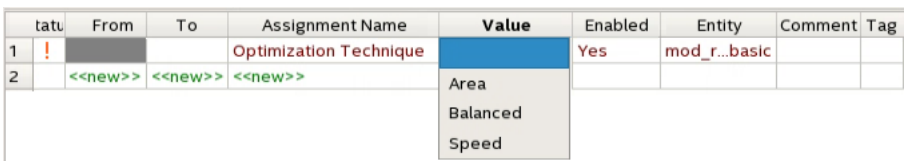
Makes aggressive effort to optimize synthesis for low power. The Compiler further reduces the routing usage of signals with the highest specified or estimated toggle rates, saving additional dynamic power but potentially affecting performance.

2.3.6. Assignment Editor Options

The Assignment Editor allows you to select Optimization Technique & Synthesis Power Optimization for individual modules. With this feature, you can focus on the parts of the design that require more work.

The **Optimization Technique** logic option specifies the overall optimization goal for Analysis & Synthesis: attempt to maximize performance or minimize logic usage.

Figure 25. Optimization Technique Options



The **Power Optimization During Synthesis** logic option determines how aggressively Analysis & Synthesis optimizes the design for power.

Figure 26. Power Optimization During Synthesis Options

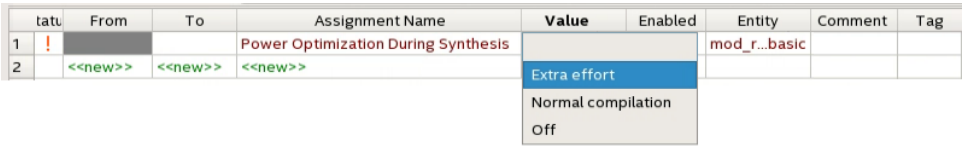


Table 12. Power Optimization During Synthesis Options

Settings	Description	Optimization Techniques Included
Off	The Compiler does not perform netlist, placement, or routing optimizations to minimize power.	-
Normal compilation (Default)	The Compiler applies low compute effort algorithms to minimize power through netlist optimizations that do not reduce design performance.	• Memory block optimization • Power-aware logic mapping
Extra effort	Besides the techniques in the **Normal compilation** setting, the Compiler applies high-compute-effort algorithms to minimize power through netlist optimizations. Selecting this option might impact performance.	• Memory block optimization • Power-aware logic mapping • Power-aware memory balance

Related Information

- Area-Driven Synthesis on page 38
- Power-Driven Fitter on page 38

2.4. Design Guidelines

During FPGA design implementation, you can apply the following design techniques to reduce power consumption. The results of these techniques are different from design to design.

2.4.1. Clock Power Management

Clocks represent a significant portion of dynamic power consumption due to their high switching activity and long paths. Actual clock-related power consumption is higher, because the power consumption of a block includes local clock distribution within logic, memory, and DSP or multiplier blocks.

The Quartus Prime software optimizes clock routing power automatically, enabling only those portions of the clock network that are necessary to feed downstream registers.

2.4.1.1. Clock Gating

For designs containing logic that is not operational 100% of the time, you can reduce power consumption by gating (disabling) the clock that feeds that logic.

This solution requires that the logic in question have its own dedicated clock source. Clock duplication (such as introducing a duplicate PLL clock output) is necessary if the logic in question does not have its own dedicated clock source.

The following topics describe methods for gating clock networks.

2.4.1.1.1. Root Clock Gate

You can gate each clock network dynamically at the root level, using the Clock Control Intel FPGA IP Core.

Refer to the *Root Clock Gate* section of the *Clocking and PLL User Guide* for your Intel device.

2.4.1.1.2. Sector Clock Gate

You can gate each clock network dynamically at the clock sector level using the Clock Control Intel FPGA IP Core.

Refer to the *Sector Clock Gate* section of the *Clocking and PLL User Guide* for your Intel device.

2.4.1.1.3. I/O PLL Clock Gate

You can dynamically gate each of the device's I/O PLL output counters, using I/O PLL Reconfiguration.

Refer to the *I/O PLL Clock Gate* section of the *Clocking and PLL User Guide* for your Intel device.

2.4.1.2. Clock Enable in Memory Blocks

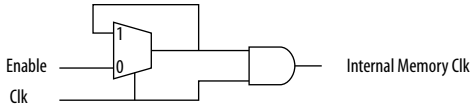
In memory blocks, power consumption is tied to the clock rate, and is insensitive to the toggle rate on the data and address lines. Memory consumes approximately 20% of the core dynamic power in typical designs.

When a memory block is clocked, a sequence of timed events occur within the block to execute a read or write. The circuitry that the clock controls consumes the same amount of power, independent of changes in address or data from one cycle to the next. Thus, the toggle rate of input data and the address bus have no impact on memory power consumption.

The key to reducing memory power consumption is to reduce the number of memory clocking events. You can achieve this reduction through network-wide clock gating, or on a per-memory basis through use of the clock enable signals on the memory ports.

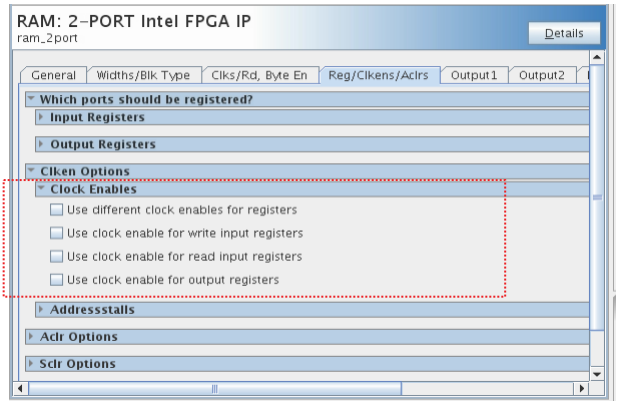
Figure 27. Memory Clock Enable Signal

Logical view of the internal clock of the memory block. Use the appropriate enable signals on the memory to make use of the clock enable signal instead of gating the clock.



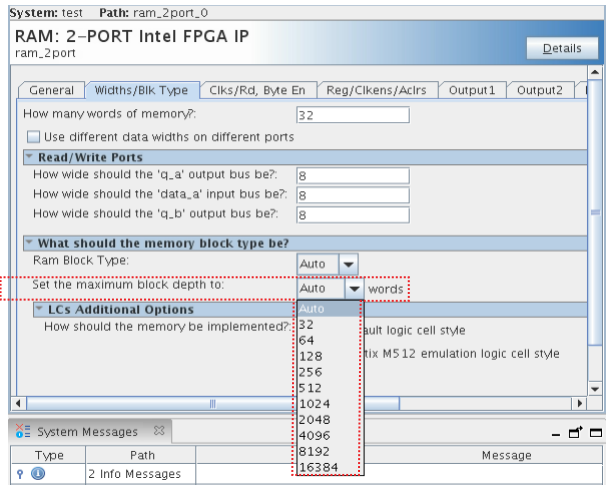
The clock enable signal enables the memory only when necessary, and shuts down for the rest of the time, reducing the overall memory power consumption. You include these enable signals when generating the memory block function.

Figure 28. Clock Enable in RAM 2-Port



The Quartus Prime software automatically chooses the best design memory configuration for optimal power. However, you can set the **MAXIMUM_DEPTH** parameter for memory modules during the IP core instantiation.

Figure 29. RAM 2-Port Maximum Depth



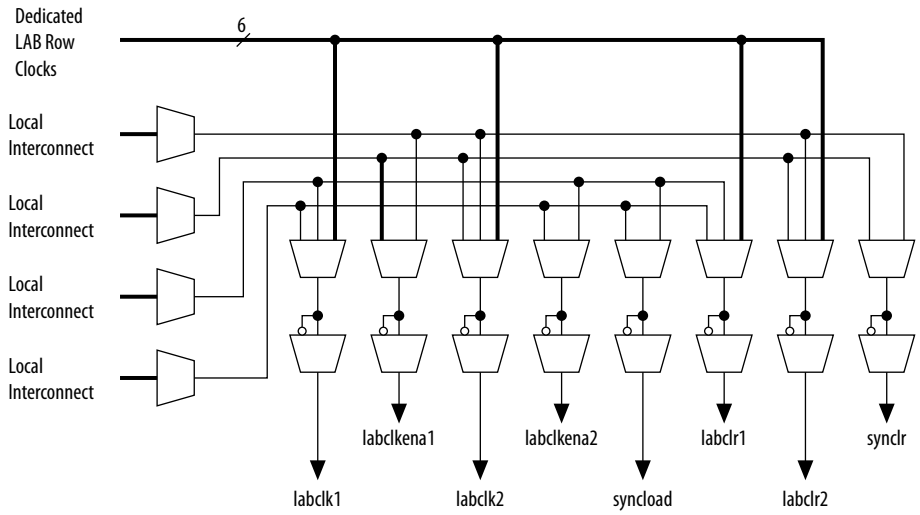
Related Information

- Power-Driven Compilation on page 35
- Clock Power Management on page 41
- Clocking Modes and Clock Enable
  In *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*

2.4.1.3. LAB Clock Power

Another contributor to clock power consumption are LAB clocks, which distribute clock to the registers within a LAB. LAB clock power can be the dominant contributor to overall clock power.

Figure 30. LAB-Wide Control Signals



To reduce LAB-wide clock power consumption without disabling the entire clock tree, use the LAB-wide clock enable to gate the LAB-wide clock. The Quartus Prime software automatically promotes register-level clock enable signals to the LAB-level. A shared gated clock controls all registers within an LAB that share a common clock and clock enable. To take advantage of these clock enables, use a clock enable construct in the relevant HDL code for the registered logic.

2.4.1.3.1. LAB-Wide Clock Enable Example

This VHDL code makes use of a LAB-wide clock enable. This clock-gating logic is automatically turned into an LAB-level clock enable signal.

```
IF clk'event AND clock = '1' THEN
     IF logic_is_enabled = '1' THEN
       reg <= value;
     ELSE
       reg <= reg;
     END IF;
END IF;
```

2.4.1.4. Clock Enables

Use clock enables instead of the gated clocks shown below:

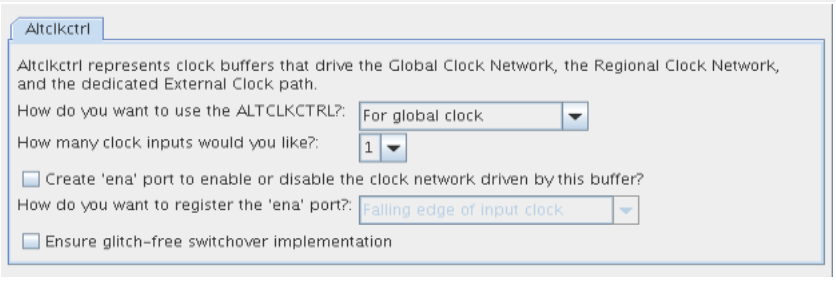
Avoid the following form of gated clocks:

```
assign clk_gate = clk1 & gateA & gateB;
always @ (posedge clk_gate) begin
   sr[N-1:1] <= sr[N-2:0];
   sr[0]<=din1;
end
```

Use clock enables, such as the following:

```
assign enable = gateA & gateB;
always @(posedge clk2) begin
   if (enable) begin
      sr[N-1:1] <= sr[N-2:0];
```

```
      sr[0]<=din2;
   end
end
```



Reduce LAB-wide clock power consumption without disabling the entire clock tree, use the LAB-wide clock enable to gate the LAB-wide clock.

```
always @(posedge clk)
begin
   if (ena)
      temp <= dataa;
   else
      temp <= temp;
   end
end
```

2.4.1.5. Global Signals

Intel FPGAs have different kinds of global signal resources available. Global signals can span the entire chip or smaller regions. Choose the clock networks that can cover all the fanout on a specific domain. For example, you can reduce clock power by switching from a clock network that spans the entire chip to one quarter of the chip, provided all the fanout for that clock is within that region of the chip.

2.4.1.5.1. Viewing Clock Details in the Chip Planner

1. Open the Chip Planner (**Tools ➤ Chip Planner**).
2. In the **Task** pane, under **Clock Reports**, double-click **Report Clock Details**.

Figure 33. Chip Planner Task Pane

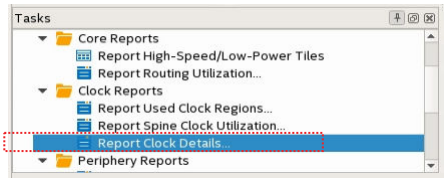
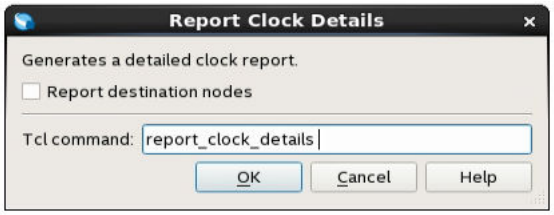


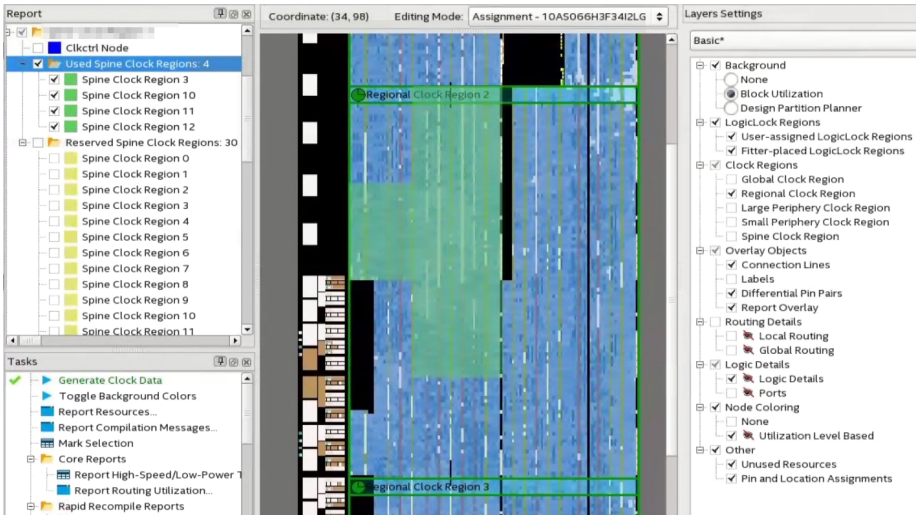
Figure 34. Report Clock Details



3. Click **OK**.
   The **Report** pane generates the **Clock** folder.
4. Expand the **Clock** folder and select **Used spine clock regions** to highlight on the Chip planner.
5. In the **Layers Settings** pane, turn on **Regional/Periphery clock region** to see whether used spine clock regions are within.

Figure 35. Clock Highlight in Chip Planner

This example uses a Regional clock Region instead of a global signal.



2.4.1.6. Merge Clocks

Evaluate the possibility of merging clocks and PLLs in the design.

Design	2clks & 2PLLs	1 Clk & 1 PLL
Oc_dma_stamp25	6.079W	5.46W

- 2clks & 2PLLs

  Clk1:350Mhz, Fanout 46788

  Clk2: 365Mhz, Fanout 2450
- 1Clk & 1PLL

  Merge clks

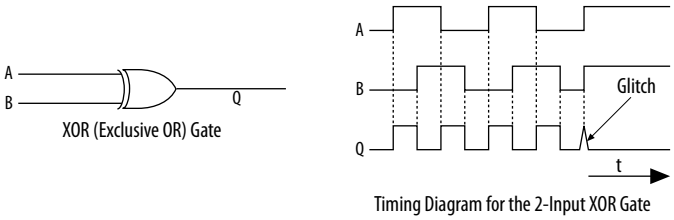
  clk: 365Mhz, Fanout 51277

2.4.2. Pipelining and Retiming

Glitches are unnecessary and unpredictable temporary logic switches at the output of combinational logic. Designs with glitches consume more power, because of faster switching activity. A glitch usually occurs when there is a mismatch in input signal timing, leading to unequal propagation delay.

For example, consider a 2-input XOR gate where one input changes from 1 to 0, and moments later the other input changes from 0 to 1. For a short time, both inputs become 1 (high), resulting in 0 (low) at the output of the XOR gate. Then, when the second input transition takes place, the XOR gate output becomes 1 (high). Therefore, before the output becomes stable, the input delay produces a glitch in the output.

Figure 36. XOR Gate Showing Glitch at the Output

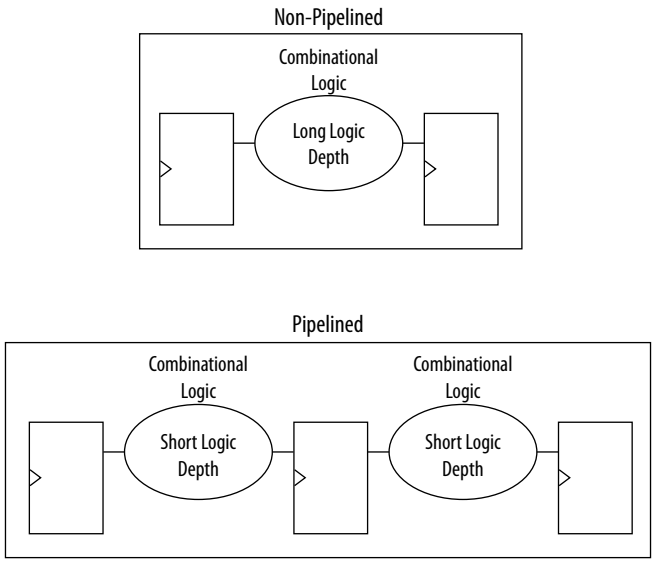


A glitch can propagate to subsequent logic and create unnecessary switching activity, increasing power consumption. Circuits with many XOR functions, such as arithmetic circuits or cyclic redundancy check (CRC) circuits, tend to have many glitches if there are several levels of combinational logic between registers.

Registers stop glitches from propagating through combinational paths. Pipelining is a technique that breaks combinational paths by inserting registers. By reducing logic-level numbers between registers, pipelining can result in higher clock speed operations. However, pipelining increases the latency of a circuit in terms of the number of clock cycles to a first result.

The following figure shows how pipelining breaks a long combinational path.

Figure 37. Pipelining Example



This reduction in switching activity lowers power dissipation in combinational logic. However, for designs with few glitches, pipelining can increase power consumption by adding unnecessary registers. Pipelining can also increase resource utilization.

2.4.3. Architectural Optimization

Design-level architectural optimizations allow you to take advantage of device architecture features. These features include dedicated memory, DSPs, or multiplier blocks that can perform memory or arithmetic-related functions. You can reduce power consumption by choosing blocks in place of LUTs. For example, you can build large shift registers from RAM-based FIFO buffers instead of building the shift registers from the LE registers.

Related Information

Timing Closure and Optimization
In *Quartus Prime Pro Edition User Guide: Design Optimization*

2.4.4. I/O Power Guidelines

The Power Analyzer calculates I/O power using the default capacitive load set for the I/O standard in the **Capacitive Loading** page of the **Device and Pin Options** dialog box. Any other components defined in the board trace model are not taken into account for the power measurement.

Nonterminated I/O Standards

Nonterminated I/O standards such as LVTTL and LVCMOS have a rail-to-rail output swing. The voltage difference between logic-high and logic-low signals at the output pin is equal to the V_{CCIO} supply voltage. If the capacitive loading at the output pin is known, the following expression determines the dynamic power consumed in the I/O buffer:

$$P = \frac{F \cdot C \cdot V^{2}}{2}$$

where:

- F is the output transition frequency
- C is the total load capacitance being switched
- V is equal to V_{CCIO} supply voltage

Because of the quadratic dependence on V_{CCIO}, lower voltage standards consume significantly less dynamic power.

Transistor-to-transistor logic (TTL) I/O buffers consume very little static power. As a result, the total power that a LVTTL or LVCMOS output consumes is highly dependent on load and switching frequency.

Resistively Terminated I/O Standards

In resistively terminated I/O standards like SSTL and HSTL, the output load voltage swings by a small amount around a bias point. The dynamic power equation above is valid as well, but V is the actual load voltage swing. This voltage is much smaller than V_{CCIO}, resulting in lower dynamic power when comparing to nonterminated I/O under similar conditions.

Resistively terminated I/O standards dissipate significant static (frequency-independent) power, because the I/O buffer is constantly driving current into the resistive termination network. However, the lower dynamic power of these I/O standards means they often have lower total power than LVCMOS or LVTTL for high-frequency applications. As a best practice, when using resistively terminated standards choose the lowest drive strength I/O setting that meets the speed and waveform requirements to minimize I/O power.

You can save a small amount of static power by connecting unused I/O banks to the lowest possible V_{CCIO} voltage.

Related Information

Managing Device I/O Pins
In *Quartus Prime Pro Edition User Guide: Design Constraints*

2.4.5. Dynamically Controlled On-Chip Terminations (OCT)

Dynamic OCT enables series termination (RS) and parallel termination (RT) to dynamically turn on/off during the data transfer. This feature is especially useful in FPGAs with external memory interfaces, such as interfacing with DDR memories.

Dynamic OCT eliminates the constant DC power that parallel termination consumes when transmitting data, reducing power consumption when compared to conventional termination. Parallel termination is extremely useful for applications that interface with external memories where I/O standards, such as HSTL and SSTL, are used. Parallel termination supports dynamic OCT, which is useful for bidirectional interfaces.

For more information about dynamic OCT in specific devices, refer to the *Stratix 10 General Purpose I/O User Guide* or the *Arria 10 Core Fabric and General Purpose I/O Handbook*.

Example 1. Example: Power Saving for a DDR3 Interface with OCT

The static current consumed by parallel OCT is equal to the V_{CCIO} voltage divided by 100 W. For DDR3 interfaces with SSTL-15, the static current per pin is:
$\frac{1.5V}{100W} = 15mA$

Therefore, the static power is:
$1.5V \times 15mA = 22.5mW$

For an interface with 72 DQ and 18 DQS pins, the static power is:
$90pins \times 2.25mW = 2.025W$

Dynamic parallel OCT disables parallel termination during write operations, so if writing occurs 50% of the time, the power saved by dynamic parallel OCT is:
$50\% \times 2.025W = 1.0125W$

For more information about dynamic OCT in Stratix IV devices, refer to the chapter in the *Stratix IV Device Handbook*.

Related Information

- Dynamic OCT
  In *Arria 10 Core Fabric and General Purpose I/O Handbook*
- Dynamic OCT
  In *Stratix 10 General Purpose I/O User Guide*

2.4.6. Memory Optimization (M20K/MLAB)

M20K memory blocks represent a big part of the power consumption in a design. The Fitter RAM Summary Report displays the utilization of the memory blocks in different parts of the design.

Figure 38. Fitter RAM Summary Report

Fitter RAM Summary

<<Filter>>

	Port A Depth	rt A Wid	rt B Dep	rt B Wi	Input Reg	Output Re	3 Input Regi	t B Output R	Size	M20K blocks	Fits in MLABs
n1:FIFOram|ALTSYNCRAM	32	1	32	1	yes	no	yes	yes	32	1	Yes
_generated|ALTSYNCRAM	16	2	16	2	yes	no	yes	no	32	1	Yes
n1:FIFOram|ALTSYNCRAM	2	24	2	24	yes	no	yes	yes	48	1	Yes
n1:FIFOram|ALTSYNCRAM	2	24	2	24	yes	no	yes	yes	48	1	Yes
n1:FIFOram|ALTSYNCRAM	128	1	128	1	yes	no	yes	yes	128	1	Yes
5n1:FIFOram|ALTSYNCRAM	16	9	16	9	yes	no	yes	yes	144	1	Yes

Some guidelines to optimize the use of memories are:

- Port shallow memories from M20K to MLAB.

  For example, implement in HDL with `ramstyle` attribute:

  ```
  (* ramstyle = "MLAB" *) reg [0:7] my_ram[0:63];
  ```

- Avoid read-during-write behavior and set to **Don't care** (at the HDL level) wherever possible.

  Read-during-write behavior impact the power of single-port and bidirectional dual-port RAMs. **Don't care** allows an optimization that sets the read-enable signal to the inversion of the existing write-enable signal (if one exists). This allows the core of the RAM to shut down, which prevents switching, saving a significant amount of power.

- Pack input/output registers in M20K.

2.4.6.1. Implementation

Table 13. Single-port Embedded Memory Configurations for Arria 10 Devices

This table lists the maximum configurations that single-port RAM and ROM modes support.

Memory Block	Depth (bits)	Programmable Width
MLAB	32	x16, x18, or x20
	64 (2)	x8, x9, x10
M20K	512	x40, x32
	1K	x20, x16
	2K	x10, x8
	4K	x5, x4
	8K	x2
	16K	x1

Figure 39. Power numbers from EPE

						Port A				Port B					Thermal Power (W)		
Module	RAM Type	# RAM Blocks	Data Width	RAM Depth	RAM Mode	Clock Freq (MHz)	Enable %	Read %	Write %	Clock Freq (MHz)	Enable %	Read %	Write %	Toggle %	Routing	Block	Total
	M20K	1	40	32	Simple Dual Port	384.0	100%	0%	100%	384.0	100%	100%	0%	12.5%	0.000	0.005	0.005
	MLAB	2	20	32	Simple Dual Port	384.0	100%	0%	100%	384.0	100%	100%	0%	12.5%	0.001	0.002	0.002

2.4.6.2. Rd/Wr Enables

Dedicated RAM blocks dissipate most energy whenever the RAM is accessed for a read or write cycle. You can save power by adding Read/Write enable.

(2) Supported through software emulation and consumes additional MLAB blocks.

						Port A				Port B					Thermal Power (W)		
Module	RAM Type	# RAM Blocks	Data Width	RAM Depth	RAM Mode	Clock Freq (MHz)	Enable %	Read %	Write %	Clock Freq (MHz)	Enable %	Read %	Write %	Toggle %	Routing	Block	Total
168	M20K	144	40	512	Simple Dual Port	384.0	100%	0%	100%	384.0	100%	100%	0%	12.5%	0.046	0.668	0.714
168	M20K	144	40	512	Simple Dual Port	384.0	100%	0%	50%	384.0	100%	50%	0%	12.5%	0.023	0.362	0.385

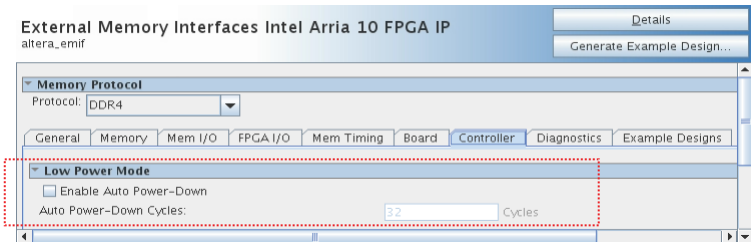
2.4.7. DDR Memory Controller Settings

The External Memory Interfaces Arria 10 FPGA IP provides low power mode settings. These settings put DDR memory in power saving mode when the controller is idle, providing power savings on external memory DDR. The **Enable Auto Power-Down** and **Auto Power-Down Cycles** settings enable this capability.

Low Power Mode Settings

- **Enable Auto Power-Down**—directs the controller to place the memory device in power-down mode after a specific number of idle controller clock cycles. You can configure the idle wait time. All ranks must be idle to enter auto power-down.
- **Auto Power-Down Cycles**—specifies the number of cycles the controller must be IDLE before entering the power down state. You determine the number based on the traffic pattern. If the number is too small, the control enters power down too frequently, affecting efficiency. The Arria 10 device family supports from 1 to 65534 cycles.

Figure 40. Arria 10 EMIF Controller Parameters



Related Information

Arria 10 EMIF IP DDR3 Parameters: Controller
In *External Memory Interfaces Arria 10 FPGA IP User Guide*

2.4.8. DSP Implementation

When you maximize the packing of DSP blocks, you reduce Logic Utilization, power consumption, and increase efficiency. The HDL coding style grants you control of the DSP resources available in the FPGA.

Example 2. Implement Multiplier + Accumulator in 1 DSP

```
always @ (posedge clk)
begin
   if (ena)
   begin
      dataout <= dataa * datab + datac * datad;
   end
end
```

Example 3. Implement multiplication in 2 DSPs and the adder in LABs

```
always @ (posedge clk)
begin
   if (ena)
   begin
      mult1 <= dataa * datab;
      mult2 <= datac * datad;
   end
end
always @(posedge clk)
begin
   if (ena)
   begin
      dataout <= mult1 + mult2
   end
end
```

Related Information

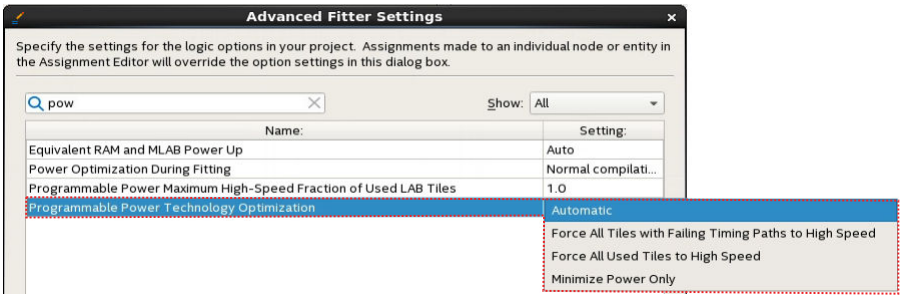
Inferring Multipliers and DSP Functions
In *Quartus Prime Pro Edition User Guide: Design Recommendations*

2.4.9. Reducing High-Speed Tile (HST) Usage

High-Speed tiles are available in the Arria 10 design family.

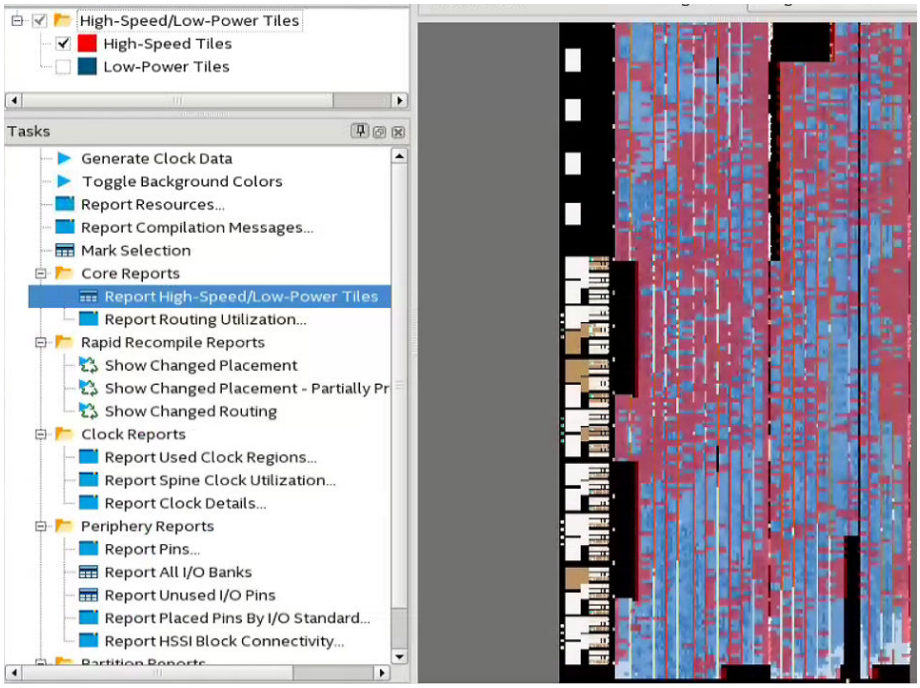
1. In the **Advanced Fitter Settings** pane, The **Programmable Power Technology Optimization** logic option controls how the fitter configures tiles to operate in high-speed mode or low-power mode. Select **Minimize Power Only**.

Figure 41. Programmable Power Technology Optimization



2. Identify entity modules that use HST by plotting entity modules and HST heatmap on the Chip Planner and modify the floorplan to reduce usage.

Figure 42. Entity Modules and HST Heatmap on the Chip Planner



2.4.10. Unused Transceiver Channels

Transceivers in the device degrade over time unless you preserve them. The Quartus Prime software generates a warning message if a design contains unused XCVRs.

You do not need to preserve transceivers under 8Gbps. For transceivers over 8Gbps, the best practice is to preserve if there is a possibility for future usage. Otherwise, you can turn the transceivers off. You enable unused transceivers through dynamic reconfiguration or a new device programming file.

2.4.11. Periphery Power reduction XCVR Settings

2.4.11.1. Transceiver Settings

- Use min VCCR/T possible (depending on data rate).
- Certain devices have DFE ON by default. If possible, turn off the channel, This depends on the how lossy is the channel.
- Turn off PDN compensation.
  This setting induces jitter, which is necessary to check system tolerance.
- Use one equalizer stage.

DFE	Adaptation	Equalizer Stage	Transmitter High-Speed Compensation
Disabled	Disabled	Non-S1 Mode	Disabled
Disabled	Disabled	Non-S1 Mode	Enabled
Disabled	Disabled	N/A	Enabled

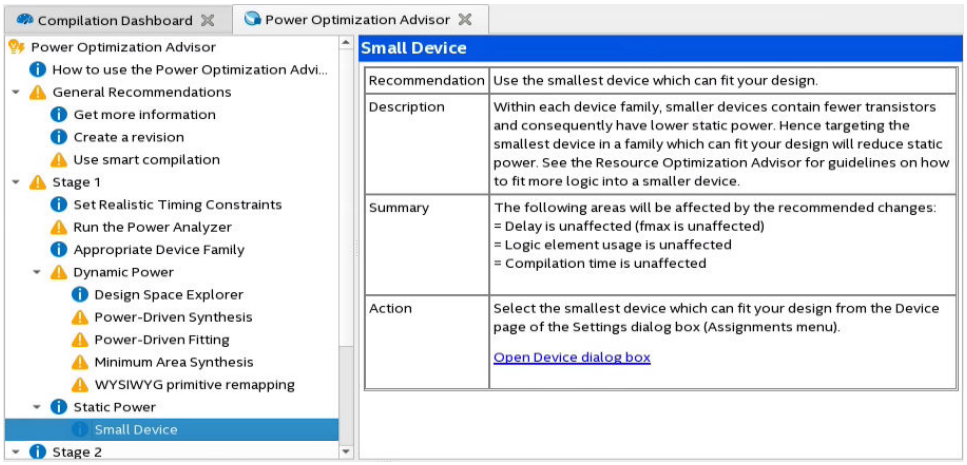
2.4.11.2. I/O Current Strength

As a best practice, choose a low voltage I/O standard and the lowest drive strength that meets the speed requirements.

2.5. Power Optimization Advisor

The Quartus Prime Power Optimization Advisor provides advice and recommendations based on the current design project settings and assignments. You run the Advisor after the Power Analyzer.

Figure 43. Power Optimization Advisor



The Power Optimization Advisor organizes the recommendations into stages that suggest the implementation order. Each recommendation includes a description, summary of the effect of the recommendation, and the action required to make the appropriate setting.

An icon indicates whether each recommended setting is made in the current project. Checkmark icons appear next to recommendations that are already implemented, warning icons appear next to recommendations that are not followed for this compilation. Information icons indicate general suggestions.

Recommendations include a link to the location in the Quartus Prime GUI where you can change the setting. After implementing the recommended changes, recompile your design. You can verify power results with the Power Analyzer.

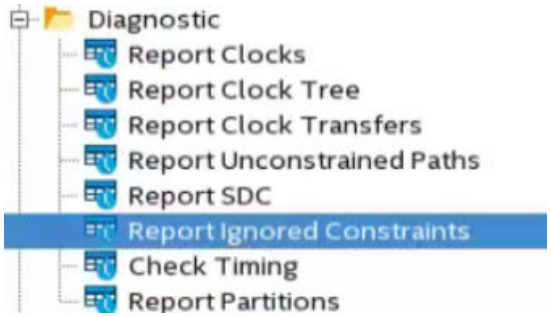
2.5.1. Set Realistic Timing Constraints

Timing requirements are too high, the Compiler increases HST Usage. In addition, the Fitter efforts focus more in timing than power optimization.

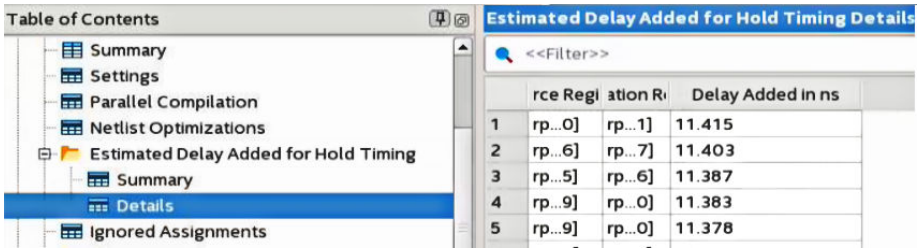
2.5.1.1. Find Timing Information

- To find False or Multi-Cycle Paths, click **Report Ignored Constraints** in the Timing Analyzer **Tasks** pane.

Figure 44. Report Ignored Constraints



- To see a list of the 10 paths with highest delay in the design, in the **Reports** pane find **Fitter Summary Report ➤ Estimate Delay Added for Hold Timing ➤ Details**.



2.5.2. Appropriate Device Family

Choose a device family with the dynamic and static power characteristics best suited to your application.

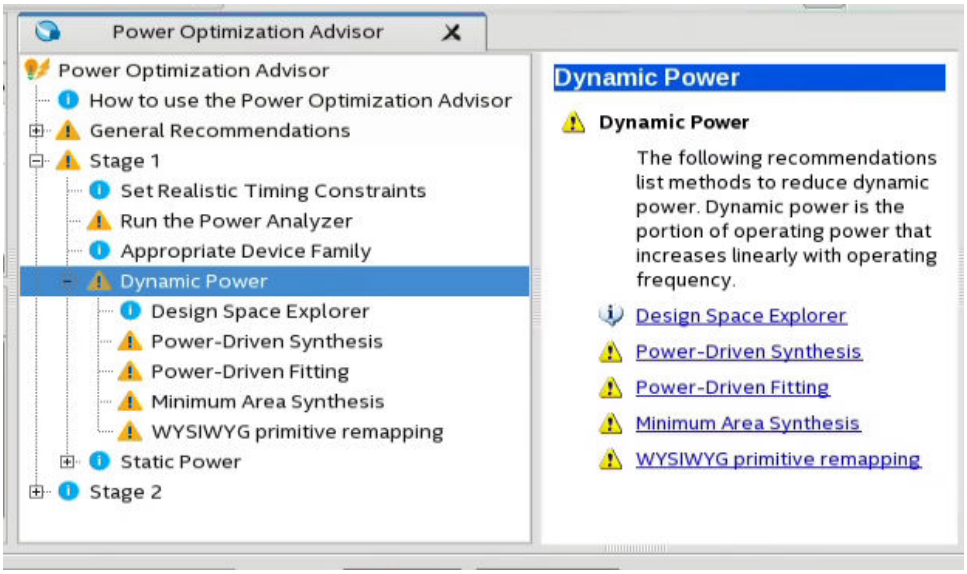
Related Information

Device Selection on page 32

2.5.3. Dynamic Power

The recommendations in this section can reduce dynamic power.

Figure 45. Dynamic Power Recommendations in the Power Optimization Advisor



Related Information

- Design Space Explorer II for Power-Driven Optimization on page 35
- Power-Driven Synthesis on page 35
- Power-Driven Fitter on page 38
- Area-Driven Synthesis on page 38

2.5.4. Static Power

The recommendations in this section can reduce static power dissipation. Static power is the frequency independent power that a design dissipates, even when the design clocks are stopped.

Small Device

Use the smallest device which can fit your design.

Related Information

Device Selection on page 32

2.5.5. Appropriate I/O Standards

Choose appropriate I/O Standards to minimize design power.

Related Information

I/O Power Guidelines on page 48

2.5.6. Use RAM Blocks

Implement RAMs and medium to large shift registers in RAM blocks instead of logic cell registers.

Related Information

Memory Optimization (M20K/MLAB) on page 50

2.5.7. Shut Down RAM Blocks

Use the clock enable, read enable and write enable ports on RAM blocks to shut them down during cycles in which the RAM is not read or written. If your design does not depend on a specific read result when reading and writing the same address, then specify "don't care" for the read-during-write parameter in the RAM IP Catalog.

Related Information

- Clock Enable in Memory Blocks on page 42
- Memory Optimization (M20K/MLAB) on page 50

2.5.8. Clock Enables on Logic

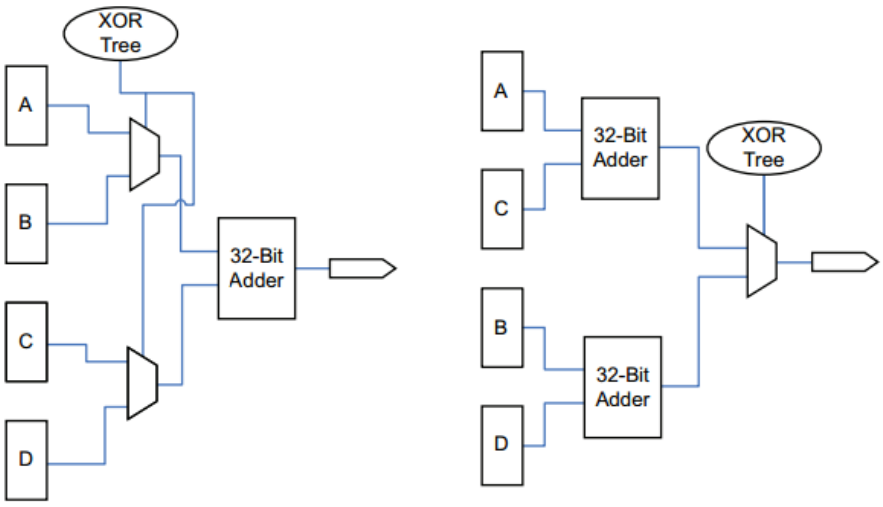
Another technique for power reduction is gating clocks when the logic does not require them. Even though you can build clock-gating logic, this approach can generate clock glitches in FPGAs using ALMs or LEs.

2.5.9. Pipeline Logic to Reduce Glitching

Long chains of cascaded logic blocks can create glitches due to path delay differences between the input signals. Inserting Flip-Flops to cut these long chains terminates the propagation of glitches to consecutive logic cells.

Circuits that heavily use of XIO functions (for example, Cyclic redundancy check) tend to glitch significantly when cascaded. Add pipeline registers or re-architect to reduce signal toggling.

Example 4. Glitch Prone Design



Related Information

Pipelining and Retiming on page 47

2.6. Power Optimization Revision History

The following revision history applies to this chapter:

Table 14. Document Revision History

Document Version	Quartus Prime Version	Changes
2022.12.12	22.4	Revised wording of *Clock Enable* topic for clarity.
2022.01.17	21.4	Added *Clock Gating* topics in the *Design Guidelines* section.
2021.10.04	21.3	Modified the *Compiler Settings* topic.
2020.12.07	18.1.0	Added note to the *Toggle Rate* topic.
2019.08.02	18.1.0	• Corrected typo in "Viewing Clock Details in the Chip Planner" topic. • Corrected typo in "Pipelining and Retiming" topic. • Corrected typo in "Implementation" topic. • Updated "DDR Memory Controller Settings" topic for latest IP name and to correct typos. • Corrected typo in "Pipeline Logic to Reduce Glitching" topic.
2018.09.24	18.1.0	• Added topic: *Factors Affecting Power Consumption*, moved from chapter: *Power Analysis* • Extended content about Power Optimization Advisor with a description of recommendations. • Added design guidelines: *Memories (M20K/MLAB)*, *DDR Memory Controller Settings*, *DSP Implementation*, *Reducing High-Speed Tile (HST) Usage*, *Unused Transceiver Channels*, *Periphery Power reduction XCVR Settings* • Removed content referring to device families not supported in *Intel Quartus Prime Pro Edition*.
2018.06.11	18.0.0	• Moved general information about the Design Space Explorer (DSE II) to the *Design Optimization User Guide*, left a section about using DSE II for Power-Driven Optimization.
2018.05.07	18.0.0	• Moved general information about the Design Space Explorer (DSE II) to the *Design Optimization User Guide*, left a section about using DSE II for Power-Driven Optimization.
2016.10.31	16.1.0	• Implemented Intel rebranding. • Removed statement of support for gate-level timing simulation.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*. • Updated screenshot for DSE II GUI. • Added information about remote hosts for DSE II.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations to Compiler Settings. • Updated DSE II GUI and optimization settings.
2014.06.30	14.0.0	Updated the format.
May 2013	13.0.0	Added a note to “Memory Power Reduction Example” on Qsys and SOPC Builder power savings limitation for on-chip memory block.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Template update.

continued...

Document Version	Quartus Prime Version	Changes
July 2010	10.0.0	• Was chapter 11 in the 9.1.0 release • Updated Figures 14-2, 14-3, 14-6, 14-18, 14-19, and 14-20 • Updated device support • Minor editorial updates
November 2009	9.1.0	• Updated Figure 11-1 and associated references • Updated device support • Minor editorial update
March 2009	9.0.0	• Was chapter 9 in the 8.1.0 release • Updated for the Quartus II software release • Added benchmark results • Removed several sections • Updated Figure 13–1, Figure 13–17, and Figure 13–18
November 2008	8.1.0	• Changed to 8½" × 11" page size • Changed references to altsyncram to RAM • Minor editorial updates
May 2008	8.0.0	• Added support for Stratix IV devices • Updated Table 9–1 and 9–9 • Updated "Architectural Optimization" on page 9–22 • Added "Dynamically-Controlled On-Chip Terminations" on page 9–26 • Updated "Referenced Documents" on page 9–29 • Updated references

3. Power Analysis and Optimization Document Archive

For the latest and previous versions of this user guide, refer to *Quartus Prime Pro Edition User Guide: Power Analysis and Optimization*. If a software version is not listed, the guide for the previous software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys* that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys*. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Design Constraints

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

683143 | 2024.04.01

Send Feedback

1. Constraining Designs

The design constraints, assignments, and logic options that you specify influence how the Quartus® Prime Compiler implements your design. The Compiler attempts to synthesize and place logic in a manner than meets your constraints. In addition, design constraints also have an impact on how the Timing Analyzer and the Power Analyzer influence synthesis, placement, and routing.

You can specify design constraints in the GUI, with scripts, or directly in the files that store the constraints. The Quartus Prime software preserves the constraints that you specify in the GUI in the following files:

- Quartus Prime Settings file (`<project_directory>/<revision_name>.qsf`)—contains project-wide and instance-level assignments for the current revision of the project, in Tcl syntax. Each revision of a project has a separate `.qsf` file.
- Synopsys* Design Constraints file (`<project_directory>/<revision_name>.sdc`)—the Timing Analyzer uses industry-standard Synopsys Design Constraint format and stores those constraints in `.sdc` files.

By combining the syntax of the `.qsf` files and the `.sdc` files with procedural Tcl, you can automate iterations over several different settings, changing constraints and recompiling.

What's New In This Version

- Applied initial Altera rebranding throughout.
- *Importing and Exporting I/O Pin Assignments* topic is updated for current list of file formats supported.

Related Information

- Quartus Prime Pro Edition Settings File Reference Manual
  For information about all settings and constraints in the Quartus Prime software.
- Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide
- Quartus Prime Pro Edition User Guide: Scripting

1.1. Specifying Design Constraints in the GUI

Quartus Prime software provides tools that help you manually implement your project. These tools can also support design visualization, pre-filled parameters, and window cross probing, facilitating design exploration and debugging.

When you create or update a constraint in the Quartus Prime software, the **System** tab of the **Messages** window displays the equivalent Tcl command. Utilize these commands as references for future scripted design definition and compilation.

1.1.1. Global Constraints and Assignments

Global constraints and project settings affect the entire Quartus Prime project and all the applicable logic in the design. You often define global constraints in early project development; for example, when running the New Project Wizard. Quartus Prime software stores global constraints in `.qsf` files, one for each project revision.

Table 1. Quartus Prime Tools to Set Global Constraints

Setting Type	New Project Wizard	Device Dialog Box	Settings Dialog Box
Project-wide	X	X	X
Synthesis	X	X	X
Fitter	X	X	X
Simulation			X
Third-party Tools			X
IP Settings			X

Related Information

Managing Project Settings
In *Quartus Prime Pro Edition User Guide: Getting Started*

1.1.2. Node, Entity, and Instance-Level Constraints

Node, entity, and instance-level constraints apply to a subset of the design hierarchy. These constraints take precedence over any global assignment that affects the same sections of the design hierarchy. The following tools are available in the Quartus Prime software to specify node, entity, and instance-level constraints:

Table 2. Quartus Prime Pro Edition Tools to Set Node, Entity and Instance Level Constraints

Assignment Type	Assignment Editor	Interface Planner	NoC Assignment Editor	Chip Planner	Pin Planner
Pin		X			X
Location	X	X		X	
Routing	X			X	
NoC			X		
Simulation	X			X	X

Although you can specify constraints using a variety of tools, the following table shows the most effective constraint tools at each design phase:

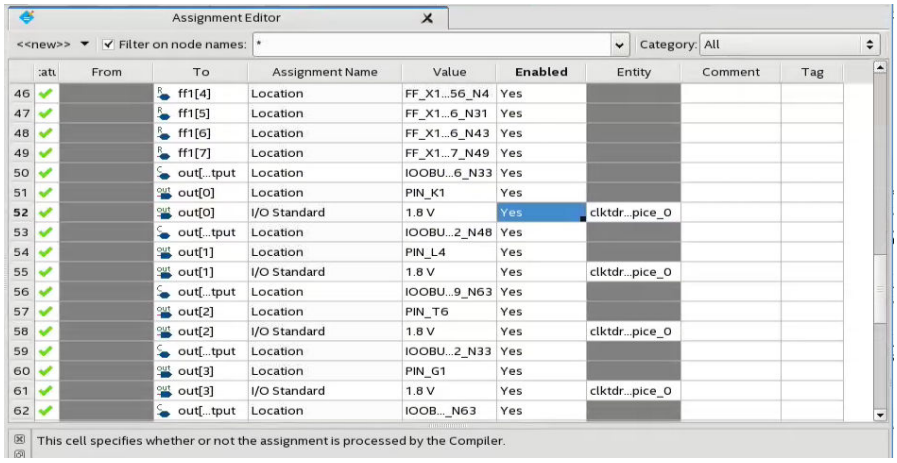
Table 3. Constraint Tools per Design Phase

Design Phase	Assignment Editor	NoC Assignment Editor	Interface Planner	Tile Interface Planner	Chip Planner	Timing Analyzer	Pin Planner
Pre-Synthesis	X	X		X			X
Post-Synthesis	X	X	X		X		
Post-Fit	X	X			X	X	

1.1.2.1. Specify Instance-Specific Constraints in Assignment Editor

Quartus Prime Assignment Editor (**Assignments ➤ Assignment Editor**) provides a spreadsheet-like interface for assigning all instance-specific settings and constraints. To help you explore your design, the Assignment Editor allows you to filter assignments by node name or category.

Figure 1. Quartus Prime Assignment Editor



Use the Assignment Editor to:

- Add, edit, or delete assignments for selected nodes
- Display information about specific assignments
- Enable or disable individual assignments
- Add comments to an assignment

Additionally, you can export assignments to a Comma-Separated Value File (`.csv`).

1.1.2.1.1. Specifying Multi-Dimensional Bus Constraints

The Quartus Prime Pro Edition software traditionally supports only 1- and 2-dimensional bus names for specifying constraints. The Quartus Prime Pro Edition version 19.3 and later now supports multi-dimensional bus names for more efficient constraints.

For example, you can specify the following assignment to apply a constraint to all bits in the `reg [31:0] r [0:2][4:5]` three-dimensional bus:

```
set_instance_assignment -name PRESERVE_REGISTER ON -to r
```

The constraint then applies to all bits `r: [0][4][31], r[0][4][30], … , r[1][5][0]`.

1.1.2.2. Specify NoC Constraints in NoC Assignment Editor

For designs targeting Agilex® 7 M-Series FPGAs only, the NoC Assignment Editor in the Quartus Prime Pro Edition software allows you to make logical assignments for hard memory NoC-related blocks in your design. These assignments include grouping, connectivity, address mapping, and bandwidth requirements.

The hard memory NoC facilitates high-bandwidth data movement between the FPGA core logic and memory resources, such as HBM2E and DDR5 memories. Refer to Interface Planner NoC Tool Flow on page 37 and the *Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide* for details on the complete NoC flow including Interface Planner.

The Quartus Prime software supports the following two flows for NoC design:

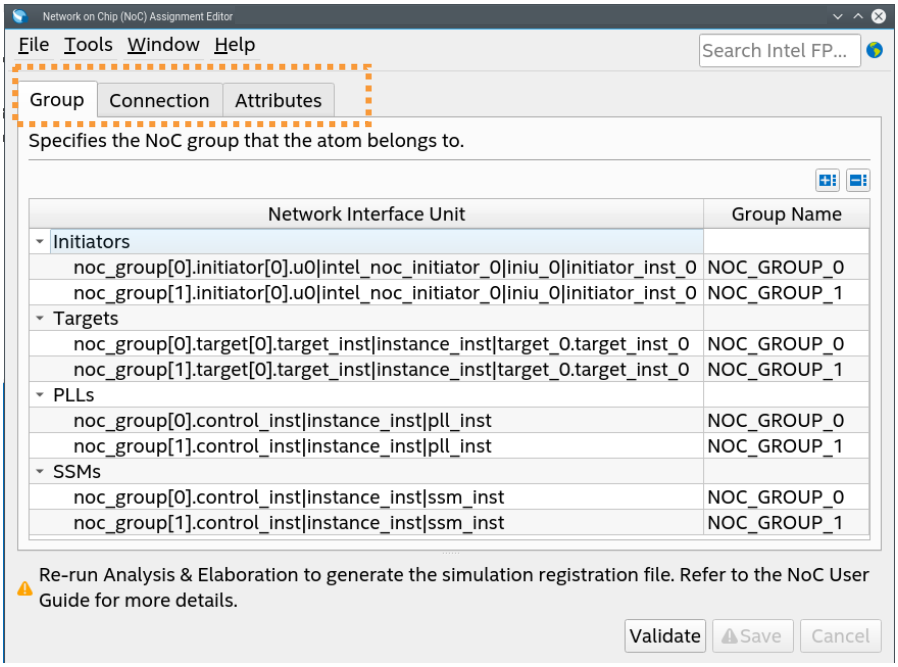
- **Platform Designer Connection Flow**—you use Platform Designer to configure and instantiate your NoC-related IP. You also use Platform Designer to make connections between NoC initiator bridges and NoC target bridges and to define the addressing mapping for these connections. Once you generate HDL for your Platform Designer system, your design is ready for RTL simulation. You must use the NoC Assignment Editor to create additional assignments, such as specifying NoC groupings and optional performance targets. You can use Interface Planner to make physical location assignments for your NoC elements. Then you compile your design and review the results.
- **NoC Assignment Editor Connection Flow**—you can configure and instantiate your NoC-related IP in either Platform Designer or directly in RTL. You then use the NoC Assignment Editor to make all NoC assignments including grouping, connectivity, address mapping, and optional performance targets. After completing the assignments and rerunning Analysis & Elaboration, your design is ready for RTL simulation. You can use Interface Planner to make physical location assignments for your NoC elements. Then compile your design and review the results.

Using the NoC Assignment Editor is similar to using the Assignment Editor, but the NoC Assignment Editor is optimized for making NoC assignments only. You must successfully complete Quartus Prime Analysis & Elaboration before using the NoC Assignment Editor. After Analysis & Elaboration, you can access the NoC Assignment Editor by clicking **Assignments ➤ Network on Chip (NoC) Assignment Editor**.

Specify assignments on the following NoC Assignment Editor tabs:

- **Group** tab—specifies the **Group Name** of the NoC initiators and targets.
- **Connection** tab—specifies the connections between NoC initiators and targets or SSM elements.
- **Attributes** tab—specifies address mapping, bandwidth requirements, and transaction sizes for each connection.

Figure 2. Network on Chip (NoC) Assignment Editor Group Tab



The assignments made on the **Group** tab affect the assignments available in the **Connection** tab. The assignments made on the **Connection** tab affect the assignments available in the **Attributes** tab.

Complete the assignments on each tab in order before moving to the next tab.

After making assignments in the NoC Assignment Editor, you click **Validate** to validate the assignments, and then click **Save** to store the assignments in the Quartus Prime settings file (`.qsf`).

Related Information

Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide

1.1.2.2.1. NoC Assignment Editor Interface Controls

For designs targeting Agilex 7 M-Series FPGAs only, you can use the following interface controls of the NoC Assignment Editor to specify NoC assignments:

Table 4. NoC Assignment Editor Interface Controls

NoC Assignment Editor GUI	Description
Group Tab	After completing Analysis & Elaboration and opening the NoC Assignment Editor, the **Group** tab displays two columns. The **Network Interface Unit** column displays a list of all NoC initiator, target, PLL, and SSM elements in your design. The **Group Name** column allows you to assign each of the elements to one of two NoC groups by entering the name of the group. You can define a custom, case-insensitive **Group Name**. Default names are `NOC_GROUP_0` and `NOC_GROUP_1`. Each group must contain the NoC initiators and targets

continued...

NoC Assignment Editor GUI	Description
	that you connect through that high-speed interconnect NoC, as well as one NoC PLL, and one NoC SSM. You must complete all assignments on the **Group** tab before proceeding to the **Connections** tab.
Connections Tab	You use the **Connections** tab to define connection assignments between NoC initiators and targets. Specify connections between a NoC initiator and a target or SSM by enabling the corresponding checkbox in the connection table. The group subtab includes a connection table with all the NoC initiators for that group listed on the left-hand side, and of all the NoC targets and SSM elements for the group listed across the top. You must complete all assignments on the **Connections** tab before proceeding to the **Attributes** tab.
Attributes Tab	You use the NoC Assignment Editor to create attribute assignments between initiators and targets. You specify assignments for address mapping and bandwidth requirements for each connection on the **Attributes** tab. The **Attributes** tab includes a subtab for each group that you specify on the **Group** tab. Each subtab lists each initiator to target connection. You must complete all assignments on the **Attributes** tab before you can **Save** the assignments.

1.1.2.2.2. Troubleshooting NoC Assignment Editor

Use the following FAQs to help you understand conditions and resolve conflicts in the NoC Assignment Editor:

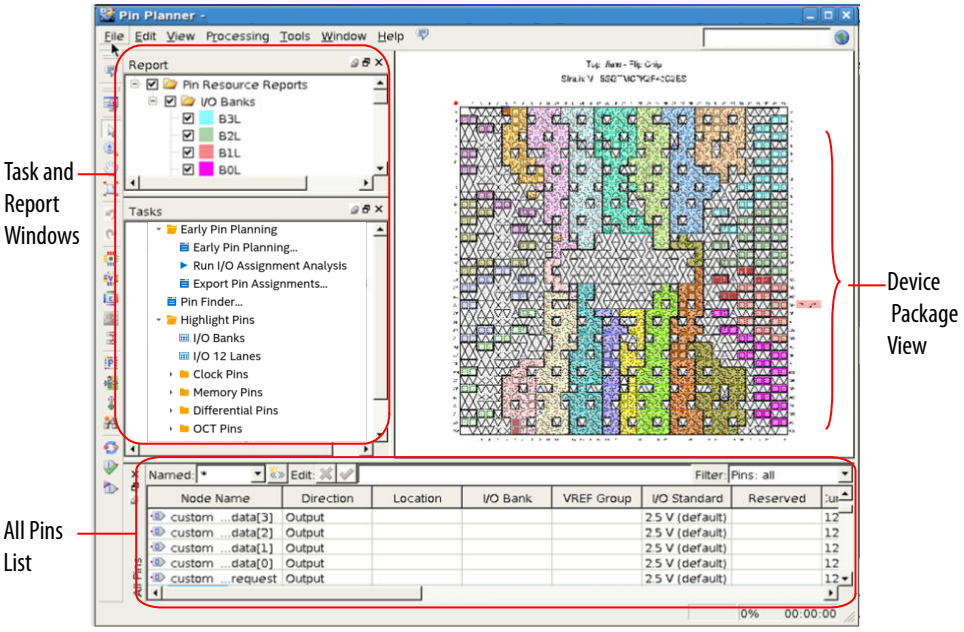
Table 5. NoC Assignment Editor FAQs

FAQs	Explanation/Resolution
Why are some assignments in NoC Assignment Editor 'read-only'?	• Any NoC assignment that you make in Platform Designer appears as read-only in NoC Assignment Editor. You cannot modify these read-only assignments inside the NoC Assignment Editor. • To modify such read-only NoC assignments, modify the corresponding elements in Platform Designer and regenerate HDL for the system.
Why are there conflicting assignments between Platform Designer and NoC Assignment Editor?	• This condition can occur if you first make assignments in NoC Assignment Editor, and then later make similar NoC assignments in Platform Designer. If you make Platform Designer assignments first, and then later make assignments in NoC Assignment Editor, those Platform Designer assignments appear as read-only in NoC Assignment Editor. • This condition can also occur if you first make NoC assignments in Platform Designer, and then later manually specify NoC assignments in the `.qsf` file. To avoid this condition, only use Platform Design or only use NoC Assignment Editor to specify NoC connectivity and addressing assignments. Avoid using the .qsf to specify NoC connectivity and addressing assignments.
How do I identify conflicting assignments between Platform Designer and NoC Assignment Editor?	• The NoC Assignment Editor cell containing any assignment conflict is highlighted in yellow. • The last assignment in the list takes precedence and is displayed. If the last assignment is a Platform Designer assignment, the cell is also read-only.
How can I resolve conflicting assignments between Platform Designer and NoC Assignment Editor?	You can use either of the following methods to replace a conflicting `.qsf` assignment with the Platform Designer assignment: • Right-click in the highlighted cell with a conflict, and then click **Delete**. • Manually delete any conflicting assignment from the `.qsf`. Avoid manually editing the `.qsf` if you can resolve issues in the NoC Assignment Editor.

1.1.2.3. Specify I/O Constraints in Pin Planner

Quartus Prime Pin Planner allows you to assign design elements to I/O pins. You can also plan and assign IP interface or user nodes not yet defined in the design.

Figure 3. Quartus Prime Pin Planner GUI



Related Information

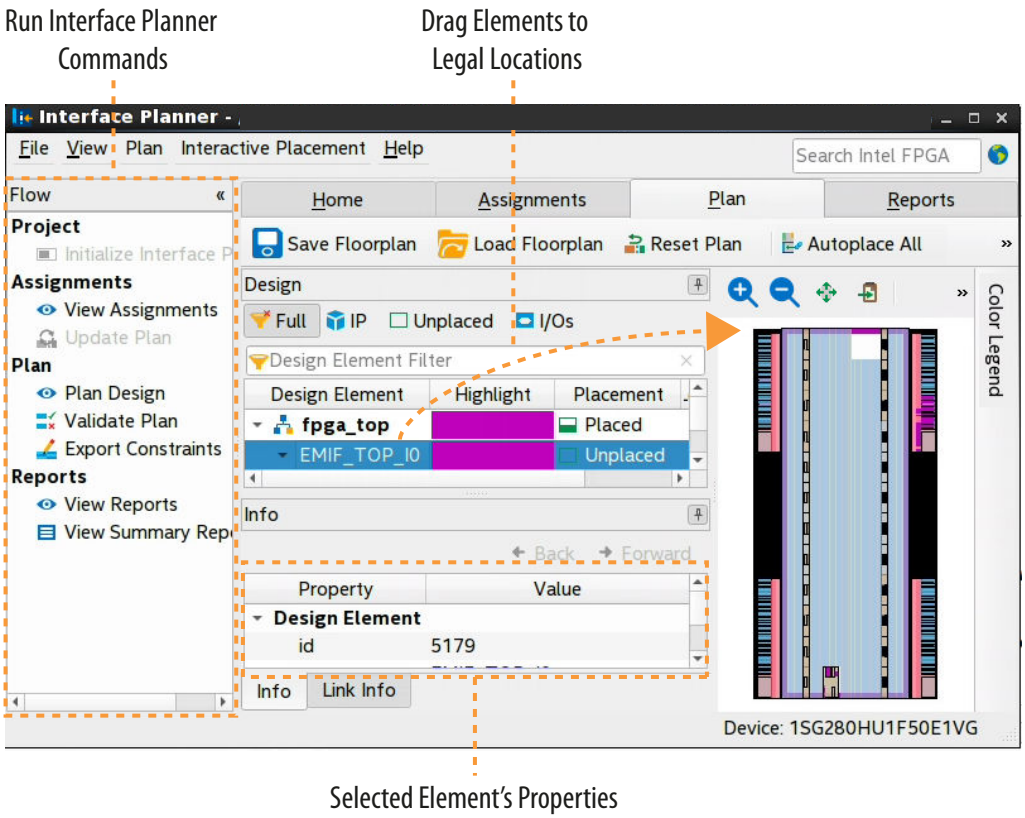
Managing Device I/O Pins on page 75

1.1.2.4. Plan Interface Constraints in Interface Planner and Tile Interface Planner

The Interface Planner simplifies the planning of accurate constraints for physical implementation. Similarly, you can use the Tile Interface Planner to build a plan for placement of IP components in each tile available on Agilex 7 F-tile devices. Use Interface Planner to prototype interface implementations, plan clocks, and rapidly define a legal device floorplan.

Interface Planner and Tile Interface Planner interact dynamically with the Quartus Prime Fitter to accurately verify placement legality while you plan. You can evaluate different floorplans, using interactive reports to accurately plan the best implementation without iterative compilation. Fitter verification ensures the highest correlation between your interface plan and actual implementation results. You can apply the interface plan constraints to your project with high confidence in the final implementation.

Figure 4. Interface Planner GUI



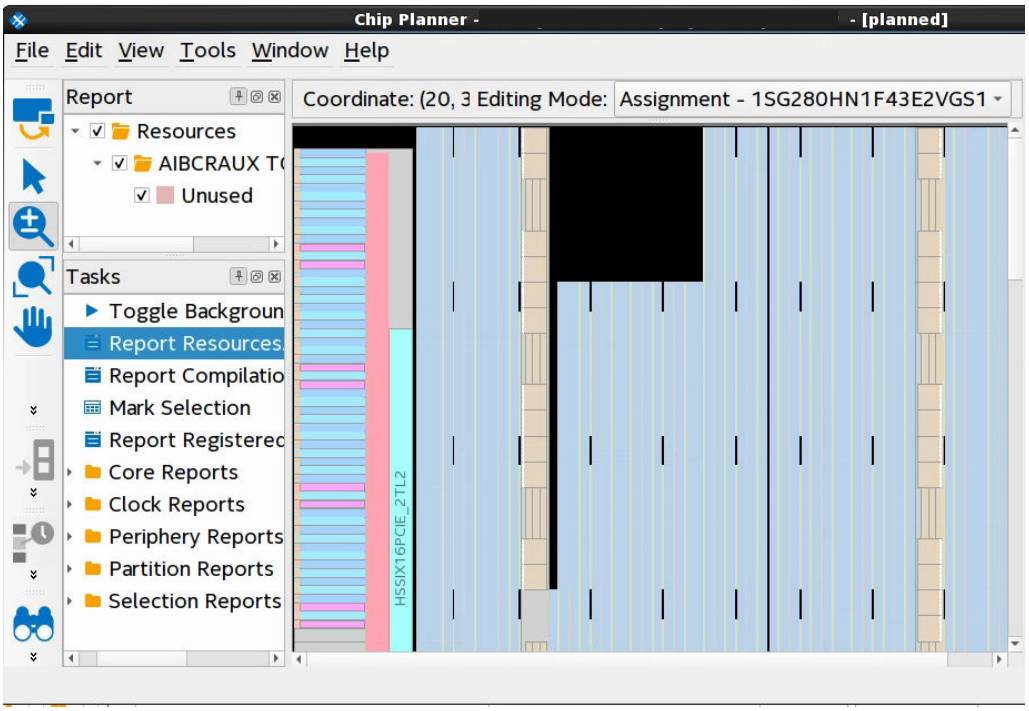
Related Information

- Using Interface Planner on page 23
- Using Tile Interface Planner on page 53

1.1.2.5. Adjust Constraints with the Chip Planner

With the Chip Planner you can adjust existing assignments to device resources, such as pins, logic cells, and LABs in a graphical representation of the device floorplan. You can also view equations and routing information and demote assignments by dragging and dropping to Logic Lock regions in the **Logic Lock Regions Window**.

Figure 5. Chip Planner GUI



Related Information

- Design Floorplan Analysis in the Chip Planner
  In *Quartus Prime Pro Edition User Guide: Design Optimization*
- Defining Logic Lock Placement Constraints, Quartus Prime Pro Edition User Guide: Design Optimization

1.1.2.6. Constraining Designs with the Design Partition Planner

The Design Partition Planner allows you to view design connectivity and hierarchy and can assist you in creating effective design partitions.

Additionally, the Design Partition Planner allows you to optimize design performance by isolating and resolving failing paths on a partition-by-partition basis.

Related Information

Creating Partitions and Logic Lock Regions with the Design Partition Planner and the Chip Planner
In *Quartus Prime Pro Edition User Guide: Design Optimization*

1.1.3. Probing Between Components of the Quartus Prime GUI

The Quartus Prime software allows you to locate nodes and instances within the compilation database from any of the following:

- Project Navigator
- Assignment Editor
- Chip Planner

- Timing Analyzer
- Resource Property Viewer
- RTL Viewer
- Technology Map Viewer
- Fast Forward Viewer
- Design Partition Planner
- Pin Planner
- HDL design files

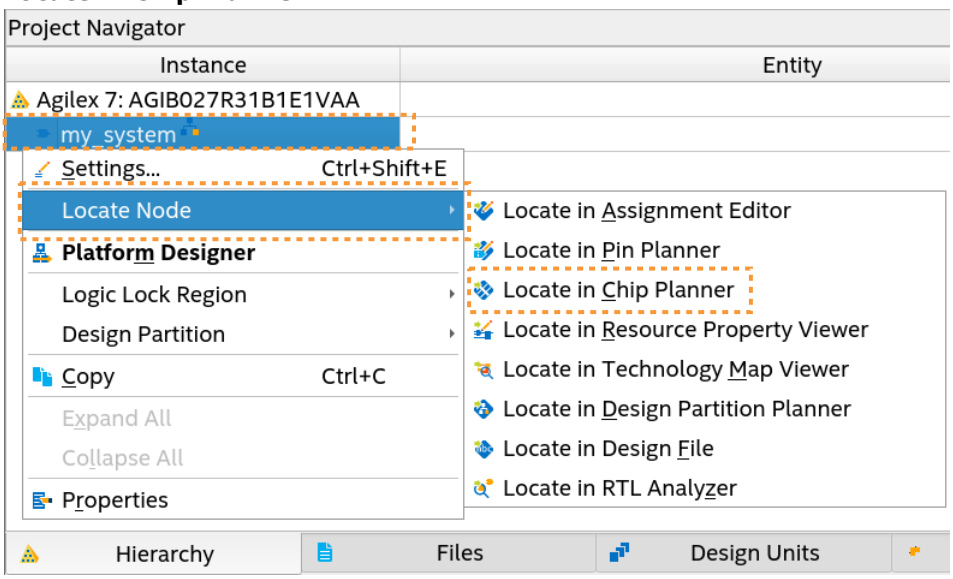
To locate nodes or instances, follow these steps:

1. Right-click the resource you want to display.
2. Click **Locate Node**, and then click any of the available menu options.

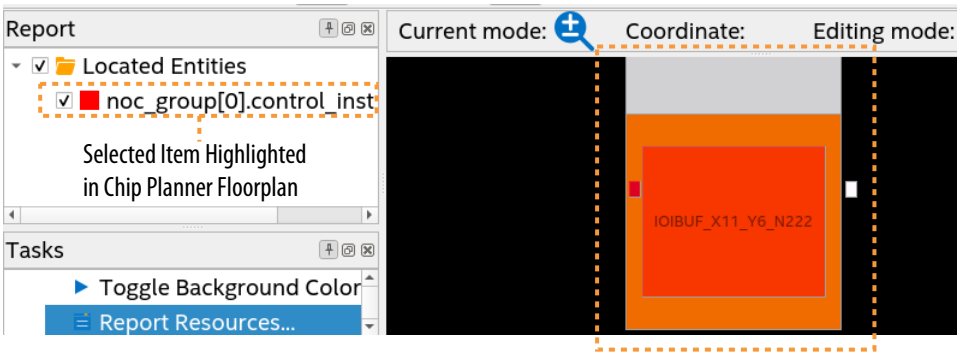
The corresponding window opens—or appears in the foreground if it is already open—and shows the element you clicked.

Example 1. Locate a Resource Selected in the Project Navigator

In the **Entity** list of the **Hierarchy** tab, right-click any object, and click **Locate ➤ Locate in Chip Planner**.



The Chip Planner opens and displays the instance you selected.



1.1.4. Specifying Timing Constraints

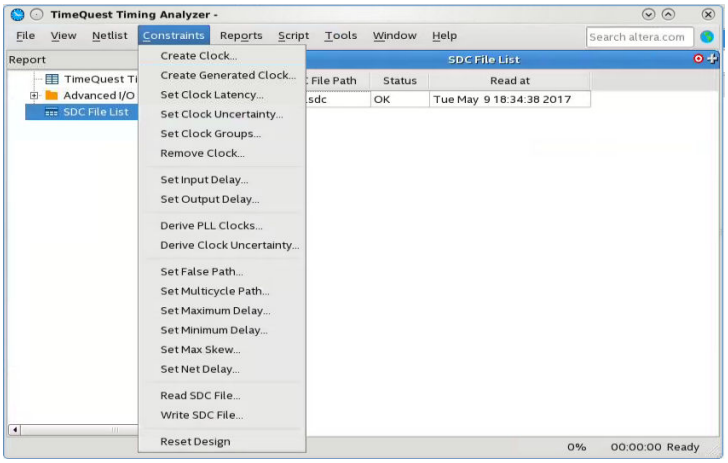
You must specify timing constraints that describe the clock frequency requirements, timing exceptions, and I/O timing requirements of your design for comparison against actual conditions observed during timing analysis. You define timing constraints in one or more Synopsys* Design Constraints (`.sdc`) files that you add to the project.

You can specify timing constraints in the Timing Analyzer GUI, which automatically generates an `.sdc` based on your inputs. Click the **Constraints** menu in the Timing Analyzer to specify timing constraints that you can apply to your project.

Alternatively, you can create an initial `.sdc` with provided `.sdc` file templates, or manually in any text editor and then add the `.sdc` to the project.

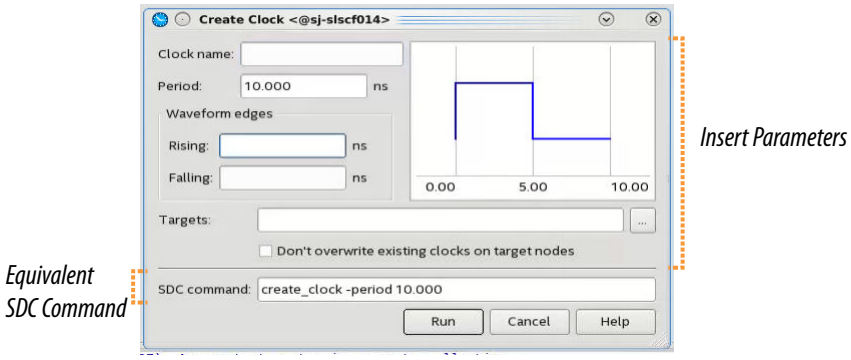
In addition, generation of Intel FPGA IP or Platform Designer systems may also automatically generate and add to your project `.sdc` constraints.

Figure 6. Constraint menu in Timing Analyzer



When you specify a constraint in the GUI, the dialog box displays the equivalent SDC command syntax.

Example 2. Create Clock Dialog Box



Individual timing assignments override project-wide requirements.

- To avoid reporting incorrect or irrelevant timing violations, you can assign timing exceptions to nodes and paths.
- The Timing Analyzer supports point-to-point timing constraints, wildcards to identify specific nodes when making constraints, and assignment groups to make individual constraints to groups of nodes.

Refer to the following Related Information:

Related Information

Applying Timing Constraints, Quartus Prime Pro Edition User Guide: Timing Analyzer

1.2. Constraining Designs with Tcl Scripts

You can perform all your design assignments using `.sdc` and `.qsf` setting files. To integrate these files in compilation and optimization flows, use Tcl scripts. Even though `.sdc` and `.qsf` files are written in Tcl syntax, they are not executable by themselves.

When you use Quartus Prime Tcl packages, your scripts can open projects, make the assignments, compile the design, and compare compilation results against known goals and benchmarks. Furthermore, such a script can automate the iterative design process by modifying constraints and recompiling the design.

1.2.1. Create a Project and Apply Constraints

The command-line executables include options for common global project settings and commands. You can use a Tcl script to apply constraints such as pin locations and timing assignments. You can write a Tcl constraint file, or generate one for an existing project by clicking **Project ➤ Generate Tcl File for Project**.

The example creates a project with a Tcl script and applies project constraints using the tutorial design files in the <Quartus Prime *installation directory*>/qdesigns/fir_filter/ directory.

```
project_new filtref -overwrite
# Assign family, device, and top-level file
set_global_assignment -name FAMILY "Arria 10"
set_global_assignment -name DEVICE <Device>
set_global_assignment -name VERILOG_FILE filtref.v
# Assign pins
set_location_assignment -to clk Pin_28
set_location_assignment -to clkx2 Pin_29
set_location_assignment -to d[0] Pin_139
set_location_assignment -to d[1] Pin_140
#
project_close
```

Save the script in a file called `setup_proj.tcl` and type the commands illustrated in the example at a command prompt to create the design, apply constraints, compile the design, and perform fast-corner and slow-corner timing analysis. Timing analysis results are saved in two files, `filtref_sta_1.rpt` and `filtref_sta_2.rpt`.

```
quartus_sh -t setup_proj.tcl
quartus_syn filtref
quartus_fit filtref
quartus_asm filtref
quartus_sta filtref --model=fast --export_settings=off
mv filtref_sta.rpt filtref_sta_1.rpt
quartus_sta filtref --export_settings=off
mv filtref_sta.rpt filtref_sta_2.rpt
```

Type the following commands to create the design, apply constraints, and compile the design, without performing timing analysis:

```
quartus_sh -t setup_proj.tcl
quartus_sh --flow compile filtref
```

The `quartus_sh --flow compile` command performs a full compilation, and is equivalent to clicking the **Start Compilation** button in the toolbar.

1.2.2. Assigning a Pin

To assign a signal to a pin or device location, use the Tcl command shown in this example:

```
set_location_assignment -to <signal name> <location>
```

Valid locations are pin location names. Some device families also support edge and I/O bank locations. Edge locations are `EDGE_BOTTOM`, `EDGE_LEFT`, `EDGE_TOP`, and `EDGE_RIGHT`. I/O bank locations include `IOBANK_1` to `IOBANK_n`, where n is the number of I/O banks in a device.

1.2.3. Generating Quartus Prime Settings Files

Quartus Prime software allows you to generate `.qsf` files from your revision. You can embed these constraints in a scripted compilation flow, and even create sets of `.qsf` files for design optimization.

To generate a `.qsf` file from the Quartus Prime software, click **Assignments ➤ Export Assignments**.

To organize the `.qsf` in a human readable form, **Project ➤ Organize Quartus Prime Settings File**.

Example 3. Organized .qsf File

This example shows how `.qsf` files characterize a design revision. The `set_global_assignment` command makes all global constraints and software settings and `set_location_assignment` constrains each I/O node in the design to a physical pin on the device.

```
# Project-Wide Assignments
# ========================
set_global_assignment -name SYSTEMVERILOG_FILE top.sv
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led.sv
set_global_assignment -name SDC_FILE blinking_led.sdc
set_global_assignment -name SDC_FILE jtag.sdc
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
set_global_assignment -name LAST_QUARTUS_VERSION "17.1.0 Pro Edition"
set_global_assignment -name TEXT_FILE blinking_led_generated.txt
# Pin & Location Assignments
# ==========================
set_location_assignment PIN_AN18 -to clock
set_location_assignment PIN_AR23 -to led_zero_on
set_location_assignment PIN_AM21 -to led_two_on
set_location_assignment PIN_AR22 -to led_one_on
set_location_assignment PIN_AL20 -to led_three_on
# Analysis & Synthesis Assignments
# ================================
set_global_assignment -name FAMILY "Arria 10"
set_global_assignment -name TOP_LEVEL_ENTITY top
# Fitter Assignments
# ==================
set_global_assignment -name DEVICE 10AS066N3F40E2SG
# -----------------
# start ENTITY(top)
# Fitter Assignments
# ==================
set_instance_assignment -name IO_STANDARD "1.8 V" -to led_zero_on
set_instance_assignment -name IO_STANDARD "1.8 V" -to led_one_on
set_instance_assignment -name IO_STANDARD "1.8 V" -to led_two_on
set_instance_assignment -name IO_STANDARD "1.8 V" -to led_three_on
set_instance_assignment -name SLEW_RATE 1 -to led_zero_on
set_instance_assignment -name SLEW_RATE 1 -to led_one_on
set_instance_assignment -name SLEW_RATE 1 -to led_two_on
set_instance_assignment -name SLEW_RATE 1 -to led_three_on
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to clock
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to led_zero_on
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to led_one_on
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to led_two_on
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to led_three_on
# end ENTITY(top)
# ---------------
```

Related Information

- Quartus Prime Pro Edition Settings File Reference Manual
  For information about all settings and constraints in the Quartus Prime software.
- Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide

1.2.4. Synopsys Design Constraint (.sdc) Files

Quartus Prime software keeps timing constraints in `.sdc` files, which use Tcl syntax. You can embed these constraints in a scripted compilation flow, and even create sets of `.sdc` files for timing optimization.

Example 4. .sdc File

The example shows the timing constrains of a small design.

```
## PROGRAM "Quartus Prime"
## VERSION "Version 17.1.0 Internal Build 91 05/07/2017 SJ Pro Edition"
## DATE    "Wed May 10 14:22:08 2017"
##
## DEVICE  "10AX115R4F40I3SG"
##
#**************************************************************
# Time Information
#**************************************************************
set_time_format -unit ns -decimal_places 3
#**************************************************************
# Create Clock
#**************************************************************
create_clock -name {clk_in} -period 10.000 -waveform { 0.000 5.000 } [get_ports
{clk_in}]
#**************************************************************
# Create Generated Clock
#**************************************************************
derive_pll_clocks
#**************************************************************
# Set Clock Uncertainty
#**************************************************************
derive_clock_uncertainty
#**************************************************************
# Set Input Delay
#**************************************************************
set_input_delay -add_delay  -clock [get_clocks {clk_in}]  1.500 [get_ports
{async_rst}]
set_input_delay -add_delay  -clock [get_clocks {clk_in}]  1.200 [get_ports
{data_in}]
#**************************************************************
# Set Output Delay
#**************************************************************
set_output_delay -add_delay  -clock [get_clocks {clk_in}]  2.000 [get_ports
{data_out}]
#**************************************************************
# Set Multicycle Path
#**************************************************************
set_multicycle_path -setup -end -from [get_keepers *] -to [get_keepers {reg2}] 2
```

1.2.5. Tcl-only Script Flows

As an alternative to `.sdc` and `.qsf` files, you can perform all design assignments and timing constraints inside the Tcl scripts. In this case, the script that automates compilation and custom results reporting also contains the design constraints.

You can export a design's contents to a procedural, executable Tcl (`.tcl`) file, and then use the generated script to restore settings after experimenting with other constraints.

To export your constraints as an executable Tcl script, click **Project ➤ Generate Tcl File for Project**.

Example 5. blinking_led_generated.tcl File

```
# Quartus Prime: Generate Tcl File for Project
# File: blinking_led_generated.tcl
# Generated on: Wed May 10 10:14:44 2017
# Load Quartus Prime Tcl Project package

package require ::quartus::project
set need_to_close_project 0
set make_assignments 1
# Check that the right project is open
if {[is_project_open]} {
  if {[string compare $quartus(project) "blinking_led"]} {
puts "Project blinking_led is not open"
set make_assignments 0
  }
} else {
  # Only open if not already open
  if {[project_exists blinking_led]} {
project_open -revision blinking_led blinking_led
  } else {
project_new -revision blinking_led blinking_led
  }
  set need_to_close_project 1
}

# Make assignments
if {$make_assignments} {
set_global_assignment -name SYSTEMVERILOG_FILE top.sv
set_global_assignment -name SYSTEMVERILOG_FILE blinking_led.sv
set_global_assignment -name SDC_FILE blinking_led.sdc
set_global_assignment -name SDC_FILE jtag.sdc
set_global_assignment -name PROJECT_OUTPUT_DIRECTORY output_files
set_global_assignment -name LAST_QUARTUS_VERSION "17.1.0 Pro Edition"
set_global_assignment -name TEXT_FILE blinking_led_generated.txt
set_global_assignment -name FAMILY "Arria 10"
set_global_assignment -name TOP_LEVEL_ENTITY top
set_global_assignment -name DEVICE 10AS066N3F40E2SG
set_location_assignment PIN_AN18 -to clock
set_location_assignment PIN_AR23 -to led_zero_on
set_location_assignment PIN_AM21 -to led_two_on
set_location_assignment PIN_AR22 -to led_one_on
set_location_assignment PIN_AL20 -to led_three_on
set_instance_assignment -name IO_STANDARD "1.8 V" -to led_zero_on
set_instance_assignment -name IO_STANDARD "1.8 V" -to led_one_on
set_instance_assignment -name IO_STANDARD "1.8 V" -to led_two_on
set_instance_assignment -name IO_STANDARD "1.8 V" -to led_three_on
set_instance_assignment -name SLEW_RATE 1 -to led_zero_on
set_instance_assignment -name SLEW_RATE 1 -to led_one_on
set_instance_assignment -name SLEW_RATE 1 -to led_two_on
set_instance_assignment -name SLEW_RATE 1 -to led_three_on
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to clock
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to led_zero_on
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to led_one_on
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to led_two_on
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to led_three_on
# Commit assignments
export_assignments
# Close project
if {$need_to_close_project} {
project_close
}
}
```

The example:

- Opens the project
- Assigns Constraints
- Writes assignments to QSF file
- Closes project

1.3. A Fully Iterative Scripted Flow

The **::quartus::flow** Tcl package in the Quartus Prime Tcl API allows you to modify design constraints and recompile in an iterative flow.

Related Information

Quartus Prime Pro Edition User Guide: Scripting

1.4. Constraining Designs Revision History

Table 6. NoC Assignment Editor Interface Controls

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2023.12.04	23.4	• Applied initial Altera rebranding throughout. • Added new *Troubleshooting NoC Assignment Editor* topic. • Revised the *Specifying Timing Constraints* topic to add introductory information and context about typical constraints. • Updated Pin Planner screenshot in *Specify I/O Constraints in Pin Planner* topic for removal of VREF Groups and Edges highlight.
2023.10.02	23.3	• Updated *What's New In This Version* for enhancements to Pin Planner reporting. • Updated screenshots in *Probing Between Components of the Quartus Prime GUI* for latest Compilation Dashboard. • Revised *Specify NoC Constraints in NoC Assignment Editor* topic for new connection flows.
2023.04.03	23.1	• Updated *What's New In This Version* for NoC support and Intel Agilex 7 device family name changes. • Updated *Node, Entity, and Instance-Level Constraints* topic for NoC Assignment Editor. • Updated *Plan Tab Controls* topic for NoC View. • Updated *Reports Tab Controls* topic for NoC Performance report. • Added new *Specify NoC Constraints in the NoC Assignment Editor* topic. • Added new *NoC Assignment Editor Interface Controls* topic. • Updated product family name to "Intel Agilex 7."
2022.06.21	22.2	• Added Top FAQs navigation to document cover. • Added *What's New In This Version* section to *Constraining Designs* topic.
2021.10.04	21.3	• Removed obsolete *Tcl-only Timing Analysis* topic. • Updated *Node, Entity, and Instance-Level Constraints* topic for latest tools and *Constraint Tools per Design Phase* table. • Revised *Plan Interface Constraints* topic for Tile Interface Planner. • Revised *Probing Between Components of the Quartus Prime GUI* for latest tools.

continued...

Document Version	Quartus Prime Version	Changes
2019.10.16	19.3	• Added "Specifying Multi-Dimensional Bus Constraints" topic. • Updated examples in "Create a Project and Apply Constraints."
2019.08.21	18.1	Corrected minor typo in "Tcl-only Script Flows" topic.
2019.01.04	18.1	• Clarified default location of .sdc and .qsf files in "Constraining Designs" topic. • Added "Plan Interface Constraints with Interface Planner" topic. • Added screenshots to "Constrain Designs with the Pin Planner" and "Constrain Designs with the Chip Planner." • Added two new "Assigning a Pin" and "Creating a Project and Applying Constraints" topics showing Tcl examples. • Added link to Using Timing Constraints topic in Timing Analyzer UG that explains all of the commands
2017.11.06	17.1	• Renamed topic: Constraining Designs with the GUI to Constraining Designs with Quartus Prime Tools. • Renamed topic: Global Constraints to Global Constraints and Assignments. • Added table: Quartus Prime Tools to Set Global Constraints. • Removed topic: Common Types of Global Constraints. • Removed topic: Settings That Direct Compilation and Analysis Flows. • Updated topic: Node, Entity and Instance-Level Constraints. • Added table: Quartus Prime Tools to Set Node, Entity and Instance Level Constraints. • Added topic: Assignment Editor. • Updated topic: Constraining Designs with the Pin Planner. • Updated topic: Constraining Designs with the Chip Planner. • Added topic: Constraining designs with the Design Partition Planner. • Updated topic: Probing Between Components of the Quartus Prime GUI. • Added example: Locate a Resource Selected in the Project Navigator. • Updated topic: SDC and the Timing Analyzer, and renamed to Specifying Individual Timing Constraints. • Added figure: Constraint Menu in Timing Analyzer. • Added example: Create Clock Dialog Box. • Updated topic: Constraining Designs with Tcl, and renamed to Constraining Designs with Tcl Scripts • Updated topic: Quartus Prime Settings Files and Tcl , and renamed to Generating Quartus Prime Settings Files. • Added example: blinking_led.qsf File. • Updated topic: Timing Analysis with Synopsys Design Constraints and Tcl, and renamed to Timing Analysis with .sdc Files and Tcl Scripts. • Added example: .sdc File with Timing Constraints. • Added topic: Tcl-only Script Flows. • Updated topic: A Fully Iterative Scripted Flow.
2017.05.08	17.0	• Removed references to deprecated Fitter Effort logic option.
2016.10.31	16.1	• Implemented Intel rebranding.
2015.11.02	15.1	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0	Formatting updates.
November 2012	12.1	Update Pin Planner description for task and report windows.
June 2012	12.0	Removed survey link.
November 2011	10.0	Template update.

continued...

Document Version	Quartus Prime Version	Changes
December 2010	10.0	Template update.
July 2010	10.0	Rewrote chapter to more broadly cover all design constraint methods. Removed procedural steps and user interface details, and replaced with links to Quartus II Help.
November 2009	9.1	• Added two notes. • Minor text edits.
March 2009	9.0	• Revised and reorganized the entire chapter. • Added section "Probing to Source Design Files and Other Quartus Windows" on page1–2. • Added description of node type icons (Table1–3). • Added explanation of wildcard characters.
November 2008	8.1	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0	Updated Quartus II software 8.0 revision and date.

2. Interface Planning

Interface planning—the feasibility analysis of interface physical constraints—is a fundamental early step in advanced FPGA design. Periphery placement can be a complex process involving many variables. The Quartus Prime Interface Planner simplifies the planning of accurate constraints for physical implementation.

You can use the Interface Planner to prototype interface implementations, plan clocks, and rapidly define a legal device floorplan.

Similarly, when targeting the Agilex 7 F-tile devices, you can use the Tile Interface Planner build a plan for placement of IP components in each tile available on the FPGA device.

Interface Planner and Tile Interface Planner (launched from the Tools menu) interact dynamically with the Quartus Prime Fitter to accurately verify placement legality while placing elements. You can evaluate different floorplans, using interactive reports to accurately plan the best implementation without iterative compilation. Fitter verification ensures the highest correlation between your interface plan or tile interface plan and actual implementation results. You can apply the interface plan or tile interface plan constraints to your project with high confidence in the final implementation.

Related Information

- Video Demo: Using Interface Planner to Place DDR-3 and PCI Express Gen3
- Video Demo: Using the Tile Interface Planner

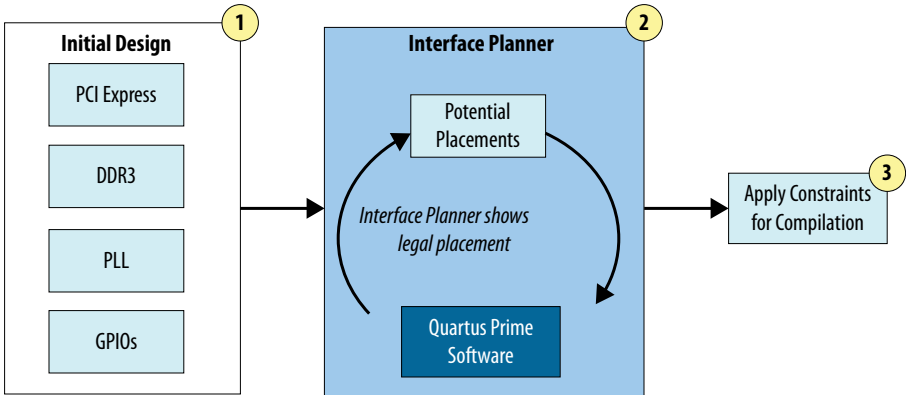
2.1. Using Interface Planner

After design synthesis, you can use Interface Planner to help you to rapidly define a legal device floorplan.

Interface Planner displays your project's logical hierarchy, post-synthesis design elements, and Fitter-created design elements, alongside a view of target device locations. The GUI supports a variety of methods for placing design elements in the floorplan. As you place elements in the floorplan, the Fitter verifies legality in real time to ensure accurate correlation with the final implementation.

Intel® FPGAs contain core and periphery device locations. The device core locations are adaptive look-up tables (ALUTs), core flip-flops, RAMs, and digital signal processors (DSPs). Device periphery locations include I/O elements, phase-locked loops (PLLs), clock buffers, and hard processor systems (HPS).

Figure 7. Interface Planner Streamlines Legal Placement



Intel FPGAs contain many silicon features in the device periphery, such as hard PCI Express® IP cores, high speed transceivers, hard memory interface circuitry, and embedded processors. Interactions among these periphery elements can be complex. Interface Planner simplifies this complexity and allows you to quickly visualize and place I/O interface and periphery elements, such as:

- I/O elements
- LVDS interfaces
- PLLs
- Clocks
- Hard interface IP Cores
- High-Speed Transceivers
- Hard Memory Interface IP Cores
- The Hard Memory Network-on-Chip (NoC)(1)
- Embedded Processors

2.1.1. Interface Planner User Interface

The Interface Planner user interface includes the following controls for planning your design platform.

Flow Controls on page 25
Home Tab Controls on page 25
Assignments Tab Controls on page 25
Plan Tab Controls on page 26
Reports Tab Controls on page 27

(1) For designs targeting Agilex 7 M-Series FPGAs only.

2.1.1.1. Flow Controls

The **Flow** control panel provides immediate access to common Interface Planner commands from anywhere within Interface Planner. The **Flow** controls appear in order of a typical interface planning flow.

Table 7. Flow Controls

Command	Description
Open Project	Allows you to select and open an Quartus Prime project in Interface Planner. Use of **Open Project** command is only required when using Interface Planner in standalone mode.
Initialize Interface Planner	Loads the synthesis netlist, starts the Fitter verification engine, and imports assignments from your Quartus Prime project.
View Assignments	Opens the **Assignments** tab, which allows you to review and reconcile any conflicting assignments that Interface Planner imports from your project. Enable or disable specific project assignments to resolve any conflicts.
Update Plan	Applies the enabled project assignments to your interface plan. You cannot perform periphery planning on the **Plan** tab until you update the plan.
Plan Design	Opens the **Plan** tab for placing logic in the interface plan.
Validate plan	Verifies that all constraints in the interface plan are compatible with placement of all remaining unplaced design elements. You can then directly locate and resolve the source of any reported validation errors. You must successfully validate the plan before running **Write Plan File**.
Export Constraints	Saves your interface plan as a Tcl script file for subsequent application in your project. This command is available only after successfully running **Validate Plan**.
View Reports	Opens the **Reports** tab for filtering data and finding entities and locations.
View Summary Reports	Opens the **Interface Planner Summary** report that summarizes the percentage of placed and unplaced periphery cells.

2.1.1.2. Home Tab Controls

The Interface Planner **Home** tab contains controls for opening projects in Interface Planner. You only need the **Home** tab when Interface Planner is in standalone mode.

Table 8. Home Tab Controls

Command	Description
Recent Projects	Provides quick access to recently opened Quartus Prime projects. A named tile represents each project. Click the tile to display **Details** about the project. Double-click the tile to open the project in Interface Planner.
Browse	Allows you to locate and open an Quartus Prime project in Interface Planner. Interface Planner requires the project's synthesized netlist for operation.
Details	Provides project and file details such as the file path, revision, and creation date of the Quartus Prime project. You can select a specific project revision.

2.1.1.3. Assignments Tab Controls

The **Assignments** tab contains controls for resolving potential conflicts with project assignments. Click **View Assignments** to display the **Assignments** tab. You can enable or disable specific or classes of assignments until you resolve all potential conflicts. After you are satisfied with the status of all project assignments, click **Update Plan** to update your interface plan with the enabled project assignments. Interface Planner reports an error for any remaining assignment conflicts.

Table 9. Assignments Tab Controls

Command	Description
Filter field	Supports creation of wildcard expressions for assignment targets. **Enabled** and **Disabled** buttons filter only enabled or disabled assignments in the list.
Enable All Project Assignments	Enables all imported project assignments in your interface plan.
Disable All Project Assignments	Disables all imported project assignments in the plan.
Clear	Clears any filter from the **Assignments** list.

2.1.1.4. Plan Tab Controls

The **Plan** tab contains the following controls to help you locate and place logic in the interface plan. Click **Plan Design** to display the **Plan** tab. Placement or unplacement in the interface plan does not apply to your Quartus Prime project until you add the generated Interface Planner constraints script to your project.

Table 10. Plan Tab Controls

Command	Description
	Lists legal locations for placement.
Locate Node	Display a list of Quartus Prime Pro Edition tools where the selected design element is referenced in the hierarchical database. If the **Locate Node** command is disabled for a specific element in the **Design Elements** list, it is because that element is not represented as an element in the design.
Autoplace All	Attempts to place all unplaced design elements in legal locations in the interface plan.
Autoplace Fixed	Attempts to place all unplaced design elements that have only one legal location into the interface plan.
Unplace All	Unplaces all placed design elements in the interface plan.
Right-click ➤ Auto-place selected element	Attempts to place the selected design element and all its children in a legal location in the interface plan.
Chip View	Displays the target device chip. Zoom in to display chip details.
Package View	Displays the target device package. Zoom in to display chip details.
NoC View	Displays a filtered view of NoC initiators and targets. Refer to Interface Planner NoC Tool Flow
Show I/O Banks	Selects and color codes the I/O banks in the **Plan** tab.
Show Differential Pin Pair Connections	In **Package View**, displays a red connection line between a pair of differential pins. The Package View labels the positive and negative pins with the letters p and n, respectively.
Show PCIe Hard IP Interface Pins	In **Package View**, selects and color codes the PCIe Hard IP interface pins in the **Plan** tab. To access this command, right-click in the **Plan** tab package view, and select **x1 Lanes**, **x2 Lanes**, **x4 Lanes**, **x8 Lanes**, or **by 16 Lanes**. After enabling, view color coding in the **Color Legend**.
Show DQ/DQS Pins	In **Package View**, selects and color codes the PCIe Hard IP interface pins in the **Plan** tab. To access this command, right-click in the **Plan** tab package view, and select **x4 Mode**, **x8/x9 Mode**, **x16/x16 Mode**, or **x32/x36 Mode**. After enabling, view color coding in the **Color Legend**.
Right-click ➤ Report Placeability of Selected Element	After selecting a low level element, displays detailed information on the **Reports** tab, showing legal locations in the interface plan for the selected cell in order of suitability for fitting.

continued...

Command	Description
Right-click ➤ Report Legal Locations of Selected Element	After selecting a low level element, displays the legal locations in the interface plan for the selected cell in order of suitability for fitting.
	Copies the current interface plan to the clipboard for pasting into other files, such as word processing or presentation files.
Reset Plan	Unplaces all placed design elements and removes applied project assignments from the interface plan. Resets all project assignments to the enabled state. You must subsequently run **Update Plan** prior to placing design elements. This command only applies to your interface plan and does not impact your Quartus Prime project assignments until you apply the Interface Planner script.
Load Floorplan	Allows you to select and load an Interface Planner Floorplan Format (`.plan`) file. You can save Interface Planner floorplan files in the format by clicking **Save Floorplan**.
Save Floorplan	Allows you to save your Interface Planner floorplan as a `.plan` file.

2.1.1.5. Reports Tab Controls

The Interface Planner **Reports** tab contains the following **Task** pane controls to help you filter data and find entities and locations.

Table 11. Reports Tab Controls

Command	Description
Create all Summary Reports	Creates the following summary reports: • **Interface Planner Summary** • **All Periphery Cells** • **Placed/Unplaced Periphery Cells** • **Periphery Location Types**.
Report All Placed/Unplaced Pins	Reports the name, parent (if any), and type of all placed (**Report All Placed Pins**) or unplaced (**Report All Unplaced Pins**) pins in the interface plan, respectively. The Placed Pins report includes the placement location name. The Unplaced Pins report includes the number of potential placement locations. Right-click any cell to place, unplace, or report connectivity or location information.
Report All Placed/Unplaced HSSI Channels	Reports the name, parent (if any), and type of all placed (**Report All Placed HSSI Channels**) or unplaced (**Report All Unplaced HSSI Channels**) channels in the interface plan, respectively. The Placed HSSI Channels report includes the placement location name. The Unplaced HSSI Channels report includes the number of potential placement locations. Right-click any cell to place, unplace, or report connectivity or location information.
Right-click ➤ Report Placed/ Unplaced Periphery Cells of Selected Type	Reports the name, parent (if any), and type of placed (**Report Placed Periphery Cells of Selected Type**) or unplaced (**Report Unplaced Periphery Cells of Selected Type**) cells matching the selected type. The placed cells report includes the placement location name. The unplaced cells report includes the number of potential placement locations. Right-click any cell to place, unplace, or report connectivity or location information.
Right-click ➤ Report Periphery Locations of Selected Type	Reports all locations in the device of the selected type, and whether the location supports merging.
Right-click ➤ Report Periphery Cell Connectivity	Reports the source port and type, destination port and type, of connections to the selected cell. Right-click any cell to report the individual cell connectivity.
Right-click ➤ Place/Unplace Cell	Places the cell in the selected location of the interface plan. Similarly, you can right-click any cell and then click **Place Cell of Selected Type** or **Unplace Cell of Selected Type** to place or unplace multiple cells of the same type.

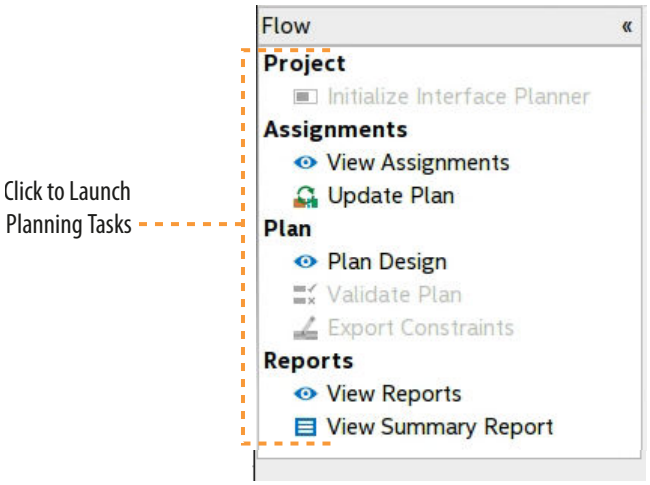
continued...

Command	Description
Right-click ➤ Report Cell Locations for Custom Placement	Reports the preferred legal locations for the selected cell in the interface plan in the Legal Location report. Right-click to immediately place the cell in a location or report all periphery location of the selected type.
Remove Invalid Reports	Removes outdated Interface Planner reports that you invalidate by changes to the interface plan.
Report Instance Assignments	Shows all imported project assignments in the interface plan. You can delete these assignments from the plan.
Report NoC Performance	Performs a static analysis of the NoC initiator and target locations to evaluate whether the placement allows your design to meet the bandwidth requirements and transaction sizes that you specify in the NoC Assignment Editor. You can review the report, and then make changes in the **Plan** tab based on the results. Refer to Report NoC Performance.

2.1.2. Interface Planner General Tool Flow

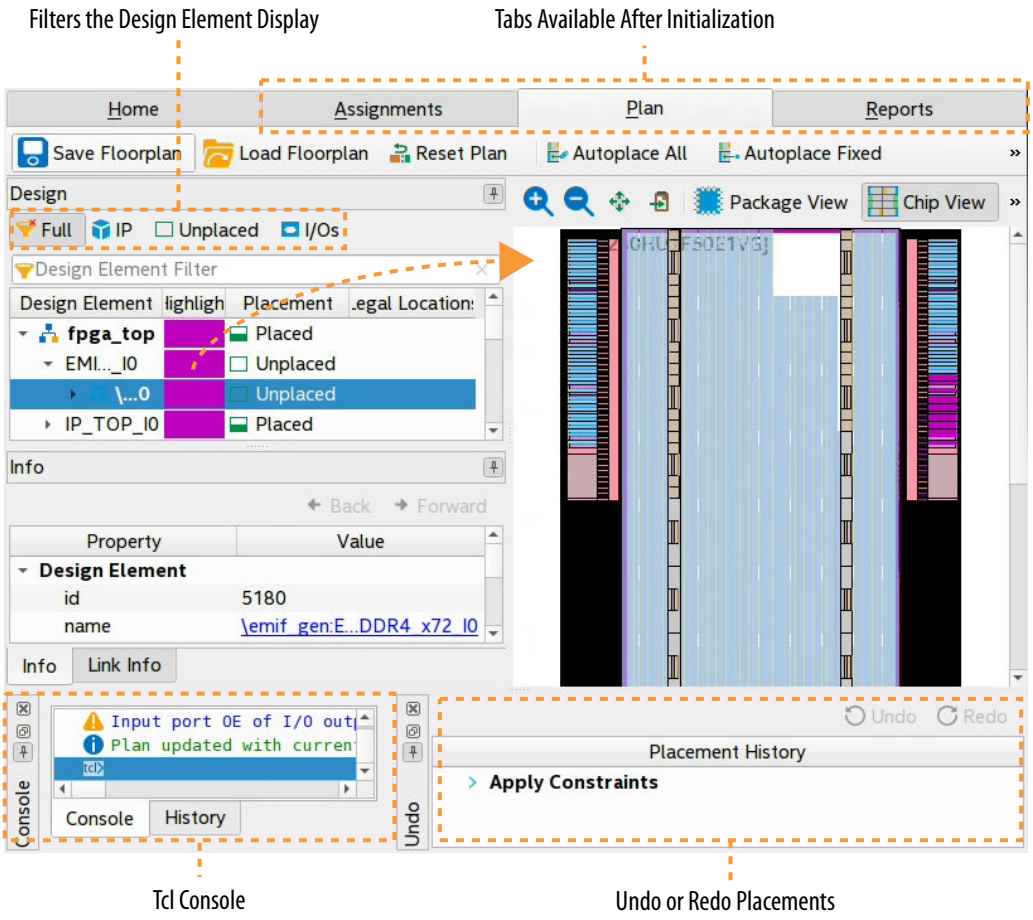
Interface Planner's user interface guides you through the design planning steps. Use Interface Planner's **Flow** control to execute the main initialization, planning, and validation functions of the flow in sequence.

Figure 8. Interface Planner Flow Control



As you run each step in the **Flow** control, downstream commands and the **Assignments**, **Plan**, and **Reports** tabs become available. Interface Planner only allows you to run commands after completing any prerequisite steps in the flow. After you **Initialize Interface Planner**, you are prompted to confirm any project assignments that you made before planning starts. Disable or enable any imported project assignments on the **Assignments** tab to resolve any conflicts and evaluate different implementations.

Figure 9. Interface Planner GUI



After you **Update Plan** with the project assignments, you are ready to place design elements onto the target device **Chip View** or **Package View** on the **Plan** tab. As you place design elements in the **Plan** tab, the Fitter verifies placement legality in real-time. Once planning is complete and validated, you export the constraints as a Tcl script for application in your project.

Note: The Interface Planner constraints you define do not apply to your project until you export and source them with the generated Tcl script.

Figure 10. Interface Planner Chip View

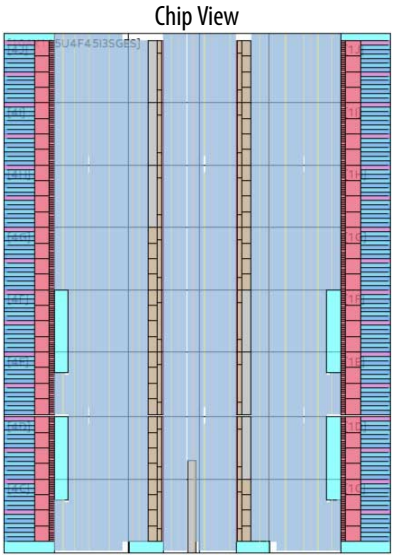
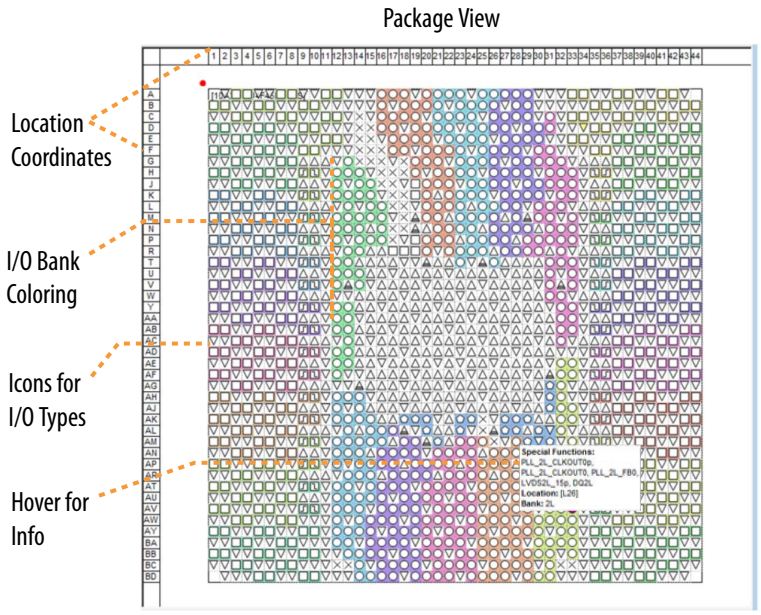


Figure 11. Interface Planner Package View



The following topics describe these interface planning flow steps in detail:

2.1.2.1. Step 1: Setup and Synthesize the Project

Interface Planner requires at least a partially complete, synthesized Quartus Prime project as input. You can also use Interface Planner to adjust placement for a fully complete design project.

Follow these steps to setup the project and run synthesis:

1. Complete at least the following steps for your design:
   - Fully define known device periphery interfaces.
   - Instantiate all known interface IP cores.
   - Declare all general purpose I/Os.
   - Define the I/O standard, voltage, drive strength, and slew rate for all general purpose I/Os.
   - Define the core clocking (optional, but recommended).
   - Connect all interfaces of the periphery IP to virtual pins or test logic. This technique creates loop backs on any interfaces in the shell design, helping to ensure that periphery interfaces persist after synthesis optimization.
2. To synthesize the design, click **Processing ➤ Start ➤ Start Analysis & Synthesis**. You must run at least Analysis & Synthesis before running Interface Planner.

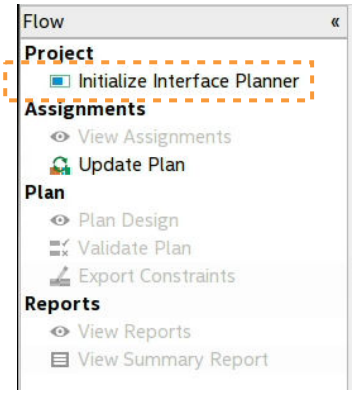
2.1.2.2. Step 2: Initialize Interface Planner

Initializing Interface Planner loads the compilation database for the synthesis snapshot, and enables the **View Assignments** command and **Assignments** tab for reconciling project assignments.

To initialize Interface Planner:

1. Click **Tools ➤ Interface Planner**. The Interface Planner opens, displaying the **Home** tab.
2. On the **Flow** control, click **Initialize Interface Planner**. After initialization, the Fitter dynamically validates your interface plan as you make changes.

Figure 12. Interface Planner Home Tab



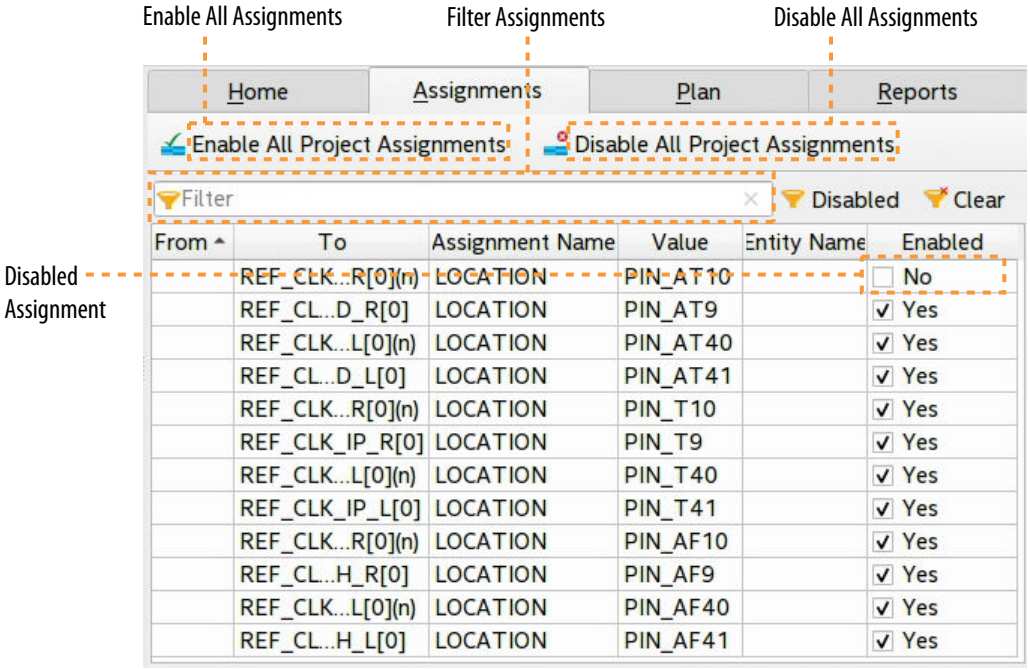
2.1.2.3. Step 3: Update Plan with Project Assignments

Before periphery planning in Interface Planner, you must reconcile any conflicting imported project assignments and **Update Plan** with the assignments you want to retain in the plan.

Follow these steps to review imported project assignments and reconcile any conflicts:

1. On the **Flow** control, click **View Assignments**.
2. On the **Assignments** tab, enable or disable specific or groups of project assignments to resolve any conflicts or experiment with different settings. You can filter the list of assignments by assignment name or status.
3. After resolving all conflicts, click **Update Plan** on the **Flow** control to apply the enabled project assignments to your interface plan.

Figure 13. Interface Planner (Assignments Tab)



Related Information

- Home Tab Controls on page 25
- Assignments Tab Controls on page 25
- AN 821: Interface Planning for Intel Stratix 10 FPGAs

2.1.2.4. Step 4: Plan Periphery Placement

Click **Plan Design** on the **Flow** control to interactively place IP cores and other design elements in legal locations in the device periphery. The **Plan** tab displays a list of your project's design elements, alongside a graphical abstraction of the target device architecture.

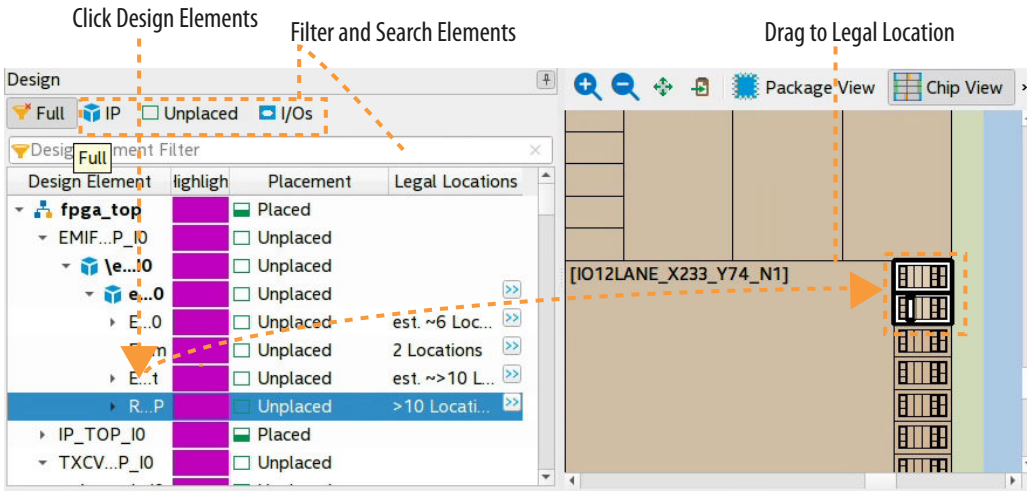
For efficiency, place design elements in the following order in Interface Planner:

1. Place all I/O pins or elements, such as PLLs, that have known, specific location requirements.
2. Place all known periphery interface IP.
3. (Optional) Place all remaining unplaced cells.

Use the following controls to place design elements in the Interface Planner floorplan:

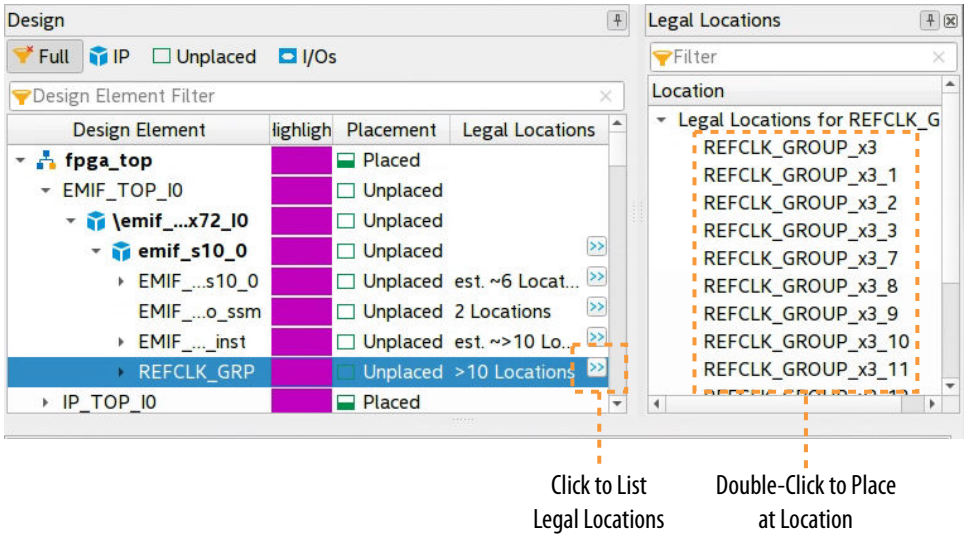
1. Locate design elements that you want to place in the **Design Element** list. You can search and filter the list by name, IP, placement status, I/Os, and other criteria.
2. To customize design element color coding definitions, click the **Highlight** column.

Figure 14. Interface Planner (Plan Tab)



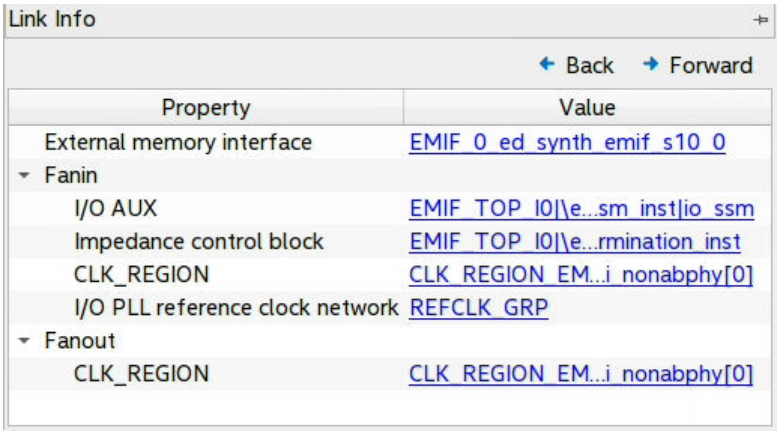
3. Use any of the following methods to place design elements in the floorplan:
   - Drag elements from the **Design Elements** list and drop them onto available device resources in the **Chip** or **Package** view. Use Ctrl+Click to drag and pan across the **Chip** or **Package** views. You may experience a small delay while dragging as Interface Planner calculates the legal locations.
   - To allow Interface Planner to place an unplaced design element in a legal location, right-click and select **Autoplace Selected**. You must use **Autoplace Selected** for all unplaced clocks .
   - Click the >> button next to the **Design Elements** to display a list of **Legal Locations**. Click any legal location in the list to highlight the location in the floorplan. Double-click any location in the list to place the element in the location.

Figure 15. Listing Legal Locations



4. To step forward and backward though your plan changes, click the **Undo** and **Redo** buttons.
5. To visualize and traverse design connectivity (for example, to view the reference clock pin and driven destination cells of a PLL), select any design element and then click the **Link Info** tab. Click the **Back** and **Forward** buttons to traverse design connectivity.
6. To generate a report that shows the placement locations the Fitter prefers, select a design element and click **Report Placeability of Selected Element**.

Figure 16. Link Info Tab for Traversing Connectivity



Note: Changes made in Interface Planner do not apply to your Quartus Prime project until you apply the generated interface plan constraints script to your project.

Related Information

Plan Tab Controls on page 26

2.1.2.4.1. Plan Clock Networks

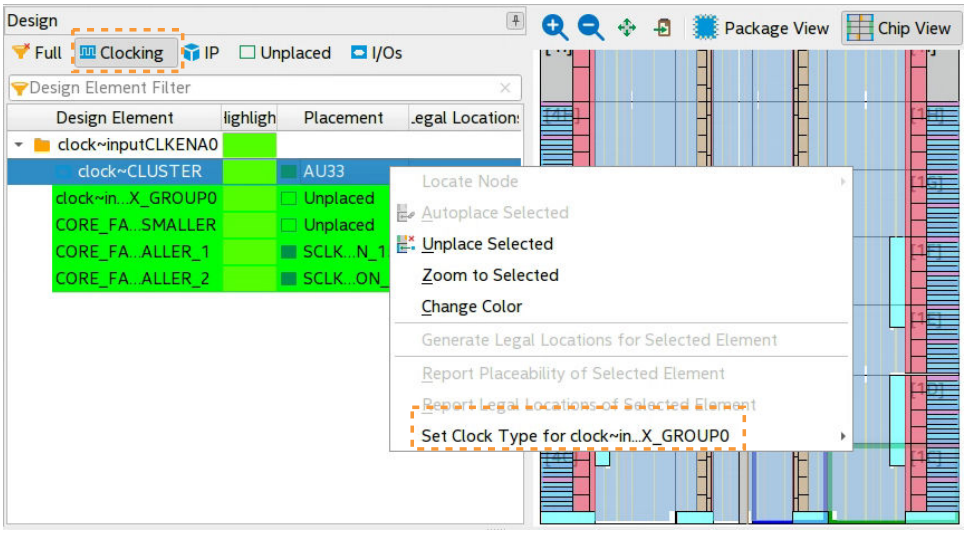
Interface Planner allows you to visualize and plan clock networks. For Arria® 10 and Cyclone® 10 GX devices, you can locate, highlight, place, and edit the type of clock elements in the **Plan** tab.

Note: The Stratix® 10 device family does not support the **Clocking** filter in Interface Planner. For Stratix 10 designs, use the **Autoplace Selected** command to place all unplaced clock elements.

Interface Planner generates a Clocks report that details the signals using low-skew routing networks (clock networks) in the device.

To identify and place clocking elements in your design, click the **Clocking** filter in the **Plan** tab. You can refine the list further by entering search text in the **Design Element Filter**. Interface Planner represents clock networks as groupings of the clock source, clock mux, and the clock region.

Figure 17. Clocking Design Elements



You can place an entire clock group or individual clock elements by dragging into the location, or using the **Report Legal Locations of Selected Element** or the **Autoplace Selected** commands. After placement, hover the cursor over the item in the **Design Element** list to highlight the placement. The placement of clock elements impacts the placement of dependent core and periphery elements.

You can edit the clock type for clocking design elements. The clock type impacts the placement of dependent core and periphery elements. Right-click any clock element to specify one of the available clock types.

2.1.2.4.2. Saving & Loading Floorplans

You can save the state of your Interface Planner floorplan for use in subsequent Interface Planner sessions. Interface Planner saves your plan in Interface Planner Floorplan Format (`.plan`). You can load a `.plan` file in Interface Planner to reopen the floorplan.

1. To save an Interface Planner floorplan, click **File ➤ Save Floorplan** and specify a file name.
2. To load an Interface Planner floorplan, click **File ➤ Load Floorplan** and browse for the `.plan` file.

Note: `.plan` files are for use only in Interface Planner and are not for use directly in the Quartus Prime software. Interface Planner generates an error if you attempt to load a `.plan` file that is not associated with the current Interface Planner project.

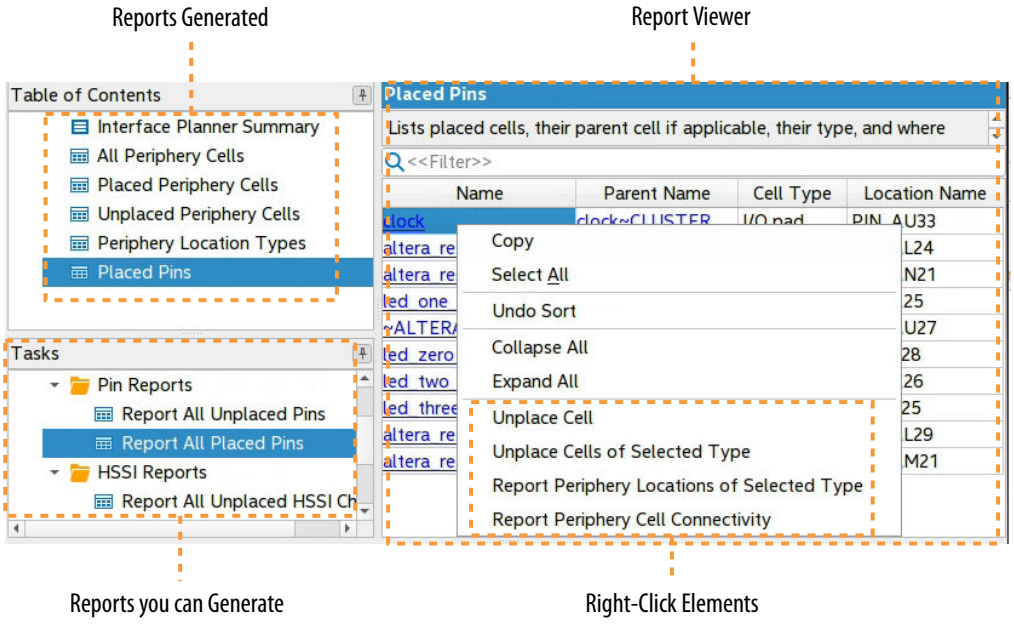
2.1.2.5. Step 5: Report Placement Data

Generate Interface Planner placement and connectivity reports to help locate cells and make the best decisions about placement for the interfaces and elements in your design. Click **View Reports** on the **Flow** control to open the **Reports** tab from which you can generate a range of reports.

Follow these steps to report Interface Planner placement data:

1. In the **Flow** control, click **View Reports**. The list of reports appears in the **Tasks** pane.
2. In the **Tasks** pane, double-click any report name to generate the report in the **Table of Contents** pane.
3. Select design elements in the report and click **Place**, **Unplace**, or report detailed data about the selected elements or locations.

Figure 18. Reports Tab



Related Information

- Reports Tab Controls on page 27
- Interface Planner Reports on page 44

2.1.2.6. Step 6: Validate and Export Plan Constraints

You must validate your interface plan before exporting the plan constraints to your project as a generated Tcl script. Validation must confirm that the Fitter can place all remaining unplaced design elements before you can generate the script. When you are satisfied with your interface plan, follow these steps to validate and apply the interface plan to your Quartus Prime project:

1. In the **Flow** control, click **Validate Plan**. The Fitter confirms placement of all remaining unplaced design elements. You must correct any errors before you can export constraints.
2. After validation, click **Export Constraints** to generate a Tcl script that applies the plan to your project. The output Tcl file contains instructions to apply the interface plan to your Quartus Prime project.
3. Close Interface Planner.
4. To apply the exported interface plan constraints to your Quartus Prime project, click **Tools ➤ Tcl Scripts** and select the `<project name>.pdp_assignments.tcl` script file.
5. Click **Run**. The script runs, applying the Interface Planner constraints to the project. Alternatively, you can run the script from the project directory:

   ```
   quartus_sh –t <assignments_file>.tcl
   ```

6. To run synthesis and apply the interface plan in your project, click **Start ➤ Start Analysis & Synthesis**.
7. Confirm the implementation of your plan by reviewing the Compilation Report.

2.1.3. Interface Planner NoC Tool Flow

For designs targeting Agilex 7 M-Series FPGAs only, you can use Interface Planner to assign physical locations for Network-on-Chip (NoC) initiators, PLLs, and SSMs. The Hard Memory NoC facilitates high-bandwidth data movement between the FPGA core logic and memory resources, such as HBM2e and DDR5 memories. Refer to the *Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide* for details on the complete NoC flow including Interface Planner.

You can use Interface Planner to assign physical locations for NoC initiators, targets (as part of the HBM2e or external memory interfaces), PLLs, and SSMs. If you do not make physical assignments for NoC elements, the Fitter places NoC elements automatically during compilation.

You use the floorplan view in Interface Planner to place hard memory NoC and periphery elements. There are three floorplan views available:

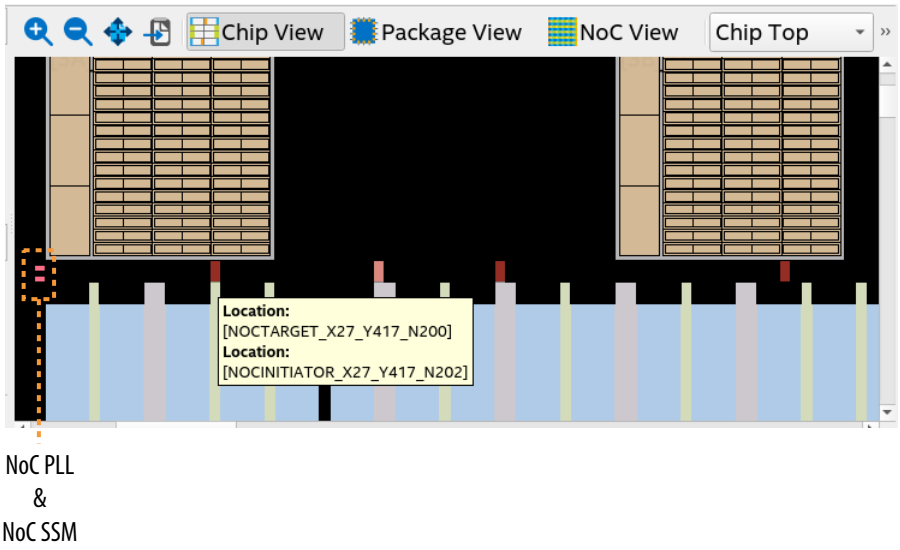
- **NoC View**—shows a filtered view of NoC initiators and targets.
- **Chip View**—shows the placeable locations for hard memory NoC elements, including NoC initiators, targets, PLLs, and SSMs.
- **Package View**—NoC elements are not visible in the **Package View**.

In the **Chip View**, the available NoC initiator and target locations appear as rows of small boxes across the top and bottom edges of the device, between the FPGA fabric and the periphery I/O structures. Placing your cursor over locations displays a tooltip indicating whether the location supports only an initiator, only a target, or both an initiator and a target.

The available NoC PLL and NoC SSM locations appear as smaller boxes at the end of the row of initiators and targets. The PLL and SSM locations appear at the left end of the rows (if using the **Chip Top** view), or at the right end of the rows (if using the **Chip Bottom** view).

Interface Planner Chip View, Closeup of NoC Features shows an example of the Interface Planner Chip View showing the top left corner of the die as viewed from the top. The two smaller pink boxes at the top left corner of the fabric are the locations of the NoC PLL and the NoC SSM.

Figure 19. Interface Planner Chip View, Closeup of NoC Features

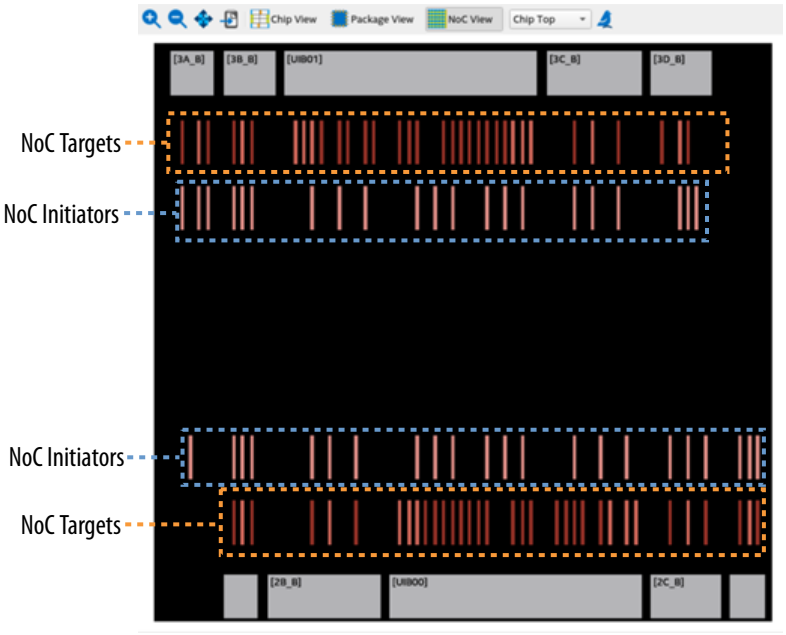


In the **NoC View**, only the NoC initiators and targets are visible as larger rectangles. The targets and initiators for both high-speed NoC along the top edge of the die, and the high-speed NoC along the bottom edge of the die, are visible. Initiators and targets that may share the same location in the Chip View are split into separate elements in the **NoC View**.

The outer-top and outer-bottom rows are the targets for the top-edge NoC and bottom-edge NoC, respectively. Similarly, the inner-top and inner-bottom rows are the initiators for the top-edge NoC and bottom-edge NoC, respectively. As with the **Chip View**, if you place your cursor over one of these locations, a tooltip reports if that location supports a target or an initiator.

NoC View Showing Targets and Initiators is an example of the Interface Planner **NoC View**, showing the targets and initiators for both the top-edge NoC and the bottom-edge NoC.

Figure 20. NoC View Showing Targets and Initiators



Related Information

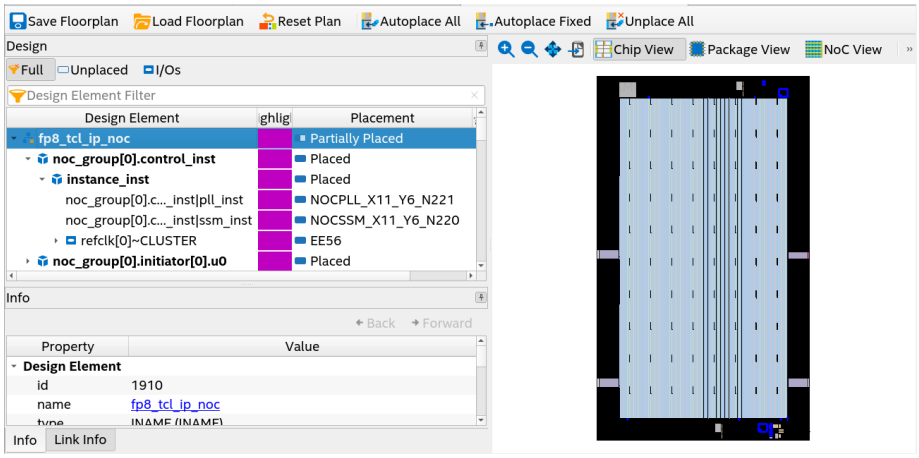
Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide

2.1.3.1. Placing NoC Design Elements Using Interface Planner

The following steps describe recommended placement of NoC design elements:

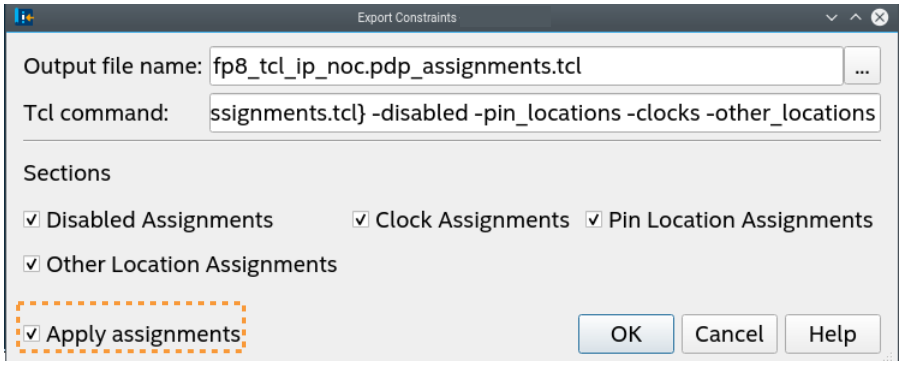
1. Open, initialize, and load assignments in Interface Planner, as Interface Planner General Tool Flow describes.
2. Click **Plan Design** on the **Flow** control to interactively place NoC elements and other design elements in legal locations in the device periphery. All placeable elements, including NoC elements and periphery elements, appear in the **Design Elements** list. Refer to Recommended Placement Order for NoC Elements in Interface Planner.
3. Use any of the following methods to place design elements in the **Chip View**:
   - Drag NoC elements from the **Design Elements** list and drop them onto available device resources in the **Chip view**. You may experience a small delay while dragging as Interface Planner calculates the legal locations.
   - To allow Interface Planner to place an unplaced design element in a legal location, right-click and select **Autoplace Selected**. You must use **Autoplace Selected** for all unplaced clocks.
   - Right-click an element the **Design Elements** list, and then click **Generate Legal Locations** to display a list of **Legal Locations** for the element. Click any legal location in the list to highlight the location in the floorplan. Double-click any location in the list to place the element in the location.
4. After making all necessary location assignments in Interface Planner, validate the placement by clicking **Validate Plan** in the **Flow** pane.

Figure 21. NoC Elements in Interface Planner Design Tab Chip View



5. To generate a Tcl script to apply the placement constraints to your project, click **Export Constraints** in the **Flow** pane. To automatically run the Tcl script, enable **Apply Assignments**.

Figure 22. Export Physical Constraints from Interface Planner to Your Project



6. To report whether the NoC initiator and target location placement allows your design to meet the bandwidth and transaction size requirements, click the **Reports** tab, and then double-click **Report NoC Performance** in the **Tasks** pane. For report details, refer to Report NoC Performance.

2.1.3.1.1. Recommended Placement Order for NoC Elements in Interface Planner

For best results, place NoC-related elements in the following order:

Note: For important considerations when choosing initiator interface placement, refer to *Horizontal Bandwidth Considerations* in the *Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide* to translate location choices into physical placements.

1. Start by placing the NoC PLL and SSM in the Interface Planner **Chip View**. Expand the contents of the NoC Clock Control IP by clicking the small triangle to the left of the IP instance name (`noc_clock_ctrl_0`) in the **Design Element** pane. Place the PLL and SSM instances for each clock control IP at either the top corner or the bottom corner.
2. Use the **Autoplace Selected** command to place the remaining NoC Clock Control Intel FPGA IP.
3. If using HBM2e memory, start by placing the UIB PLL using the Interface Planner **Chip View**. Expand the contents of the HBM2e IP by clicking the small triangle to the left of the IP instance name (`hbm_fp_0`) in the **Design Element** pane. Interface Planner may display legal locations on both the top edge and on the bottom edge of the die. Place the IP along the same edge of the die as the corresponding NoC PLL.
4. Place the HBM2e instance (design element name ending in …|`xhbmc`). Interface Planner displays only one legal location for the HBM2e instance.
5. Use **Autoplace Selected** to place the remaining HBM2e IP, including all NoC target interfaces.
6. If using high-speed external memory interfaces that connect to the NoC, place these interfaces next. As with the HBM2e Intel FPGA IP above, start by placing the PLL for the external memory interface. Ensure you place this interface along the same edge as the corresponding NoC PLL.
7. Place the `mem_ck` pins to fix the pin-out for the interface.
8. Use **Autoplace Selected** to place the remaining external memory interface IP, including all of the NoC target interfaces.
9. Select the **NoC View** to place the initiator interfaces. The **NoC View** shows the target interfaces that you already placed. As you place each initiator, the targets you connect to highlight. When placing each initiator, consider which targets communicate with the initiator.

Low-speed external memory interfaces and other GPIO functions that bypass the NoC can conflict with initiator interface placement. Depending on design requirements, you can place these I/O functions that bypass the NoC before or after placing the NoC initiator interfaces. Placing I/O functions first gives greater flexibility to their placement, while restricting which initiator locations you can use. Placing NoC initiator interfaces first allows optimal initiator placement, while restricting which I/O locations are available.

Other interfaces, such as transceivers, have no direct interaction with the hard memory NoC. Therefore, you can place such interfaces before or after the NoC.

Related Information

Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide

2.1.3.1.2. High-Speed Interconnect NoC Locations in Interface Planner

When placing NoC initiators and targets, refer to the following tables to correlate locations with NoC elements visible in the Interface Planner.

Table 12. Top-Edge High-Speed Interconnect NoC Locations in Interface Planner

NoC Segment	Initiator	Target	Interface Planner Location
PLL/SSM			NOCPLL_X11_Y417_N221 NOCSSM_X11_Y417_N220
GPIO-B_0	I0	T0	NOCINITIATOR_X27_Y417_N202 NOCTARGET_X27_Y417_N200
	I1	AXI4 Lite	NOCINITIATOR_X42_Y417_N202 NOCAXILITETARGET_X42_Y417_N200
	I2	T2	NOCINITIATOR_X53_Y417_N202 NOCTARGET_X53_Y417_N200
GPIO-B_1	I0	T0	NOCINITIATOR_X79_Y417_N202 NOCTARGET_X79_Y417_N200
	I1	AXI4 Lite	NOCINITIATOR_X94_Y417_N202 NOCAXILITETARGET_X94_Y417_N200
	I2	T2	NOCINITIATOR_X105_Y417_N202 NOCTARGET_X105_Y417_N200
UIB_L		AXI4 Lite	NOCAXILITETARGET_X123_Y417_N200
		AXI4 Lite	NOCAXILITETARGET_X129_Y417_N200
	I0	AXI4 Lite	NOCINITIATOR_X134_Y417_N202 NOCAXILITETARGET_X134_Y417_N200
		T3	NOCTARGET_X140_Y417_N200
	I1	T4	NOCINITIATOR_X150_Y417_N202 NOCTARGET_X150_Y417_N200
		T5	NOCTARGET_X156_Y417_N200
	I2	T6	NOCINITIATOR_X161_Y417_N202 NOCTARGET_X161_Y417_N200
		T7	NOCTARGET_X167_Y417_N200
UIB_M		T0	NOCTARGET_X177_Y417_N200
		T1	NOCTARGET_X183_Y417_N200
	I0	T2	NOCINITIATOR_X188_Y417_N202 NOCTARGET_X188_Y417_N200
	I1		NOCINITIATOR_X204_Y417_N202
		T3	NOCTARGET_X210_Y417_N200
	I2	T4	NOCINITIATOR_X215_Y417_N202 NOCTARGET_X215_Y417_N200
		T5	NOCTARGET_X221_Y417_N200
UIB_R		T0	NOCTARGET_X231_Y417_N200
		T1	NOCTARGET_X237_Y417_N200
	I0	T2	NOCINITIATOR_X242_Y417_N202 NOCTARGET_X242_Y417_N200
		T3	NOCTARGET_X248_Y417_N200
	I1	T4	NOCINITIATOR_X258_Y417_N202 NOCTARGET_X258_Y417_N200
		AXI4 Lite	NOCAXILITETARGET_X264_Y417_N200

continued...

NoC Segment	Initiator	Target	Interface Planner Location
	I2	AXI4 Lite	NOCINITIATOR_X269_Y417_N202 NOCAXILITETARGET_X269_Y417_N200
		AXI4 Lite	NOCAXILITETARGET_X275_Y417_N200
GPIO-B_3	I0	T0	NOCINITIATOR_X296_Y417_N202 NOCTARGET_X296_Y417_N200
	I1	AXI4 Lite	NOCINITIATOR_X311_Y417_N202 NOCAXILITETARGET_X311_Y417_N200
	I2	T2	NOCINITIATOR_X322_Y417_N202 NOCTARGET_X322_Y417_N200
GPIO-B/HPS		T0	NOCTARGET_X346_Y417_N200
	I0(fabric)	AXI4 Lite	NOCINITIATOR_X357_Y417_N204 NOCAXILITETARGET_X357_Y417_N200
	I1(fabric)	T2	NOCINITIATOR_X365_Y417_N204 NOCTARGET_X365_Y417_N200
	I0(MPFE) AXI4 Lite I2(MPFE)		NOCINITIATOR_X373_Y417_N202 NOCAXILITEINITIATOR_X373_Y417_N201 NOCINITIATOR_X373_Y417_N200

Table 13. Bottom-Edge High-Speed Interconnect NoC Locations in Interface Planner

NoC Segment	Initiator	Target	Interface Planner Location
PLL/SSM			NOCPLL_X11_Y6_N221 NOCSSM_X11_Y6_N220
SDM	I0		NOCINITIATOR_X28_Y6_N200
GPIO-B_1	I0	T0	NOCINITIATOR_X79_Y6_N202 NOCTARGET_X79_Y6_N200
	I1	AXI4 Lite	NOCINITIATOR_X94_Y6_N202 NOCAXILITETARGET_X94_Y6_N200
	I2	T2	NOCINITIATOR_X105_Y6_N202 NOCTARGET_X105_Y6_N200
GPIO-B_2	I0	T0	NOCINITIATOR_X134_Y6_N202 NOCTARGET_X134_Y6_N200
	I1	AXI4 Lite	NOCINITIATOR_X149_Y6_N202 NOCAXILITETARGET_X149_Y6_N200
	I2	T2	NOCINITIATOR_X160_Y6_N202 NOCTARGET_X160_Y6_N200
UIB_L		AXI4 Lite	NOCAXILITETARGET_X177_Y6_N200
		AXI4 Lite	NOCAXILITETARGET_X183_Y6_N200
	I0	AXI4 Lite	NOCINITIATOR_X188_Y6_N202 NOCAXILITETARGET_X188_Y6_N200
		T3	NOCTARGET_X194_Y6_N200
	I1	T4	NOCINITIATOR_X204_Y6_N202 NOCTARGET_X204_Y6_N200
		T5	NOCTARGET_X210_Y6_N200
	I2	T6	NOCINITIATOR_X215_Y6_N202 NOCTARGET_X215_Y6_N200
		T7	NOCTARGET_X221_Y6_N200
UIB_M		T0	NOCTARGET_X231_Y6_N200
		T1	NOCTARGET_X237_Y6_N200

continued...

NoC Segment	Initiator	Target	Interface Planner Location
	I0	T2	NOCINITIATOR_X242_Y6_N202 NOCTARGET_X242_Y6_N200
	I1		NOCINITIATOR_X258_Y6_N202
		T3	NOCTARGET_X264_Y6_N200
	I2	T4	NOCINITIATOR_X269_Y6_N202 NOCTARGET_X269_Y6_N200
		T5	NOCTARGET_X275_Y6_N200
UIB_R		T0	NOCTARGET_X285_Y6_N200
		T1	NOCTARGET_X291_Y6_N200
	I0	T2	NOCINITIATOR_X296_Y6_N202 NOCTARGET_X296_Y6_N200
		T3	NOCTARGET_X302_Y6_N200
	I1	T4	NOCINITIATOR_X312_Y6_N202 NOCTARGET_X312_Y6_N200
		AXI4 Lite	NOCAXILITETARGET_X318_Y6_N200
	I2	AXI4 Lite	NOCINITIATOR_X323_Y6_N202 NOCAXILITETARGET_X323_Y6_N200
		AXI4 Lite	NOCAXILITETARGET_X329_Y6_N200
GPIO-B_3	I0	T0	NOCINITIATOR_X350_Y6_N202 NOCTARGET_X350_Y6_N200
	I1	AXI4 Lite	NOCINITIATOR_X365_Y6_N202 NOCAXILITETARGET_X365_Y6_N200
	I2	T2	NOCINITIATOR_X376_Y6_N202 NOCTARGET_X376_Y6_N200
GPIO-B_4	I0	T0	NOCINITIATOR_X404_Y6_N202 NOCTARGET_X404_Y6_N200
	I1	AXI4 Lite	NOCINITIATOR_X419_Y6_N202 NOCAXILITETARGET_X419_Y6_N200
	I2	T2	NOCINITIATOR_X430_Y6_N202 NOCTARGET_X430_Y6_N200

2.1.4. Interface Planner Reports

Use Interface Planner reports to locate cells and assign suitable placement locations for specific interfaces and elements in your design. Interface Planner reports provide detailed, actionable feedback to help you quickly implement the best plan for your design. You can access placement and further reporting functions directly from Interface Planner reports. Interface Planner generates the following reports that provide detailed planning information:

2.1.4.1. Report Summary

Click **Create all Summary Reports** on the **Reports** tab to generate summary reports about periphery cells in the interface plan. Right-click any cell type to report placed, unplaced, connectivity, or location information.

Figure 23. Summary Reports

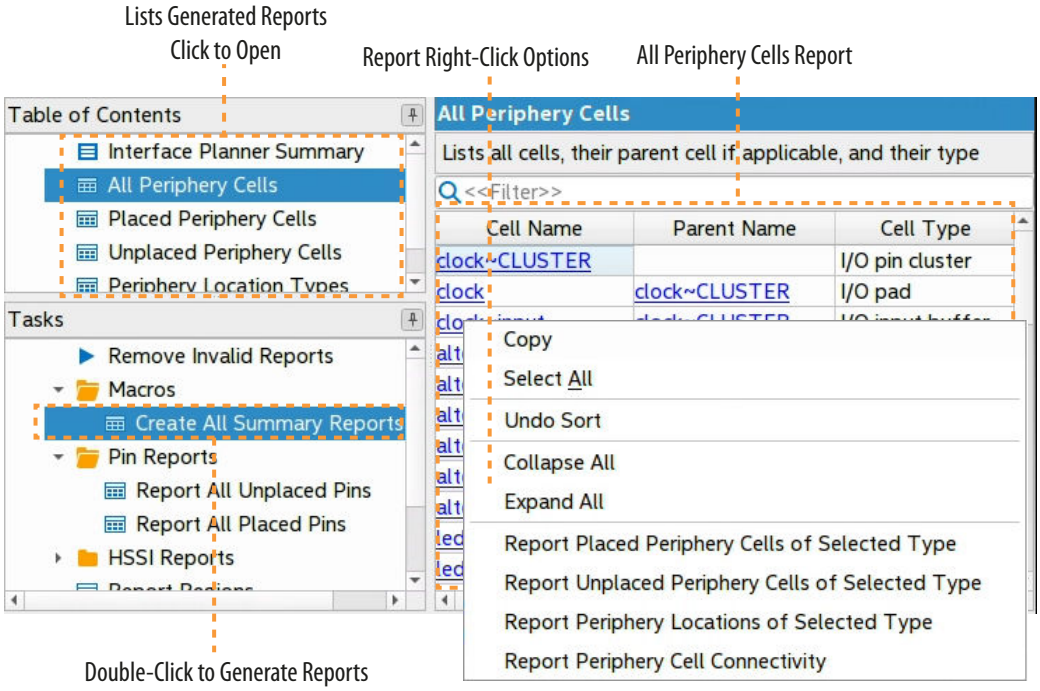


Table 14. Report Summary

Command	Description
Create all Summary Reports	Creates the following summary reports: • **Interface Planner Summary**—reports software version and total number of periphery and top-level periphery cells. • **All Periphery Cells**— reports the name, parent, and type of all periphery cells in the design. • **Placed/Unplaced Periphery Cells**—reports the name, parent, and type of all placed and unplaced periphery cells in the interface plan. • **Periphery Location Types**—reports the number of each type of periphery location available in the target device and the number required by your design.

2.1.4.2. Report Pins

Generate reports about I/O pins in the design. Right-click any cell type to place, unplace, or report connectivity or location information.

Table 15. Report Pin Commands

Command	Description
Report All Placed Pins	Generates the Placed Pins report. This report lists the name, parent, type, and location of all placed pins in the interface plan.
Report All Unplaced Pins	Generates the Unplaced Pins report. This report lists the name, parent, type, and the number of potential placements for all unplaced pins in the interface plan.

Figure 24. Placed Pins Report

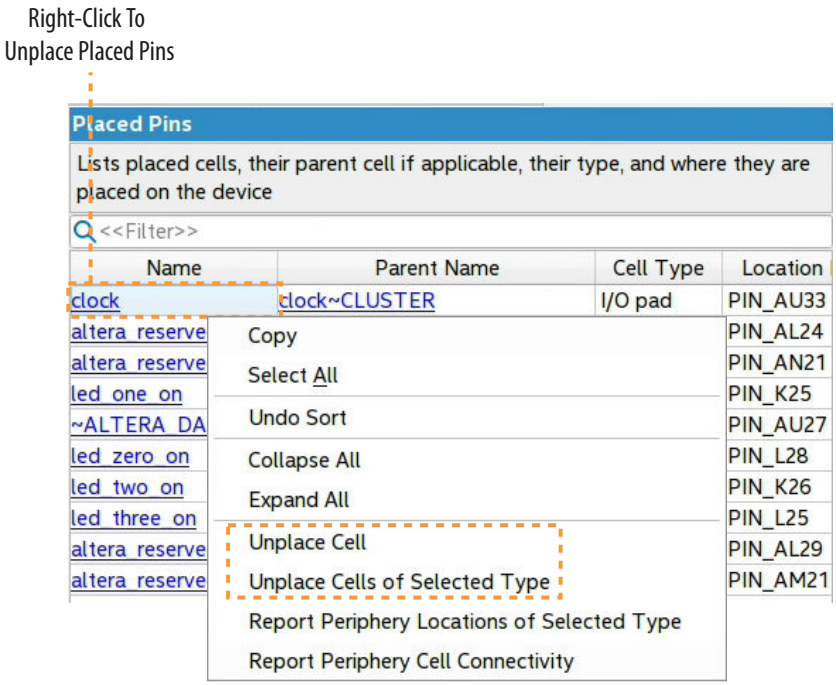
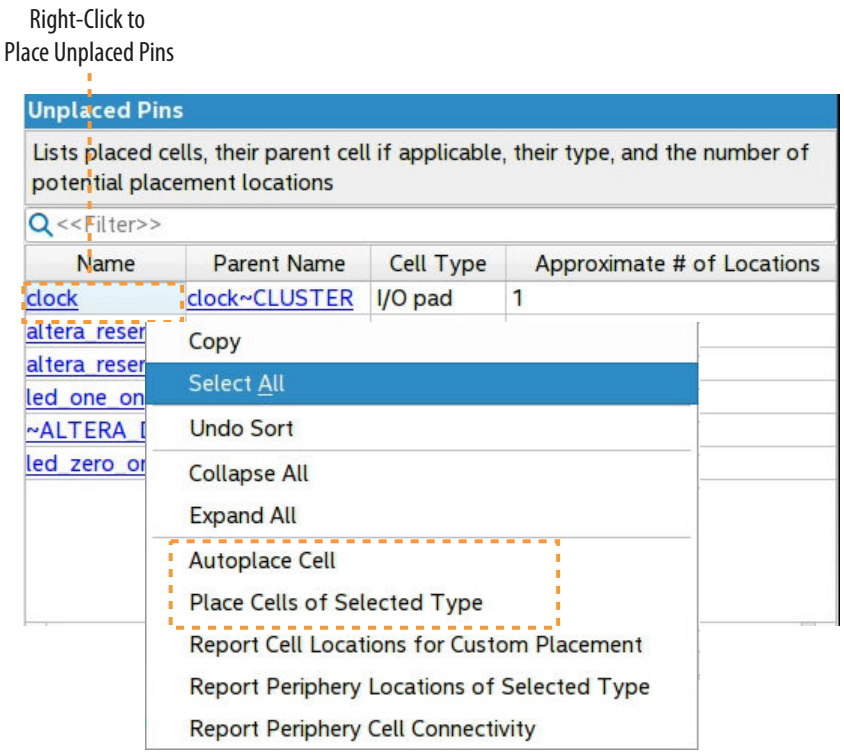


Figure 25. Unplaced Pins Report



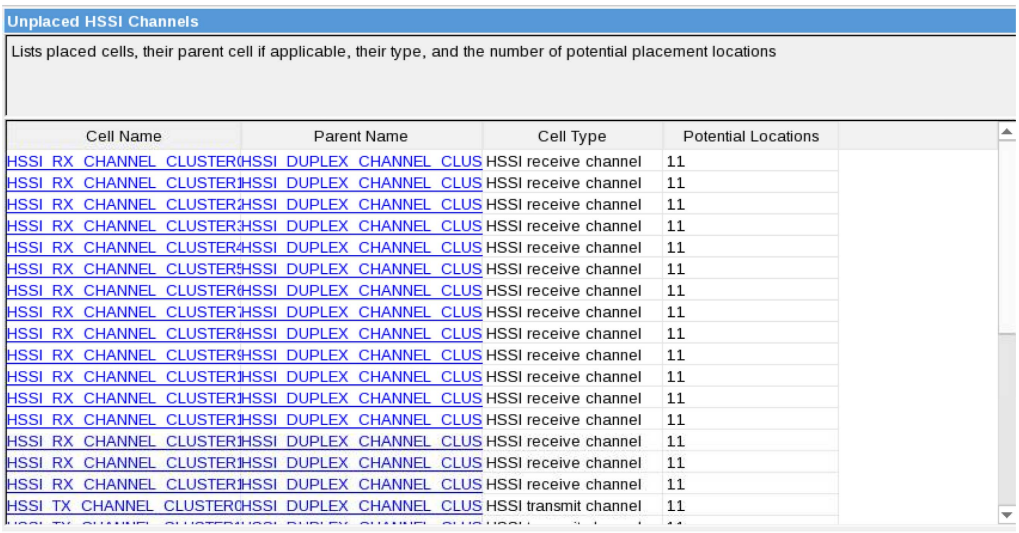
2.1.4.3. Report HSSI Channels

Generate reports about HSSI channels in the interface plan. Right-click any cell type to place, unplace, or report connectivity or location information.

Table 16. Report Channel Commands

Command	Description
Report All Placed HSSI Channels	Generates the Placed HSSI Channels report. This report lists the name, parent, type, and location of all placed HSSI RX/TX channels in the interface plan.
Report All Unplaced HSSI Channels	Generates the Unplaced HSSI Channels report. This report lists the name, parent, type, and location of all unplaced HSSI RX/TX channels in the interface plan.

Figure 26. Unplaced HSSI Channels Report



Unplaced HSSI Channels

Lists placed cells, their parent cell if applicable, their type, and the number of potential placement locations

Cell Name	Parent Name	Cell Type	Potential Locations
HSSI_RX_CHANNEL_CLUSTER0	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER1	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER2	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER3	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER4	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER5	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER6	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER7	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER8	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER9	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER10	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER11	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER12	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER13	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_RX_CHANNEL_CLUSTER14	HSSI_DUPLEX_CHANNEL_CLUS	HSSI receive channel	11
HSSI_TX_CHANNEL_CLUSTER0	HSSI_DUPLEX_CHANNEL_CLUS	HSSI transmit channel	11

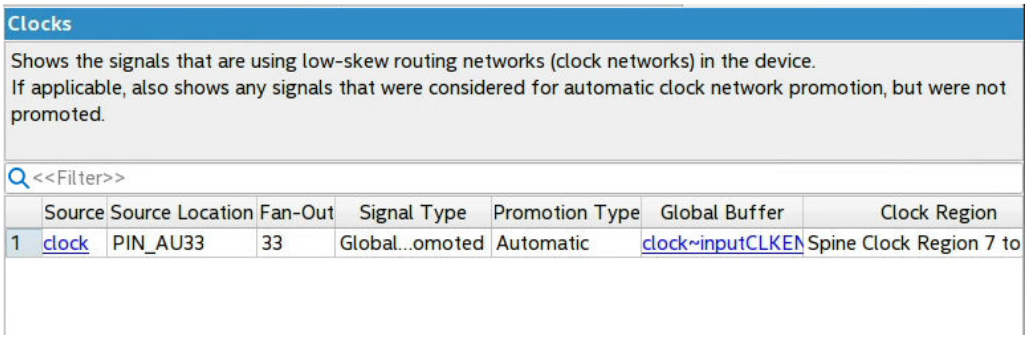
2.1.4.4. Report Clocks

Generate reports showing clock networks in the plan. Use this report to analyze clock network scenarios and ensure that specific device regions are fed by high fan-out signals.

Table 17. Report Clocks Commands

Command	Description
Report Clocks	Generates the Global and other Fast Signals report.

Figure 27. Clocks Report



Clocks

Shows the signals that are using low-skew routing networks (clock networks) in the device.
If applicable, also shows any signals that were considered for automatic clock network promotion, but were not promoted.

<<Filter>>

	Source	Source Location	Fan-Out	Signal Type	Promotion Type	Global Buffer	Clock Region
1	clock	PIN_AU33	33	Global...omoted	Automatic	clock~inputCLKEN	Spine Clock Region 7 to

2.1.4.5. Report Periphery Locations

Generate reports that show the status of periphery cells in the interface plan.

Table 18. Report Periphery Locations Commands

Command	Description
Right-click ➤ Report Placed Periphery Cells of Selected Type	Accessible from the **All Periphery Cells** report. This command reports the name, parent (if any), type, and location of the selected placed periphery cells matching the selected type. Right-click any cell to place, unplace, or report connectivity or location information.
Right-click ➤ Report Unplaced Periphery Cells of Selected Type	Accessible from the **All Periphery Cells** report. This command reports the name, parent (if any), type, and number of suitable locations for the selected unplaced periphery cells matching the selected type. Right-click any cell to place, unplace, or report connectivity or location information.
Right-click ➤ Report Periphery Locations of Selected Type	Reports all locations in the device of the selected type, and whether the location supports merging.

Figure 28. Placed Periphery Cells Report

Placed Periphery Cells

Lists placed cells, their parent cell if applicable, their type, and where they are placed on the

<<Filter>>

Name	Parent Name	Cell Type	Location Name
clock~CLUSTER		I/O pin cluster	AU33
clock	clock~CLUSTER	I/O pad	PIN_AU33
clock~input	clock~CLUSTER	I/O input buffer	IOIBUF_X78_Y125_N32
altera_reserved_tms~CLUS		I/O pin cluster	AL24
altera_reserved_tms	altera_reserved_tms~	I/O pad	PIN_AL24
altera_reserved_tms~input	altera_reserved_tms~	I/O input buffer	IOIBUF_X115_Y0_N65
altera_reserved_tdi~CLUST		I/O pin cluster	AN21
altera_reserved_tdi	altera_reserved_tdi~(	I/O pad	PIN_AN21
altera_reserved_tdi~input	altera_reserved_tdi~(	I/O input buffer	IOIBUF_X115_Y0_N47
led_one_on~CLUSTER		I/O pin cluster	K25
led_one_on	led_one_on~CLUSTE	I/O pad	PIN_K25
led_one_on~output	led_one_on~CLUSTE	I/O output buffer	IOOBUF_X142_Y205_N18

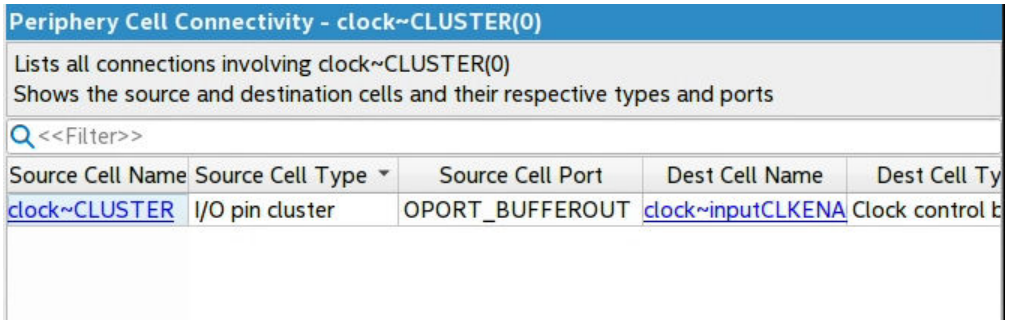
2.1.4.6. Report Cell Connectivity

Generate reports showing the connections between all cells in the interface plan.

Table 19. Report Cell Connectivity Command

Command	Description
Right-click ➤ Report Periphery Cell Connectivity	Right-click any **Cell Name** in the reports to **Report Periphery Cell Connectivity**. The report lists the source and destination ports and type of connections to the selected cell. Right-click any cell to report all connections to the cell.

Figure 29. Periphery Cell Connectivity Report



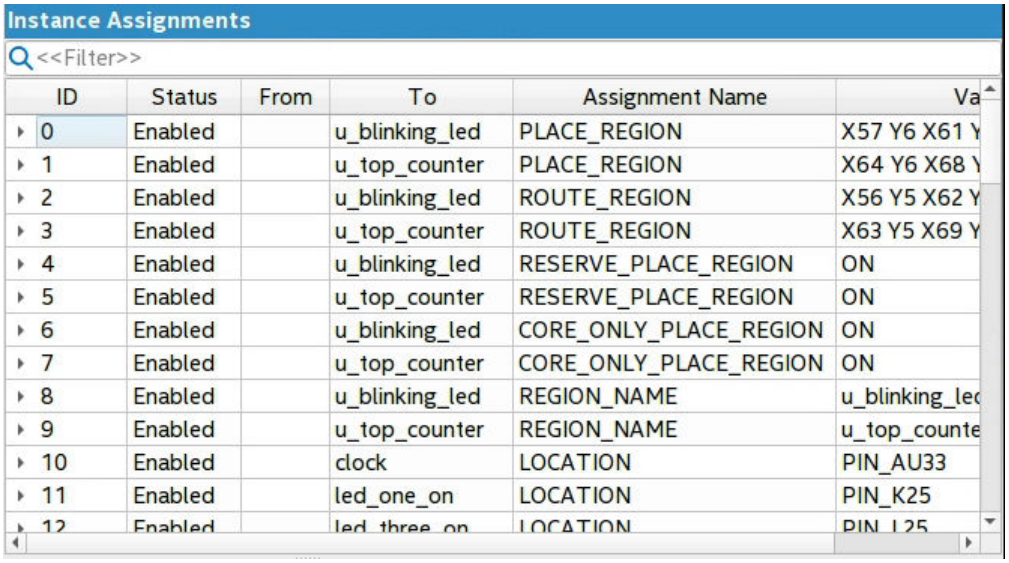
2.1.4.7. Report Instance Assignments

Click **Report Instance Assignments** to show all imported project assignments in the interface plan. You can delete these assignments from the plan.

Table 20. Report Instance Assignments Command

Command	Description
Report Instance Assignments	Reports all enabled instance assignments in your design. Right-click any cell to delete the assignment or to delete all assignments of the same type.

Figure 30. Instance Assignments Report



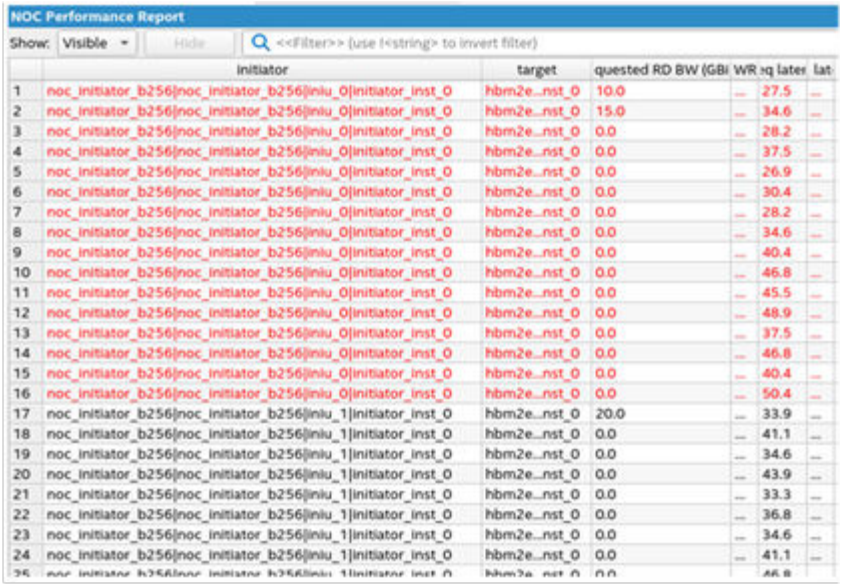
2.1.4.8. Report NoC Performance

For designs that include the Hard Memory NoC, you can interactively generate a NoC Performance Report in Interface Planner.

The NoC Performance Report generation performs a static analysis of the NoC initiator and target locations to evaluate whether the placement allows your design to meet the bandwidth requirements and transaction sizes that you specify in the NoC Assignment Editor. You can review the report, and then make changes in the **Plan** tab based on the results.

To access the NoC Performance Report in Interface Planner, click the **Reports** tab, and then double-click **Report NoC Performance** in the **Tasks** pane.

Figure 31. Sample NoC Performance Report



NOC Performance Report

Show: Visible — Hide — <<Filter>> (use !<string> to invert filter)

	initiator	target	quested RD BW (GBi	WR	q latei	lat
1	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	10.0	...	27.5	...
2	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	15.0	...	34.6	...
3	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	28.2	...
4	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	37.5	...
5	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	26.9	...
6	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	30.4	...
7	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	28.2	...
8	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	34.6	...
9	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	40.4	...
10	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	46.8	...
11	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	45.5	...
12	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	48.9	...
13	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	37.5	...
14	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	46.8	...
15	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	40.4	...
16	noc_initiator_b256|noc_initiator_b256|iniu_0|initiator_inst_0	hbm2e...nst_0	0.0	...	50.4	...
17	noc_initiator_b256|noc_initiator_b256|iniu_1|initiator_inst_0	hbm2e...nst_0	20.0	...	33.9	...
18	noc_initiator_b256|noc_initiator_b256|iniu_1|initiator_inst_0	hbm2e...nst_0	0.0	...	41.1	...
19	noc_initiator_b256|noc_initiator_b256|iniu_1|initiator_inst_0	hbm2e...nst_0	0.0	...	34.6	...
20	noc_initiator_b256|noc_initiator_b256|iniu_1|initiator_inst_0	hbm2e...nst_0	0.0	...	43.9	...
21	noc_initiator_b256|noc_initiator_b256|iniu_1|initiator_inst_0	hbm2e...nst_0	0.0	...	33.3	...
22	noc_initiator_b256|noc_initiator_b256|iniu_1|initiator_inst_0	hbm2e...nst_0	0.0	...	36.8	...
23	noc_initiator_b256|noc_initiator_b256|iniu_1|initiator_inst_0	hbm2e...nst_0	0.0	...	34.6	...
24	noc_initiator_b256|noc_initiator_b256|iniu_1|initiator_inst_0	hbm2e...nst_0	0.0	...	41.1	...

The NoC Performance Report reports performance data for each initiator to target connection. The latencies in this report are based on the minimum structural latency with respect to the initiator and target placement. These latencies are for the NoC portion of the path only. These latencies do not include any latency of, for instance, external memory access. Nor do these latencies account for potential delay due to congestion on the NoC. You can achieve lower minimum structural latency by placing the NoC initiators and targets closer together.

Table 21. NoC Performance Report Data

NoC Performance Report Column	Description
Requested RD BW	The requested read bandwidth.
Requested WR BW	The requested write bandwidth.
NoC Req latency	The requested request latency.
NoC Res latency	The requested response latency.

continued...

NoC Performance Report Column	Description
Initiator placement	The placement location of the initiator element.
Target placement	The placement location of the target element.
Message	Text message that identifies the reason that the current placement cannot meet the requested bandwidth.

One possible reason the **Message** reports that the current placement cannot meet the requested bandwidth is because of over-saturation of an initiator or a target. For example, if the sum of all bandwidth requirements through a particular initiator is greater than the bandwidth that the initiator can support, based on the data width and operating frequency of its AXI4 interface. To avoid this problem, either reduce bandwidth requirements or increase bandwidth capability.

Another possible reason that the current placement cannot meet the requested bandwidth is over-saturation of the horizontal bandwidth available in the NoC. This condition is the result of multiple initiator to target connections requesting bandwidth in the same direction through a horizontal section of the NoC. The NoC Performance Report message reports the congested segments. You can adjust initiator placement to alleviate congestion.

Note: For important considerations when choosing initiator interface placement, refer to the tables in *Horizontal Bandwidth Considerations*, along with tables in *High-Speed Interconnect NoC Locations in Interface Planner* in the *Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide* to translate location choices into physical placements.

You can also view NoC elements in the Quartus Prime Chip Planner, and view connectivity of NoC elements in following fitting in the NoC Connectivity Report, as the *Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide*.

Related Information

Agilex 7 M-Series FPGA Network-on-Chip (NoC) User Guide

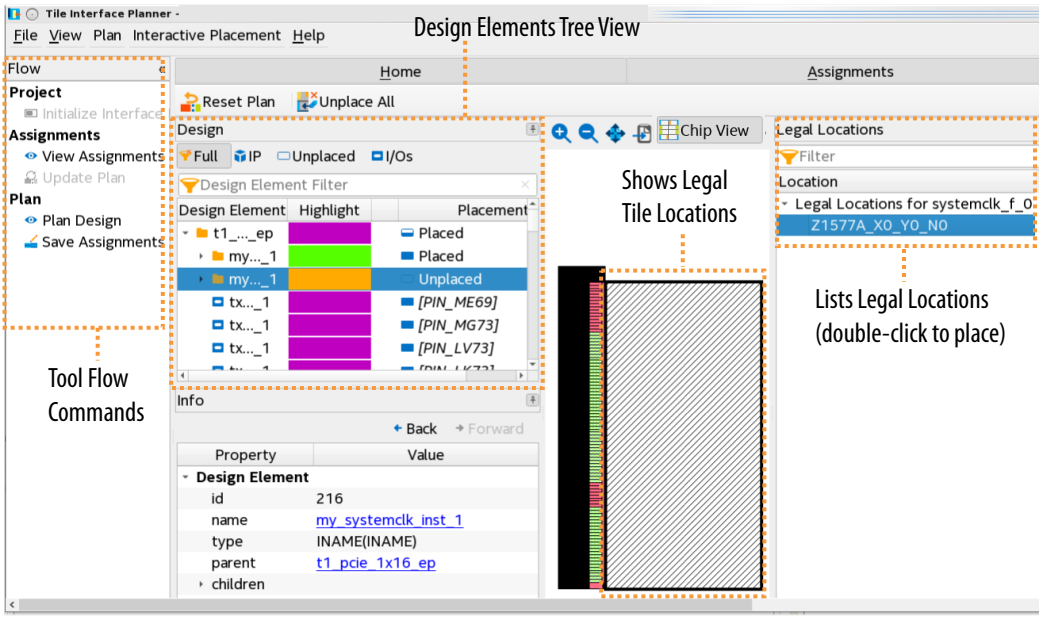
2.2. Using Tile Interface Planner

The Quartus Prime Tile Interface Planner helps you to quickly place component IP in legal tile locations of device F-tiles. Tile Interface Planner is an interactive floorplanning tool that simplifies this process.

Tile Interface Planner displays your project's component IP in a hierarchical tree view, next to a visual representation of the device tile segments. You can then locate the potential legal locations for each IP within the tile, place each IP at one of these locations, and apply generated placement constraints to the project for downstream Compiler stages.

As you place elements in the tile floorplan, the legality engine verifies that the placement is legal in real-time, thus ensuring correlation with your intent in the final implementation.

Figure 32. Tile Interface Planner GUI



In contrast with Interface Planner, Tile Interface Planner is specifically for placing component IP on the F-tile, and requires you to run an initial Design Analysis stage before you define a legal tile floorplan for all component IP targeting device tiles.

- Tile Interface Planner Tool Flow on page 55
- Tile Interface Planner GUI Reference on page 69

2.2.1. Tile Interface Planner Terminology

Tile Interface Planner refers to the following terminology:

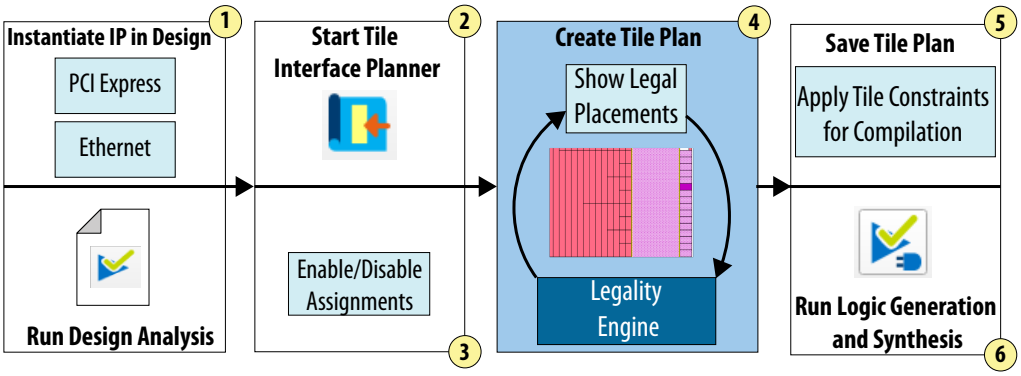
Table 22. Tile Interface Planner Terminology

Term	Description
Dynamic Reconfiguration	Intel FPGA IP technology that allows you to modify some features of a supported multi-rate Intel FPGA IP interface in real time while the FPGA remains in operation. For example, you can dynamically reconfigure the settings in the F-tile CPRI PHY Multi-Rate Intel FPGA IP to run your design at different data rates and features for different IP "profiles."
Floorplan	The layout of physical resources on the device. Creating a design floorplan, or floorplanning, is the process of mapping the logical design hierarchy to physical regions in the device. Tile Interface Planner is a tile IP floorplanning tool.
IP building block	Intel FPGA IP cores are comprised of building blocks that combine to provide all functionality of the IP. The Design Tree view in Tile Interface Planner displays each IP's building blocks. Building blocks can be movable, fixed, or always movable types. • **Movable building blocks**—building blocks are initially movable. Movable building blocks can be re-placed by the legality engine in potentially one of several legal locations to accommodate other building blocks. You can convert a movable building block to fixed by specifying a fixed placement location. The legality engine cannot change the placement of fixed building blocks. Movable building block placements appear in italic text in the Design Tree view. Movable building blocks have a dithered fill in **Chip View**. • **Fixed building blocks**—building blocks that you place in a fixed, legal location that the legality engine cannot change. You can convert a movable building block to fixed, and a fixed building block to movable. Fixed building block placements appear in plain text in the Design Tree view. Fixed building blocks have a solid fill In the **Chip View**. • **Always movable building blocks**—building blocks that are always movable by the legality engine and cannot be fixed. These building blocks must remain movable to prevent inadvertent conflicting constraints. Always movable building blocks appear in gray italic text in the Design Tree view.
Quartus Prime Settings File (.qsf)	Quartus Prime software file that preserves project settings and assignments, including the placement of fixed IP building blocks and fixed tile assignments that you specify in Tile Interface Planner.
JSON file	Quartus Prime software internal file that preserves the most recent placement from the Logic Generation stage of the Compiler. You can load this placement when you click **Update Assignments** if you want the starting point for planning to include the last Logic Generation assignments.
Legal location	Tile Interface Planner legality engine identifies the legal locations in the tile floorplan for placement of the IP or building block that you select in the Design Tree.
Legality engine	Tile Interface Planner function that generates valid legal locations for tile placement, and places movable and always movable building blocks in the tile plan.
Placed design element	IP or building block that you or the legality engine has assigned to a fixed or movable legal location.
Support-Logic Generation stage	A Compiler stage, preceding Analysis & Synthesis, that includes the Design Analysis and Logic Generation sub-stages. This stage is only present when targeting F-tile. • **Design Analysis stage**—A Compiler stage, preceding Analysis & Synthesis, that elaborates the design RTL to extract design information about component IP targeting F-tile. You must run this stage before running Tile Interface Planner. This stage is not present for other FPGA device families or for designs without required IP. • **Logic Generation stage**—A Compiler stage, following Design Analysis, that uses your Tile Interface Plan to generate logic for synthesis and implementation of your tile configuration plan. You must run Logic Generation after Design Analysis before you can synthesize your tile plan.
Tile plan	One or more fixed placements that you define and save in Tile Interface Planner using the (`.qsf`).
Unplaced design element	IP or building blocks that are unassigned to a fixed or movable legal location.

2.2.2. Tile Interface Planner Tool Flow

The Tile Interface Planner user interface guides you through each step in the tile interface planning process.

Figure 33. Tile Interface Planner Tool Flow



- Step 1: Instantiate IP and Run Design Analysis on page 55—Tile Interface Planner first requires a design with component IP, targeting the Agilex 7 FPGA with F-tile. After initial design setup, you run Design Analysis to elaborate the component IP in the design.
- Step 2: Initialize Tile Interface Planner on page 56—launch Tile Interface Planner, component IP and existing assignment data loads, and the legality engine initializes.
- Step 3: Update Plan with Project Assignments on page 57—enable or disable any existing placement assignments and optionally load placement data from previous planning sessions for the current tile planning session.
- Step 4: Create a Tile Plan on page 58—use the **Plan** tab to locate the potential legal locations for each unplaced component IP, place the IP in the tile location, and verify that the placement is legal in real-time to ensure correlation in the final implementation.
- Step 5: Save Tile Plan Assignments on page 62—save the tile IP plan assignments to the project for design compilation.
- Step 6: Run Logic Generation and Design Synthesis on page 63—run the Compiler Logic Generation stage to implement your tile plan and continue synthesis and the remaining design compilation stages.

2.2.2.1. Step 1: Instantiate IP and Run Design Analysis

Tile Interface Planner requires an Quartus Prime project that includes component IP targeting the Agilex 7 FPGA with F-tile.

After instantiating the component IP in a top-level project design file (for example, `top.v`), you run the Design Analysis compilation stage to elaborate the design RTL to extract component IP and target device information. Upon launch, Tile Interface Planner initializes and displays this component IP information in the Design Tree view.

Follow these steps to instantiate IP and run Design Analysis:

1. Open or create an Quartus Prime project that includes component IP targeting F-tile:

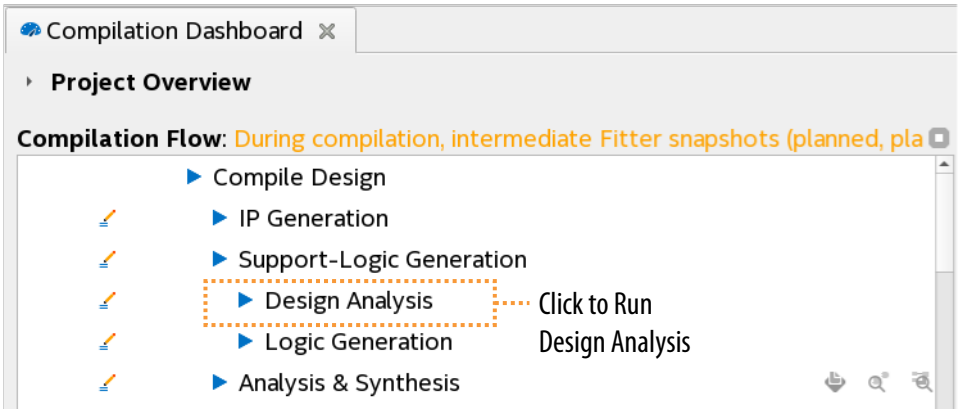
- Create a new project, add design files, and specify the target Agilex 7 FPGA by clicking **File ➤ New Project Wizard**.

  Or

- Parameterize and instantiate component IP with IP Catalog (**View ➤ IP Catalog**) or Platform Designer (**Tools ➤ Platform Designer**).

2. To run the Design Analysis stage of the Compiler, double-click **Design Analysis** on the Compilation Dashboard (**Processing ➤ Compilation Dashboard**).

Figure 34. Design Analysis Stage in Compilation Dashboard

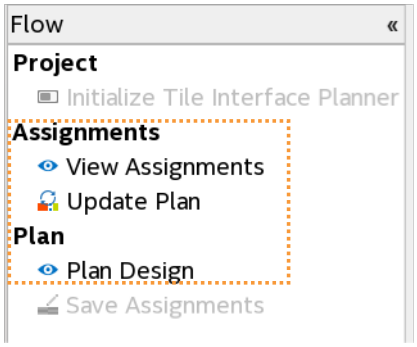


3. Initialize Tile Interface Planner, as Step 2: Initialize Tile Interface Planner on page 56 describes.

2.2.2.2. Step 2: Initialize Tile Interface Planner

When you launch Tile Interface Planner, the tool initializes the placement legality engine and loads component IP and target device data extracted by Design Analysis. Tile Interface Planner then displays the component IP information in the Design Tree view and enables the **Flow** control.

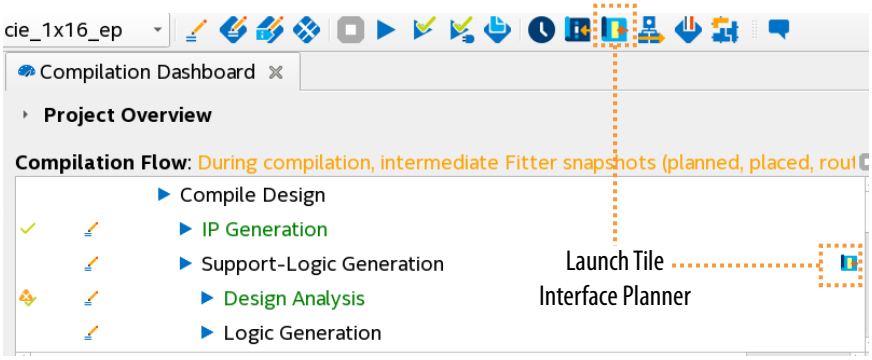
Figure 35. Tile Interface Planner Flow Control



To initialize Tile Interface Planner:

1. Run the Design Analysis stage of the Compiler, as Step 1: Instantiate IP and Run Design Analysis on page 55 describes.
2. When Design Analysis is complete, launch Tile Interface Planner by clicking the Tile Interface Planner icon in the Compilation Dashboard, the main toolbar, or by clicking **Tools ➤ Tile Interface Planner**.

Figure 36. Launching Tile Interface Planner



Tile Interface Planner launches and initializes with the legality engine and the component IP and target device data that Design Analysis extracts.

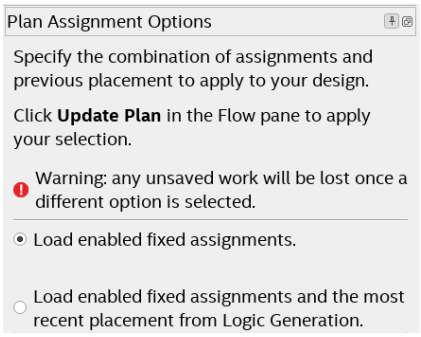
3. Load existing project assignments, as Step 3: Update Plan with Project Assignments on page 57 describes.

2.2.2.3. Step 3: Update Plan with Project Assignments

You can determine which fixed assignments to load from the project settings `.qsf`, and optionally load the latest placement from Logic Generation. The enabled assignments become the starting point for the tile plan. Follow these steps to update the plan with existing assignments:

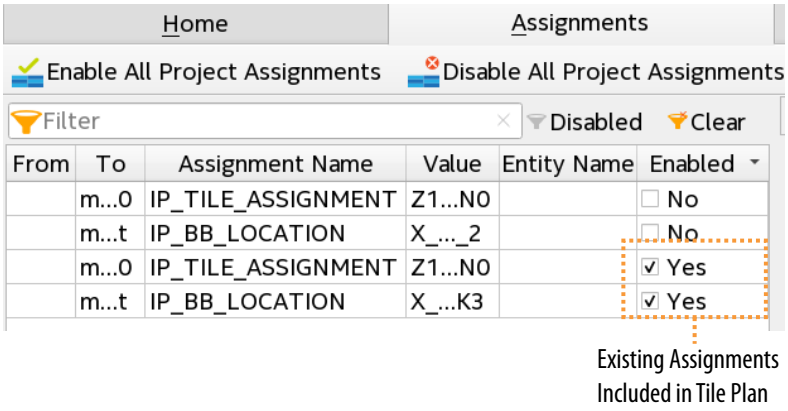
1. On the **Flow** control, click **View Assignments**.

Figure 37. Plan Assignment Options



2. On the **Assignments** tab, select the assignment types to load for the current planning session, as Assignments Tab Controls on page 70 describes.
3. On the **Assignments** tab, enable or disable assignments to resolve any conflicts or experiment with alternative placements. Filter the list of assignments by assignment name or status.

Figure 38. Enable or Disable Existing Assignments for Current Planning Session

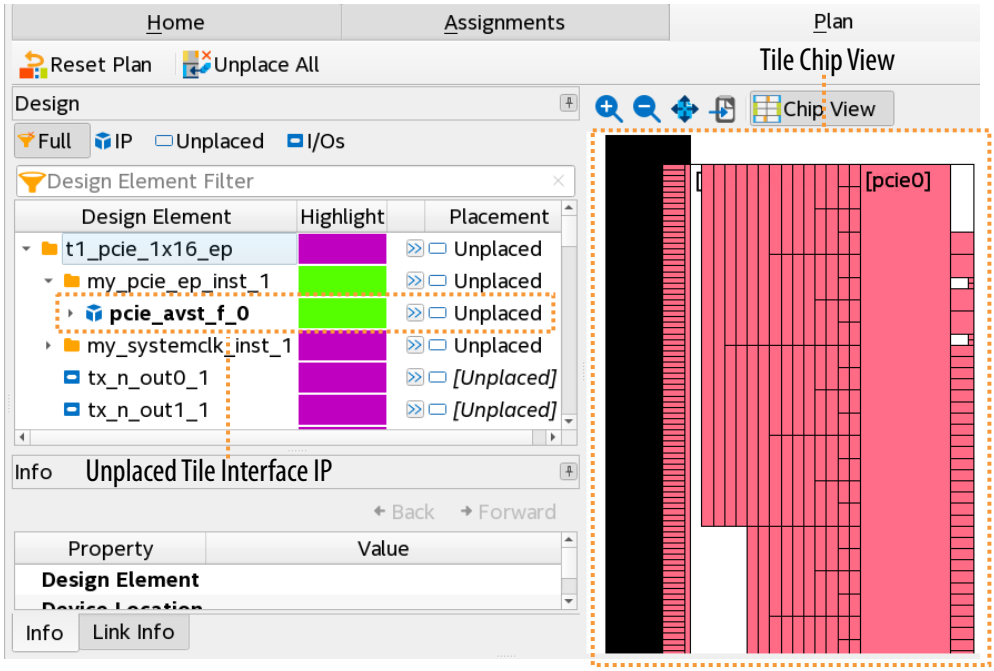


4. When assignment selections are complete, or if you have no existing assignments, click **Update Plan** on the **Flow** control to apply the enabled project assignments to your current tile interface plan.
5. Place IP components and building blocks on the **Plan** tab, as Step 4: Create a Tile Plan on page 58 describes.

2.2.2.4. Step 4: Create a Tile Plan

Click **Plan Design** on the **Flow** control to interactively place component IP in legal locations on device tiles. The **Plan** tab displays a hierarchical list of your project component IP design elements, alongside a graphical abstraction of the target device tile architecture. Place IP (and IP building blocks) in legal tile locations within the graphical tile floorplan.

Figure 39. Tile Interface Planner Design Elements and Chip View



Recommended Two-Stage Tile IP Placement

Handle IP tile placement in two stages for the most efficiency:

Table 23. Two-Stage Tile IP Placement

Tile IP Placement	Description
Stage 1	• Place all of the IPs targeting the same tile to ensure that all IP can be placed within the tile, as Placing IP Components on page 59 describes. • Fill each tile with the desired IP, before refining any IP building block placement for any specific placement requirements you may have for a particular building block.
Stage 2	• Review the placement of IP building blocks. • Refine any building block placement to meet any specific placement requirements, as Constraining IP Building Blocks on page 61 describes.

Note: Changes made in Tile Interface Planner do not apply to your Quartus Prime project until you apply the generated tile interface plan constraints to your project, as Step 5: Save Tile Plan Assignments on page 62 describes.

Placing IP Components on page 59

Constraining IP Building Blocks on page 61

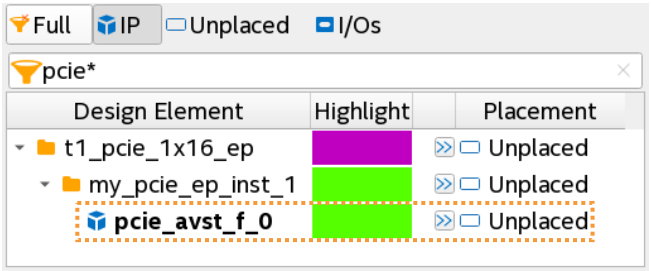
2.2.2.4.1. Placing IP Components

To place IP components and create a tile plan, follow these steps:

Note: For placing multi-rate IP components, refer to Constraining Dynamic Reconfiguration IP on page 64.

1. Update the plan with existing assignments, as Step 3: Update Plan with Project Assignments on page 57 describes.
2. In Tile Interface Planner, click the **Plan** tab. Tile Interface Planner displays the **Design Element** hierarchy, alongside a graphical representation of the tile chip or package view.
3. In the **Design Element** list, locate the tile interface IP that you want to place. You can search and filter the list by name, IP, placement status, I/Os, and other criteria.

Figure 40. Unplaced PCIe Tile Interface IP in Plan Tab



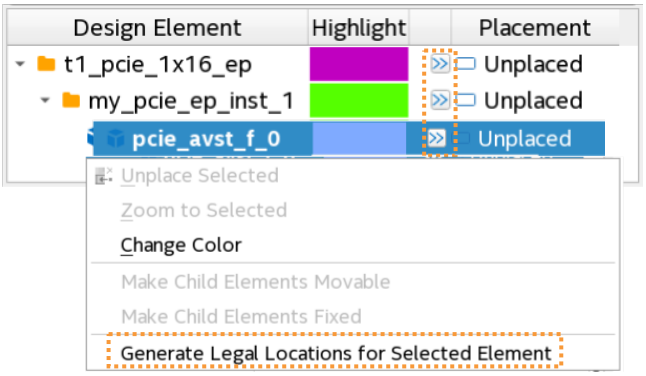
4. To customize design element color coding, double-click a color in the **Highlight** column to specify a new color.
5. Use any of the following methods to locate legal tile placements for component IP:

- In the **Design Element** list, right-click the tile interface IP that you want to place, and then click **Generate Legal Locations for Selected Element**.

  Note: You can select multiple IP targeting the same tile to generate legal locations for all IP at once.

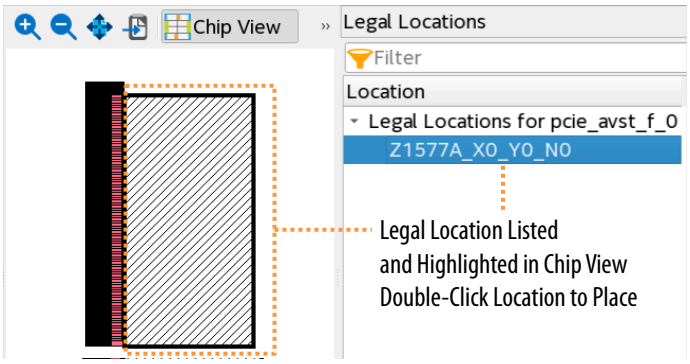
- Click the  button next to the **Design Element** to display a list of **Legal Locations**.

Figure 41. Listing Legal Locations for Tile Placement



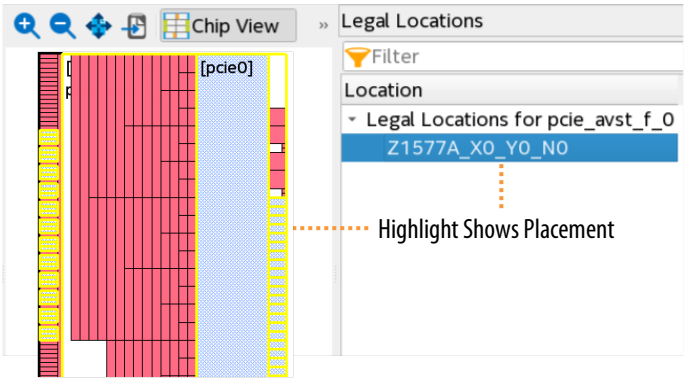
6. In **Legal Locations**, click any location in the list to highlight the location in the floorplan.

Figure 42. Highlighting Legal Locations for Tile Placement



7. Double-click any location in **Legal Locations** to place the element in a legal location. Tile Interface Planner places the IP in the legal location on the device tile, as indicated by color highlighting in the Chip View. When listing legal locations for multiple IPs at once, you can also place multiple IPs at once.

Figure 43. IP Placed In Legal Tile Location

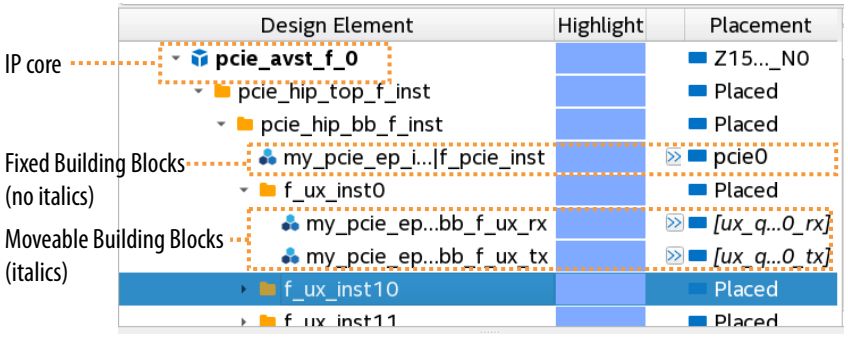


2.2.2.4.2. Constraining IP Building Blocks

Intel FPGA IP cores are comprised of building blocks that combine to provide all functionality of the IP. The Design Tree view displays each IP building block hierarchy.

When you place component IP, Tile Interface Planner also places the corresponding IP building blocks in the tile. Each building block has a movable, fixed, or always movable state. You can review the building block placement and determine whether to refine any building block placement, or allow the legality engine to determine the best building block placement.

Figure 44. IP Core Comprised of IP Building Blocks in Design Tree View



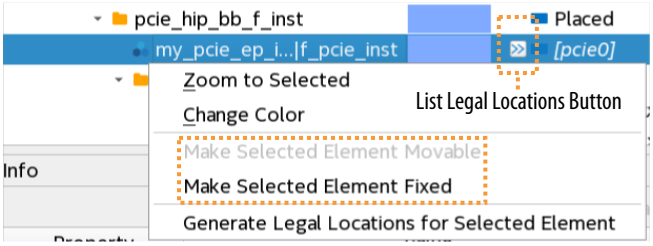
The placement column indicates the movable, fixed, and always-movable state of the design element. In general, use movable building block placement to allow placement flexibility. Only apply fixed building blocks if a specific building block placement is essential.

Follow these steps to constrain or relax the placement of IP building blocks:

1. Place all IP on the tile, as Placing IP Components on page 59 describes.
2. To the left of the **Placement** column, click the List Legal Locations button to display all legal locations for a building block.
3. To constrain building blocks to specific placement:

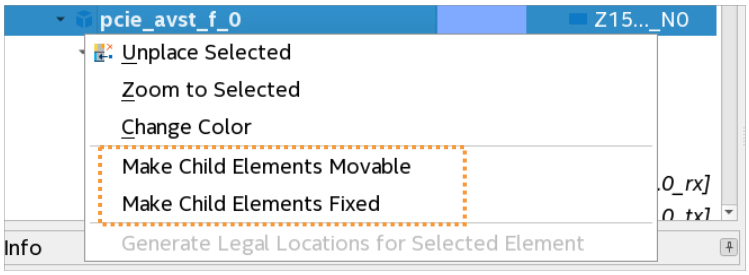
- To fix a movable building block, right-click one or more movable building blocks and click **Make Selected Element Fixed**. The building block is fixed and does not move to accommodate other components.
- To fix an IP component's building blocks, right-click the IP component and click **Make Child Elements Fixed**. The IP and child building blocks are fixed and do not move to accommodate other components.

Figure 45. Make Selected Element Fixed



4. To remove specific building block placement constraints from fixed building blocks:
   - To make fixed building blocks movable, right-click one or more fixed building blocks and click **Make Selected Element Movable**. The building block can move to accommodate other components.
   - To make all of an IP component's fixed building blocks movable, right-click the IP component and click **Make Child Elements Movable**. All child building blocks can move automatically to accommodate other components.

Figure 46. Make Child Elements Movable



5. When all tile IP placement is complete, save the tile plan, as Step 5: Save Tile Plan Assignments on page 62 describes.

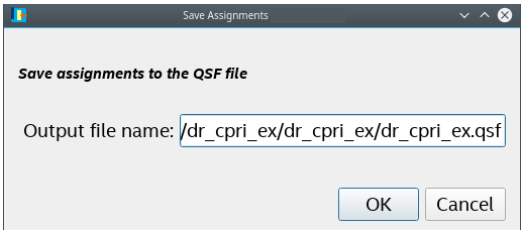
2.2.2.5. Step 5: Save Tile Plan Assignments

Once you have placed all IP components, and fixed any movable building blocks that you want to constrain, you save the constraints in Tile Interface Planner. Tile Interface Planner saves the fixed tile constraints to the project `.qsf`. The Compiler reads the `.qsf` assignments during the Logic Generation stage.

To save the tile plan assignments, follow these steps:

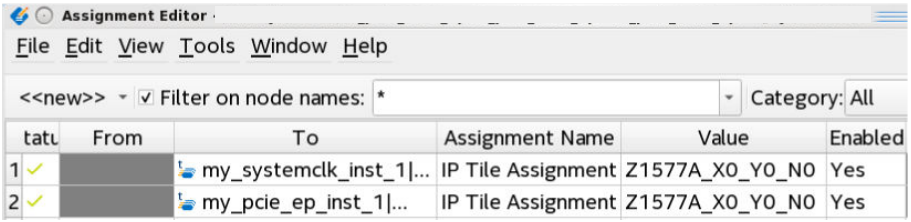
1. Review and consider constraining any IP building blocks. To guarantee placement to exact locations, you must fix the IP building blocks connected to the IP pins to preserve those constraints in the `.qsf`, as Constraining IP Building Blocks on page 61 describes.

Figure 47. Save Assignments from Tile Interface Planner



2. In Tile Interface Planner, click **Save Assignments** on the Flow control, and then click **OK**.
3. Close Tile Interface Planner and return to the Quartus Prime GUI. The tile IP assignments are visible in the Assignment Editor (**Assignments ➤ Assignment Editor**) and in the `.qsf` file.

Figure 48. Tile IP Assignments in Assignment Editor



4. Run the Logic Generation stage, as Step 6: Run Logic Generation and Design Synthesis on page 63 describes.

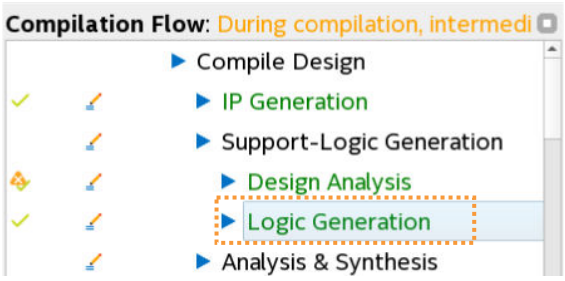
2.2.2.6. Step 6: Run Logic Generation and Design Synthesis

After saving your tile plan assignments, run the Compiler's Logic Generation stage to implement your tile plan and run the remaining design compilation stages.

To run Logic Generation and design synthesis, follow these steps:

1. Save your tile interface plan, as Step 5: Save Tile Plan Assignments on page 62 describes.
2. In the Quartus Prime software, double-click the **Logic Generation** stage in the Compilation Dashboard. Logic Generation reads the tile plan assignments from the `.qsf`.

Figure 49. Run Logic Generation Stage Before Synthesis



3. Once Logic Generation completes, double-click **Analysis & Synthesis** on the dashboard.
4. Once Analysis & Synthesis complete, run the other remaining downstream stages in the compilation flow when ready.

2.2.3. Constraining Dynamic Reconfiguration IP

Tile Interface Planner and Tile Assignment Editor provide support for constraining IP instances that are part of a dynamic reconfiguration group.

Dynamic reconfiguration allows you to modify some features of an Intel FPGA IP interface in real time, while the FPGA remains in continuous operation. This dynamic reconfiguration capability allows you to change your design to run at different data rates, and with different features, for different IP "profiles."

When you generate a dynamically reconfigurable IP instance, the IP includes a `.mif` file that specifies the base and secondary profiles that you define. Each profile contains the delta programming sequences for the dynamic reconfiguration of the IP in a linked-list format.

Related Information

- F-Tile Dynamic Reconfiguration Suite Intel FPGA IP User Guide
- F-Tile Dynamic Reconfiguration Design Example User Guide
- F-tile CPRI PHY IP User Guide
- F-tile CPRI Multi-rate PHY Dynamic Reconfiguration Design Example User Guide

2.2.3.1. Defining a Dynamic Reconfiguration Group

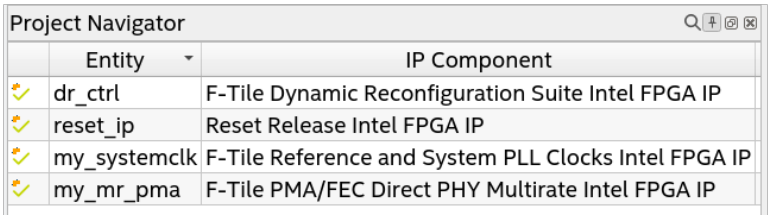
You can define a dynamic reconfiguration group to declare the tree structure that defines how dynamic reconfiguration operates for all of the IP instances that you assign to the group. The reconfiguration group can be either **Exclusive** (only one DR group IP instance is active at any time), or **Inclusive** (any combination of DR group IP instances can be active at any time).

You define the members and properties of the group, and then assign the tile placement of the group members. The Tile Assignment Editor allows you to easily create a dynamic reconfiguration group scheme in a unified GUI. You can then visualize and interactively place the dynamic reconfiguration group in the Tile Interface Planner floorplan.

When you place or unplace any member IPs of a multirate DR group, Tile Interface Planner also appropriately places or unplaces the children and sibling IPs in the DR group automatically. You can then expand and refine the placement of member IPs individually.

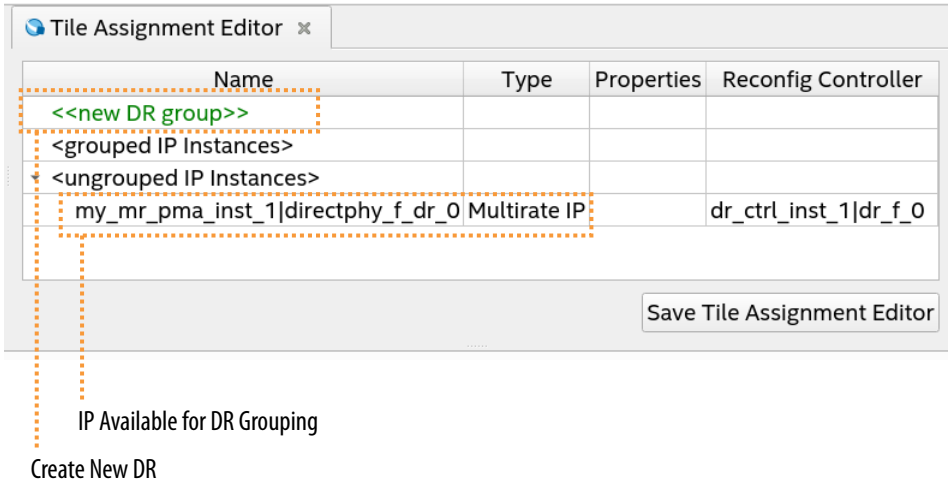
1. Define and add to your project all of the dynamically reconfigurable IP instances to be included in the dynamic reconfiguration group or groups, as Step 1: Instantiate IP and Run Design Analysis on page 55 describes. Connect the IP instances in a manner that is compatible with the dynamic reconfiguration scheme that you want to create.

Figure 50. Dynamically Reconfigurable IP In Project Navigator



2. To run the Design Analysis stage of the Compiler, double-click **Design Analysis** on the Compilation Dashboard. Design Analysis discovers your project's dynamically reconfigurable IP instance information for use in Tile Assignment Editor and Tile Interface Planner.
3. To open Tile Assignment Editor, click **Assignments ➤ Tile Assignment Editor**. Tile Assignment Editor displays the **Name**, **Type**, and other **Properties** of the ungrouped and any grouped IP instances that are valid for dynamic reconfiguration grouping within your project.

Figure 51. Tile Assignment Editor GUI

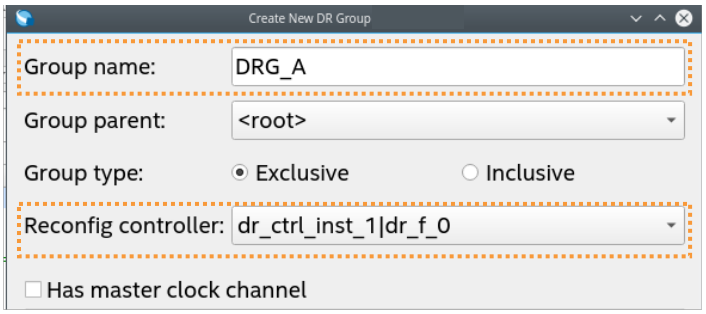


4. To define a new DR group, double-click **<<new DR group>>** in the Tile Assignment Editor **Name** column. The **Create New DR Group** dialog box appears.
5. In the **Group name** box, specify a name for the DR group, along with the following options:

Table 24. Create New DR Group Options

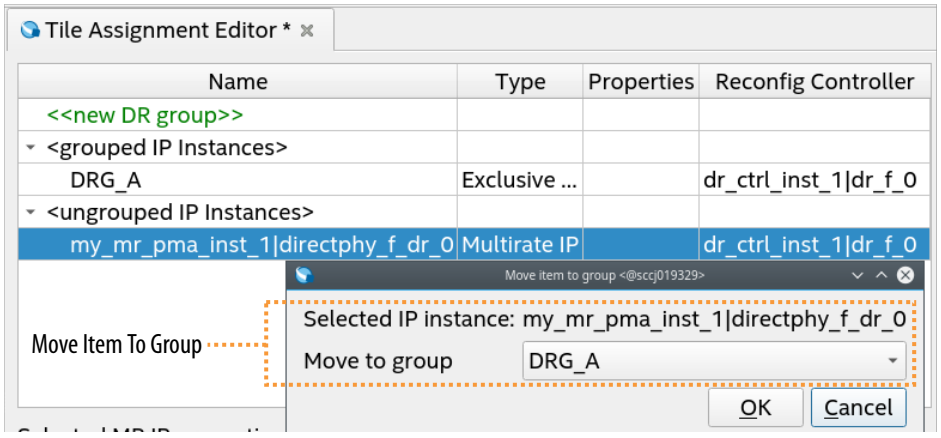
Option	Description
Group parent	Specifies that the DR group is the **<root>** (parent), or allows you to select an existing DR group as the parent to newly created dynamic reconfiguration group..
Group type	Specifies that the group is **Exclusive** (only one IP instance is active at any given time) or **Inclusive** (Any combination of IP instances may be active at any given time). Default value is **Exclusive**.
Reconfig controller	Allows you to select the appropriate reconfiguration controller for the group. You may have multiple reconfiguration controllers in your project.(2)
Has master clock channel	By default, the group inherits this value from the Specifies **On** if your IP group has a master clock channel, or **Off** if your IP group does not have a master clock channel. When **On**, you can also specify the **Building block instance name** and the **Clock port** of the master clock channel.

Figure 52. Create New DR Group Dialog Box



6. To add IP instances to a DR group, right-click the DR instance name and then click **Move IP Instance**. In **Move to group**, select the DR group to add the IP instance and click **OK**.

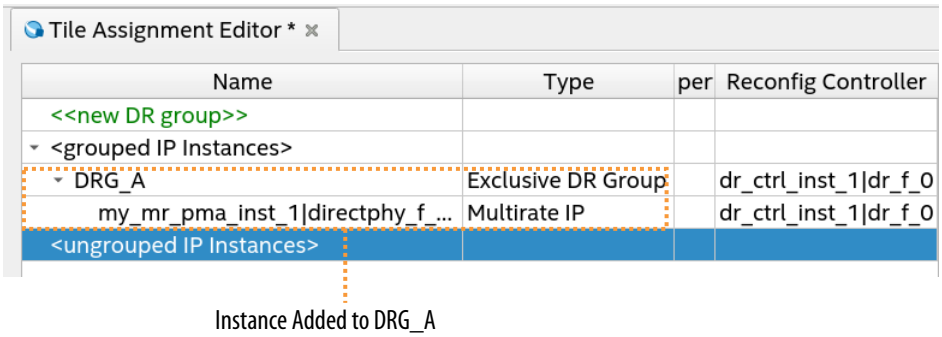
Figure 53. Move Item To Group



7. To specify the startup instance, right click any instance and then click **Toggle as startup IP instance**.

(2) If you do not specify this option child DR groups inherit this value from the parent.

Figure 54. Moved Item Within DR Group



8. Once you are done defining the DR groups for the project, click **Save Tile Assignment Editor** to save the tile assignments to the project `.qsf`.

Figure 55. DR Group Assignments in QSF

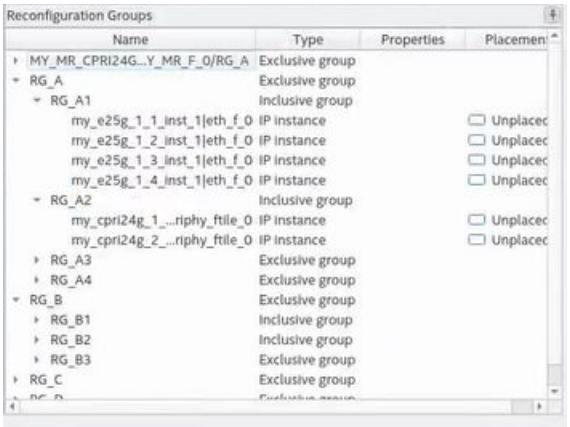
```
set_global_assignment -name IP_RECONFIG_GROUP_TYPE "DRG_A:EXCLUSIVE" -entity dr_mr_pma_1x50g_2_2x25g_1
set_global_assignment -name IP_RECONFIG_GROUP_PARENT "DRG_A:MY_MR_PMA_INST_1|DIRECTPHY_F_DR_0/RG_A_E" -er
set_instance_assignment -name IP_RECONFIG_GROUP_STARTUP_INSTANCE OFF -to my_mr_pma_inst_1|directphy_f_dr_
set_instance_assignment -name IP_RECONFIG_GROUP_MASTER_CLOCK_CHANNEL OFF -to my_mr_pma_inst_1|directphy_
```

9. Use the DR groups to place DR groups in Tile Interface Planner, as Assigning Dynamic Reconfiguration Group Placement on page 67 describes.

2.2.3.2. Assigning Dynamic Reconfiguration Group Placement

Once you load a design that includes dynamic reconfiguration groups, Tile Interface Planner displays the tile location of the dynamically reconfigurable IP instances and related building blocks. You can dynamically assign the tile location of IP instances within each dynamic reconfiguration group, and place other IP components in relation to the dynamic reconfiguration group. When placing other components, Tile Interface Planner takes all of the dynamic reconfiguration group placements into account.

Figure 56. Reconfiguration Groups Tree View

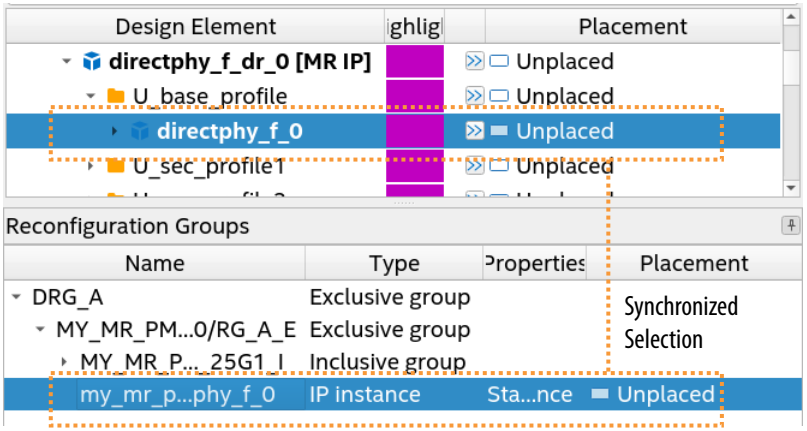


The **Reconfiguration Groups** view shows the members and placement of the dynamic reconfiguration groups that you create. Use the **Reconfiguration Groups** view to select the dynamic reconfiguration group that you want to assign. You can only modify the DR groupings in Tile Assignment Editor, not in Tile Interface Planner.

To assign DR groups to tile locations, follow these steps:

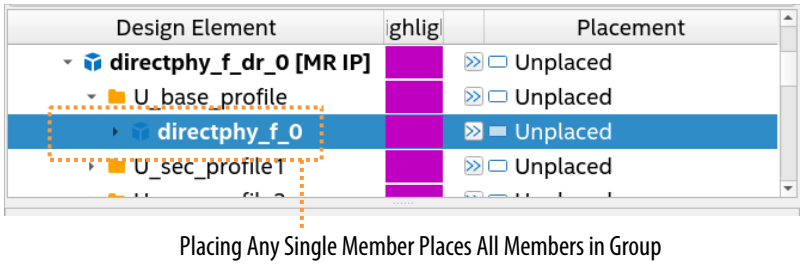
1. Define one or more dynamic reconfiguration groups, as Defining a Dynamic Reconfiguration Group on page 64 describes.
2. To run Logic Generation, double-click **Logic Generation** on the Compilation Dashboard.
3. To start Tile Interface Planner, click the **Tile Interface Planner** icon on Compilation Dashboard, as Step 2: Initialize Tile Interface Planner on page 56 describes.
4. In Tile Interface Planner, click the **Plan** tab. Tile Interface Planner displays the dynamic reconfiguration IP profiles in the **Design Element** hierarchy.
5. Select an instance in the **Reconfiguration Groups** tree view, the selection synchronizes with the selection in the **Design Elements** lists.

Figure 57. Reconfiguration Groups and Design Elements Selection Synchronization



6. Click the >> button next to the **Design Element** to display a list of **Legal Locations** for the selected DR group.
7. Double-click any location in **Legal Locations** to place the element in a legal location. Tile Interface Planner places the IP in the legal location on the device tile.

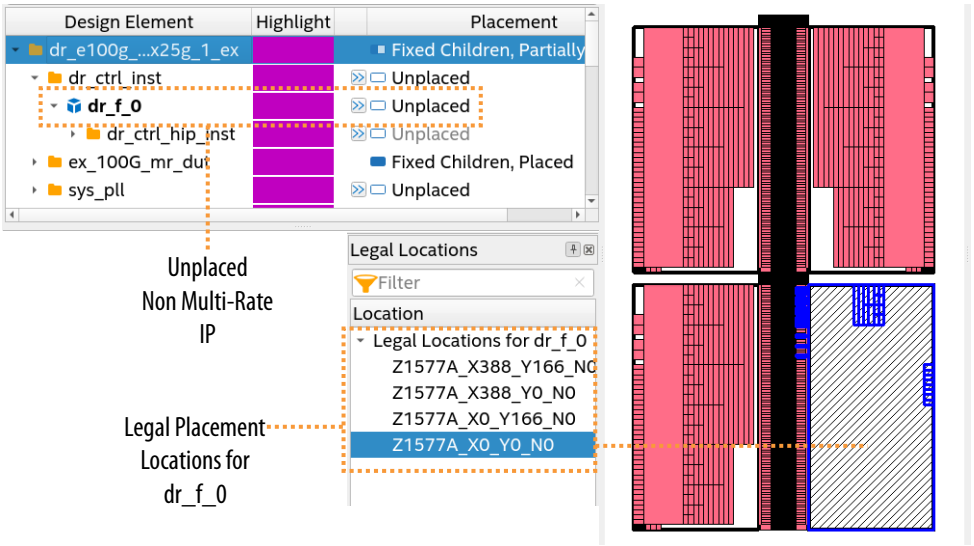
Figure 58. Placing One Member Automatically Places All Others in DR Group



Note: When you place or unplace any member IPs of a multirate DR group, Tile Interface Planner also appropriately places or unplaces the children and sibling IPs in the DR group automatically. You can then expand and refine the placement of member IPs individually.

8. Place other IP components in relation to the DR group location, as Placing IP Components on page 59 describes. Tile Interface Planner takes the dynamic reconfiguration group placement into account when you place other IP components.

Figure 59. Placing Other IP Components



2.2.4. Tile Interface Planner GUI Reference

The Tile Interface Planner user interface includes the following controls for floorplanning your design.

2.2.4.1. Flow Controls

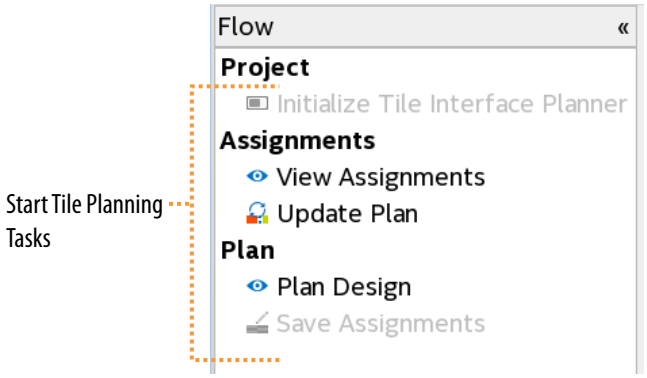
The **Flow** control panel provides immediate access to common Tile Interface Planner commands from anywhere within Tile Interface Planner.

Table 25. Flow Controls

Command	Description
Initialize Tile Interface Planner	Launches the placement legality engine and loads the component IP and target device data that Design Analysis extracts.
View Assignments	Opens the **Assignments** tab, which allows you to review and enable or disable any existing placement assignments for the current planning session.
Update Plan	Optionally applies a previous tile planning session fixed placement assignments from the `.qsf`, and movable placements from a `.json` to the current tile interface plan.
Plan Design	Opens the **Plan** tab for placing component IP in the tile interface plan.
Save Assignments	Opens the **Save Assignments** dialog box for saving the fixed tile constraints to the project `.qsf` and the movable building block constraints to a `.json` file.

The **Flow** controls appear in order of a typical tile interface planning flow.

Figure 60. Tile Interface Planner Flow Control



2.2.4.2. Home Tab

The Tile Interface Planner **Home** tab is the default tab that displays the **Flow** control and an introductory interface planning infographic. There are no controls specific to the Tile Interface Planner **Home** tab.

2.2.4.3. Assignments Tab Controls

The **Assignments** tab allows you to review and enable or disable any existing placement assignments for the current tile interface planning session. Click **View Assignments** on the **Flow** control to display the **Assignments** tab.

You can enable or disable any placement assignments that Design Analysis finds. After you are satisfied with the status of all project assignments, click **Update Plan** on the **Flow** control to update your tile interface plan with the enabled project assignments.

Table 26. Assignments Tab Controls

Command	Description
Filter field	Supports creation of wildcard expressions for assignment targets. **Enabled** and **Disabled** buttons filter only enabled or disabled assignments in the list.
Enable All Project Assignments	Enables all existing assignments for the current tile interface planning session. These assignments then become the starting point for your tile plan.
Disable All Project Assignments	Disables all existing assignments for the current tile interface planning session.
Clear	Clears any filter from the **Assignments** list.
Plan Assignment Options	The following options are mutually exclusive: • **Load enabled fixed assignments**—loads fixed assignments from `.qsf`. These assignments then become the starting point for your tile plan. • **Load enabled fixed assignments and the most recent placement from Logic Generation**—loads fixed assignments from `.qsf` and the most recent placement data from the last Logic Generation stage of the Compiler.

2.2.4.4. Plan Tab Controls

The **Plan** tab contains the Design Tree and tile visualization panes. The Design Tree lists design elements for placement. The tile visualization pane displays a graphical view of the target device tile to help you visualize the appropriate legal locations for placement of component IP.

Click **Plan Design** on the **Flow** control to display the **Plan** tab.

Table 27. Plan Tab Controls

Command	Description
	Lists legal locations for placement in the Legal Locations pane.
Unplace All	Unplaces all placed design elements in the interface plan.
Chip View	Displays the target device at the chip level of detail, showing a representation of the divisions of device resources spread across the device. Zoom in to display chip details.
Package View	Displays the target device package at the package level of detail, showing the I/O pin details of the device package. Zoom in to display package details.
Birdseye View	Displays the target device chip or package view at maximum **Zoom Out**.
Reset Plan	Unplaces all placed design elements and removes applied project assignments from the tile interface plan.
Zoom In and **Zoom Out**	Increases or decreases the magnification of the tile view to show more or less detail.
Fit in Window	Increases or decreases the magnification of the tile view to fit in the current window.

2.2.4.4.1. Design Tree and Filters

The Design Tree View displays a hierarchy of the IP components and building block design elements found during the Design Analysis compilation stage. You can locate the IP and building blocks in the design tree, and then assign the elements to legal locations in the tile floorplan on the **Plan** tab.

The Design Tree view includes these columns:

- **Design Element**—lists all component IP and building blocks that Design Analysis identifies. IP cores and building blocks are distinguished by different icons. Fixed IP building blocks are shown in plain text. Movable IP building blocks are shown in italic text. Always movable building blocks are shown in gray italic text.
- **Highlight**—indicates the color for display of design elements in the tile visualization pane.
- **Placement**—indicates the placement status (placed, unplaced) or placement tile location for design elements.

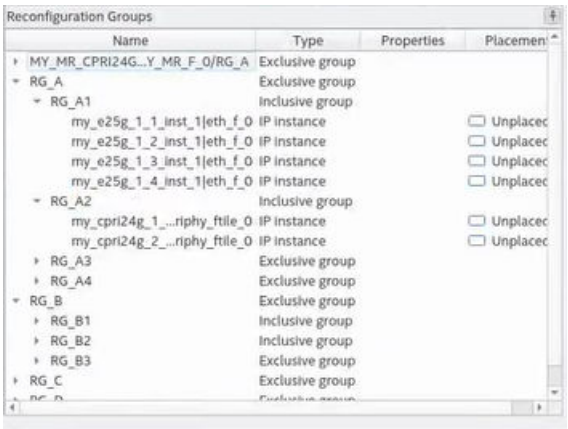
You can type a partial or complete name in the design element filter field to refine the list of elements displayed.

- Click the **Full** icon to show all design elements in the tree.
- Click the **IP** icon to show only IP level hierarchy in the tree.
- Click the **Unplaced** icon to show only unplaced design elements in the tree.
- Click the **I/Os** icon to show only I/O design elements in the tree.

2.2.4.4.2. Reconfiguration Groups View

The **Reconfiguration Groups** view shows all IP instances that are part of a dynamic reconfiguration group, or in a multi-rate IP instance. Use this view to readily display the dynamic reconfiguration group hierarchy in the current design. The **Reconfiguration Groups** view shows you which IP instances in the dynamic reconfiguration group are placed, and identifies the tile placement location.

Figure 61. Reconfiguration Groups View (Multi-Rate IP Design)



2.2.4.4.3. Legal Locations Pane

The Legal Locations pane lists the legal locations for tile placement that the legality engine determines. You can enter a text string in the **Filter** field to limit the list.

- Click any legal location in the list to highlight that location in the tile visualization pane.
- Double-click any legal location in the list to assign placement to that tile location.

2.2.4.5. Tcl Console Window

The Tcl Console Window echoes the commands that you run in the Tile Interface Planner GUI. The Tile Interface Planner GUI operates on top of the Tile Interface Planner API. You can alternatively execute Tile Interface Planner commands in the Tcl console.

Related Information

Video Demo: Using the Tile Interface Planner

2.3. Interface Planning Revision History

This document has the following revision history:

Table 28. Interface Planning Revision History

Document Version	Intel Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout.
2023.10.02	23.3	• Updated screenshot in *Step 1: Instantiate IP and Run Design Analysis* for latest Compilation Dashboard.
2023.04.03	23.1	• Added new *Interface Planner NoC Tool Flow* section describing general use of new Interface Planner features that support the Hard Memory NoC in Agilex 7 M-Series FPGAs. • Added new *Report NoC Performance* topic describing the new NoC Performance Report that Interface Planner can generate. • Updated product family name to "Intel Agilex 7."
2022.09.26	22.3	• Updated *Defining a Dynamic Reconfiguration Group* topic to describe automated placement of children and siblings in DR groups. • Added note and image to *Assigning Dynamic Reconfiguration Group Placement* topic to describe automated placement of children and siblings in DR groups.
2022.06.21	22.2	• Updated *Constraining Dynamic Reconfiguration Group IP* topic for Tile Assignment Editor support. • Added new *Defining a Dynamic Reconfiguration Group* topic describing use of new Tile Assignment Editor. • Updated *Assigning Dynamic Reconfiguration Group Placement* topic for support of interactive placement in Tile Interface Planner.
2022.03.28	22.1	• Added dynamic reconfiguration to *Tile Interface Planner Terminology* topic. • Retitled and updated *Dynamic Reconfiguration Multi-Rate IP Tile Planning* topic for new display of base and secondary dynamic reconfiguration profiles. • Added new Reconfiguration Groups View topic.
2021.10.04	21.3	• Updated *Tile Interface Planner Terminology* JSON file and IP building block definitions. • Updated GUI options in *Step 3: Update Plan Assignments* and *Assignments Tab Controls* topics. • Added "Multi-Rate IP Tile Planning" topic. • Updated "Step 5: Save Tile Plan Assignments" for latest GUI. • Added new video demo link to *Interface Planning* topic. • Added new controls to *Flow Controls* topic. • Revised *Plan Tab Controls* topic for new toolbar controls. • Added application note link to *Step 3: Update Plan with Project Assignments*
2021.06.21	21.2	• Integrated new *Tile Interface Planning* section.
2019.04.01	19.1.0	• Updated "Plan Tab Controls" to describe new color coding controls for I/O banks, differential pin pairs, DQ/DQS pins, and PCIe hard IP pins. • Update screenshots and procedure steps for latest user interface.
2018.05.07	18.0.0	• Initial release in Design Constraints User Guide: Intel Quartus Prime Pro Edition. • Updated *Step 2: Initialize Interface Planner* to remove the requirement to close Intel Quartus Prime. • Updated *Step 4: Plan Periphery Placement* to describe when the **Locate Node** command is disabled.
2017.11.06	17.1.0	• Removed support for the **Clocking** feature for Intel Stratix 10. Intel Stratix 10 clocks must use **Autoplace Selected**. • Renamed BluePrint to Interface Planner. • Renamed chapter from *BluePrint Design Planning* to *Interface Planning.*

continued...

Document Version	Intel Quartus Prime Version	Changes
2016.10.31	16.1.0	• Implemented Intel rebranding.
2016.05.03	16.0.0	• Added Plan Clock Networks topic. • Added Saving and Loading Floorplans topic. • Added Undo/Redo command descriptions. • Added Flow control description. • Added note about panning feature. • Updated all screenshots for latest GUI.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Integration of content into Quartus Prime Handbook. • Added descriptions of new dynamic reports. • Added Package View description. • Added GUI controls reference.
2015.05.04	15.0.0	Second beta release of document on Molson. Added information about the following subjects: • Overview information • Reset Plan command • Legal Assignments list and prompt • Tcl console
2014.12.15	14.1.	First beta release of document on Molson.

3. Managing Device I/O Pins

This chapter describes efficient planning and assignment of I/O pins in your target device. Consider I/O standards, pin placement rules, and your PCB characteristics early in the design phase.

Figure 62. Quartus Prime Pin Planner GUI

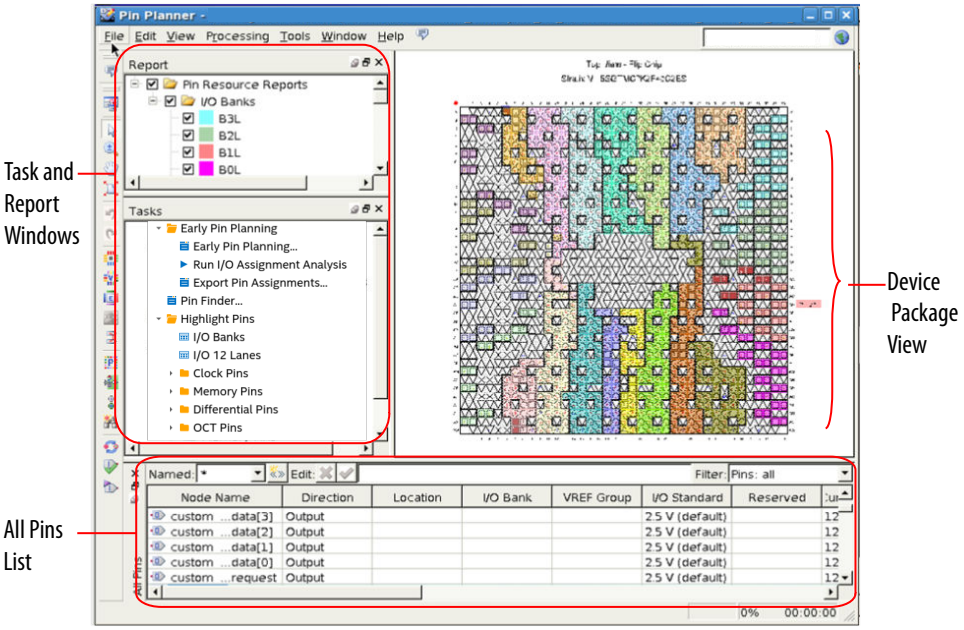


Table 29. Quartus Prime I/O Pin Planning Tools

I/O Planning Task	Click to Access
Plan interfaces and device periphery	**Tools ➤ Interface Planner**
Edit, validate, or export pin assignments	**Assignments ➤ Pin Planner**

For more information about special pin assignment features for the Arria 10 SoC devices, refer to *Instantiating the HPS Component* in the *Arria 10 Hard Processor System Technical Reference Manual*.

Related Information

Instantiating the HPS Component
In *Arria 10 Hard Processor System Technical Reference Manual*

3.1. I/O Planning Overview

On FPGA design, I/O planning includes creating pin-related assignments and validating them against pin placement guidelines. This process ensures a successful fit in your target device. When you plan and assign I/O pins in the initial stages of your project, you design for compatibility with your target device and PCB characteristics. As a result, your design process goes through fewer iterations, and you develop an accurate PCB layout sooner.

You can plan your I/O pins even before defining design files. Assign expected nodes not yet defined in design files, including interface IP core signals, and then generate a top-level file. The top-level file instantiates the next level of design hierarchy and includes interface port information like memory, high-speed I/O, device configuration, and debugging tools.

Assign design elements, I/O standards, interface IP, and other properties to the device I/O pins by name or by dragging to cells. You can then generate a top-level design file for I/O validation.

Use I/O assignment validation to fully analyze I/O pins against VCCIO, VREF, electromigration (current density), Simultaneous Switching Output (SSO), drive strength, I/O standard, PCI_IO clamp diode, and I/O pin direction compatibility rules.

Quartus Prime software provides the Pin Planner tool to view, assign, and validate device I/O pin logic and properties. Alternatively, you can enter I/O assignments in a Tcl script, or directly in HDL code.

3.1.1. Basic I/O Planning Flow

The following steps describe the basic flow for assigning and verifying I/O pin assignments:

1. Click **Assignments ➤ Device** and select a target device that meets your logic, performance, and I/O requirements. Consider and specify I/O standards, voltage and power supply requirements, and available I/O pins.
2. Click **Assignments ➤ Pin Planner**.
3. Assign I/O properties to match your device and PCB characteristics, including assigning logic, I/O standards, output loading, slew rate, and current strength.
4. Click **Run I/O Assignment Analysis** in the **Tasks** pane to validate assignments and generate a synthesized design netlist. Correct any problems reported.
5. Click **Processing ➤ Start Compilation**. During compilation, the Quartus Prime software runs I/O assignment analysis.

3.1.2. Integrating PCB Design Tools

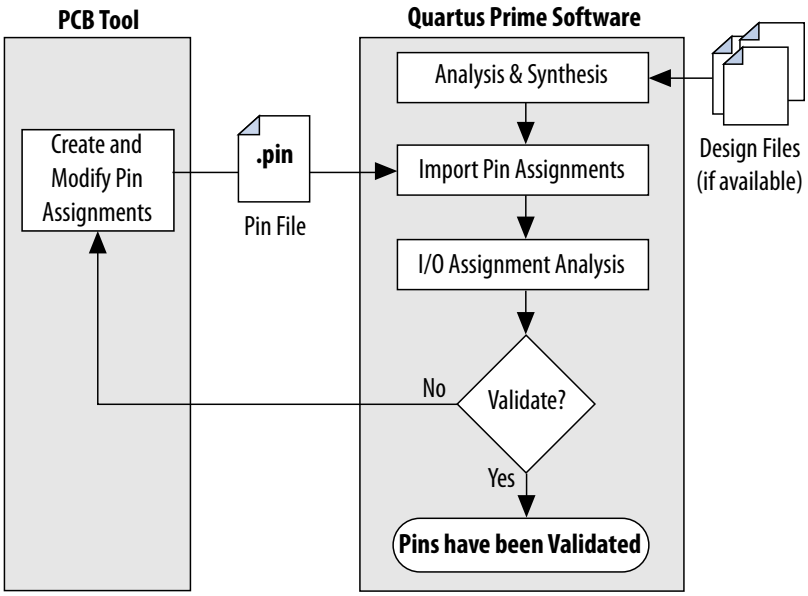
You can integrate PCB design tools into your work flow to map pin assignments to symbols in your system circuit schematics and board layout.

The Quartus Prime software integrates with board layout tools by allowing import and export of pin assignment information in Quartus Prime Settings Files (`.qsf`) or Pin-Out Files (`.pin`).

Table 30. Integrating PCB Design Tools

PCB Tool Integration	Supported PCB Tool
Define and validate I/O assignments in the Pin Planner, and then export the assignments to the PCB tool for validation	Cadence Allegro
Define I/O assignments in your PCB tool, and then import the assignments into the Pin Planner for validation	Cadence Allegro

Figure 63. PCB Tool Integration

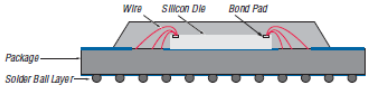
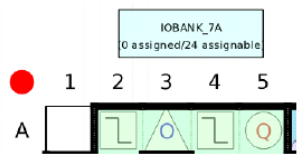
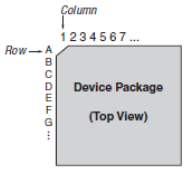
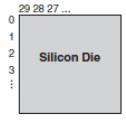
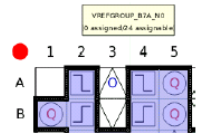


Related Information

Cadence PCB Design Tools Support
In *Quartus Prime Pro Edition User Guide: PCB Design Tools*

3.1.3. Intel FPGA Device and I/O Terminology

The following terms describe Intel FPGA device and I/O structures:

Terms	Description	Diagram
Device Package (BGA example)	Ceramic or plastic heat sink surface mounted with FPGA die and I/O pins or solder balls. In a wire bond BGA example, copper wires connect the bond pads to the solder balls of the package. Click **View > Show > Package Top** or **View > Show > Package Bottom** in Pin Planner	
I/O Bank	I/O pins are grouped in I/O banks for assignment of I/O standards. Each numbered bank has its own voltage source pins, called VCCIO pins, for high I/O performance. The specified VCCIO pin voltage is between 1.5 V and 3.3 V. Each bank supports multiple pins with different I/O standards. All pins in a bank must use the same VCCIO signal. Click **View > Show > I/O Banks** in Pin Planner.	
I/O Pin	A wire lead or small solder ball on the package bottom or periphery. Each pin has an alphanumeric row and column number. I, O, Q, S, X, and Z are never used. The alphabet is repeated and prefixed with the letter A when exceeded. All I/O pins display by default.	
Pad	I/O pins are connected to pads located on the perimeter of the top metal layer of the silicon die. Each pad is numbered with an ID starting at 0, and increments by one in a counterclockwise direction around the device. Click **View > Pad View** in Pin Planner.	
VREF Pin Group	A group of pins including one dedicated VREF pin required by voltage-referenced I/O standards. A VREF group contains a smaller number of pins than an I/O bank. This maintains the signal integrity of the VREF pin. One or more VREF groups exist in an I/O bank. The pins in a VREF group share the same VCCIO and VREF voltages.	

3.2. Assigning I/O Pins

Use the Pin Planner to visualize, modify, and validate I/O assignments in a graphical representation of the target device. You can increase the accuracy of I/O assignment analysis by reserving specific device pins to accommodate undefined but expected I/O.

To assign I/O pins in the Pin Planner, follow these steps:

1. Open an Quartus Prime project, and then click **Assignments ➤ Pin Planner**.
2. Click **Processing ➤ Start Analysis & Elaboration** to elaborate the design and display **All Pins** in the device view.
3. To locate or highlight pins for assignment, click **Pin Finder** or a pin type under **Highlight Pins** in the **Tasks** pane.
4. (Optional) To define a custom group of nodes for assignment, select one or more nodes in the **Groups** or **All Pins** list, and click **Create Group**.

5. Enter assignments of logic, I/O standards, interface IP, and properties for device I/O pins in the **All Pins** spreadsheet, or by dragging into the package view.
6. To assign properties to differential pin pairs, click **Show Differential Pin Pair Connections**. A red connection line appears between positive (p) and negative (n) differential pins.
7. (Optional) To create board trace model assignments:
   a. Right-click an output or bidirectional pin, and click **Board Trace Model**. For differential I/O standards, the board trace model uses a differential pin pair with two symmetrical board trace models.
   b. Specify board trace parameters on the positive end of the differential pin pair. The assignment applies to the corresponding value on the negative end of the differential pin pair.
8. To run a full I/O assignment analysis, click **Run I/O Assignment Analysis**. The Fitter reports analysis results. Only reserved pins are analyzed prior to design synthesis.

3.2.1. Assigning to Exclusive Pin Groups

You can designate groups of pins for exclusive assignment. When you assign pins to an **Exclusive I/O Group**, the Fitter does not place the signals in the same I/O bank with any other exclusive I/O group. For example, if you have a set of signals assigned exclusively to `group_a`, and another set of signals assigned to `group_b`, the Fitter ensures placement of each group in different I/O banks.

3.2.2. Assigning Slew Rate and Drive Strength

You can designate the device pin slew rate and drive strength. These properties affect the pin's outgoing signal integrity. Use either the **Slew Rate** or **Slow Slew Rate** assignment to adjust the drive strength of a pin with the **Current Strength** assignment.

Note: The slew rate and drive strength apply during I/O assignment analysis.

3.2.3. Assigning I/O Banks

Some Intel FPGA devices support assignments to I/O banks. I/O banks are a logical grouping of I/O pins for convenience in making certain types of assignments, such as I/O standard assignments.

When targeting a device family that supports I/O bank assignments, the **I/O Bank** cell value automatically populates in Pin Planner once you select a corresponding pin **Location**. The rows for various I/O banks are color coded for easy visual identification.

Figure 64. Pin Location and I/O Bank Cells in Pin Planner

Node Name	Direction	Location	I/O Bank
ddr4_emif_mem_mem_a[7]	Output	PIN_CE34	2B
ddr4_emif_mem_mem_a[3]	Output	PIN_CE32	2B
pio_0_external_connection_export[3]	Output	PIN_C30	3C
pio_0_external_connection_export[0]	Output	PIN_B31	3C

When you save your Pin Planner constraints, the pin location saves to the project `.qsf` that also saves the I/O bank locations as a comment. Command-line users can use this comment to identify I/O bank locations for the placed pins without launching the Quartus Prime software GUI.

Figure 65. I/O Bank Location Saved As Comment in QSF

```
set_location_assignment PIN_C30 -to pio_0_external_connection_export[3] -comment IOBANK_3C
set_location_assignment PIN_B31 -to pio_0_external_connection_export[0] -comment IOBANK_3C
set_location_assignment PIN_CE32 -to ddr4_emif_mem_mem_a[3] -comment IOBANK_2B
set_location_assignment PIN_CE34 -to ddr4_emif_mem_mem_a[7] -comment IOBANK_2B
```

3.2.4. Changing Pin Planner Highlight Colors

The Pin Planner **Task** window provides one-click access to execute common pin planning tasks. After clicking a pin planning task, you can view and highlight the results in the **Report** window by selecting or deselecting I/O types.

You can highlight the various pin types in the current device view for easy visualization and assignment to specific types of pins. You can optionally customize the highlight color for each item in the report window to suit your preferences.

To change the default color for any item in the Pin Planner Report panel, follow these steps:

1. Right-click on any item in the **Report** window, and then click **Change Color**.
2. From the color pallet, specify the highlight color for the item.
3. Click **OK**. The highlight color changes for the selected items in Pin Planner.

Figure 66. 2A I/O Bank Green Before Highlight Color Change

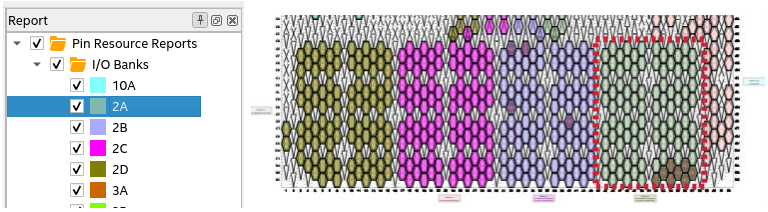
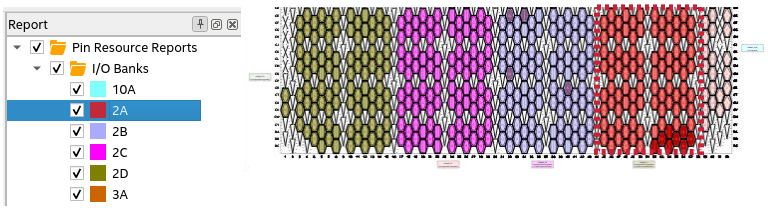


Figure 67. 2A I/O Bank Red After Highlight Color Change



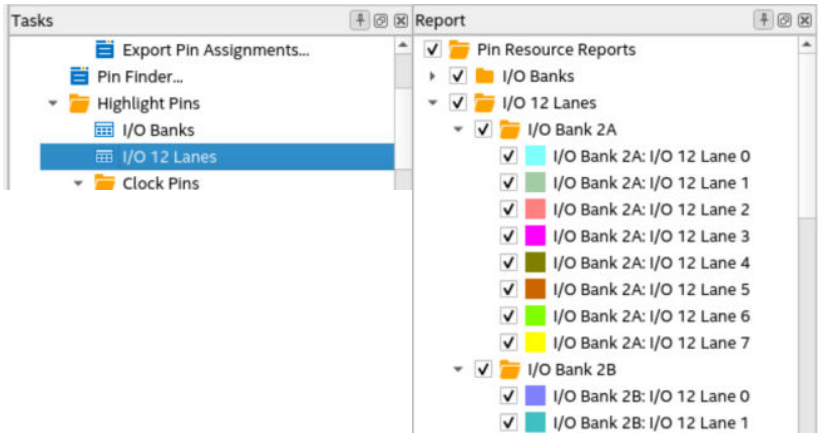
3.2.5. Showing I/O Lanes

You can use the **Report** window **I/O 12 Lanes** command to generate a report that displays all I/O lanes in an I/O bank.

You can highlight the various I/O lanes and change color coding for easy visualization and assignment.

Note: Alternatively, you can generate this report by clicking **View ➤ Show ➤ Show I/O 12 Lanes** in Pin Planner.

Figure 68. Showing I/O Lanes in Pin Planner



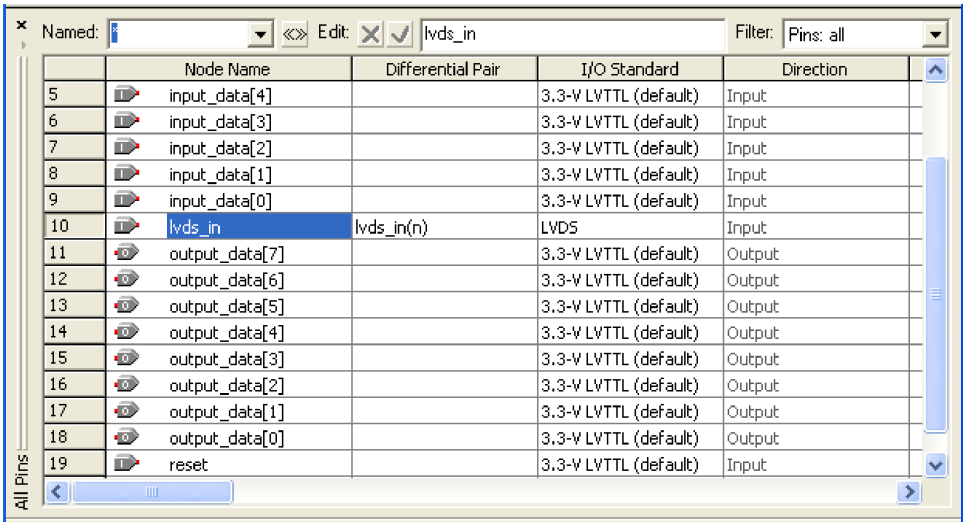
3.2.6. Assigning Differential Pins

When you assign a differential I/O standard to a single-ended top-level pin in your design, the Pin Planner automatically recognizes the negative pin as part of the differential pin pair assignment and creates the negative pin for you. The Quartus Prime software writes the location assignment for the negative pin to the `.qsf`; however, the I/O standard assignment is not added to the `.qsf` for the negative pin of the differential pair.

The following example shows a design with `lvds_in` top-level pin, to which you assign a differential I/O standard. The Pin Planner automatically creates the differential pin, `lvds_in(n)` to complete the differential pin pair.

Note: If you have a single-ended clock that feeds a PLL, assign the pin only to the positive clock pin of a differential pair in the target device. Single-ended pins that feed a PLL and are assigned to the negative clock pin device cause the design to not fit.

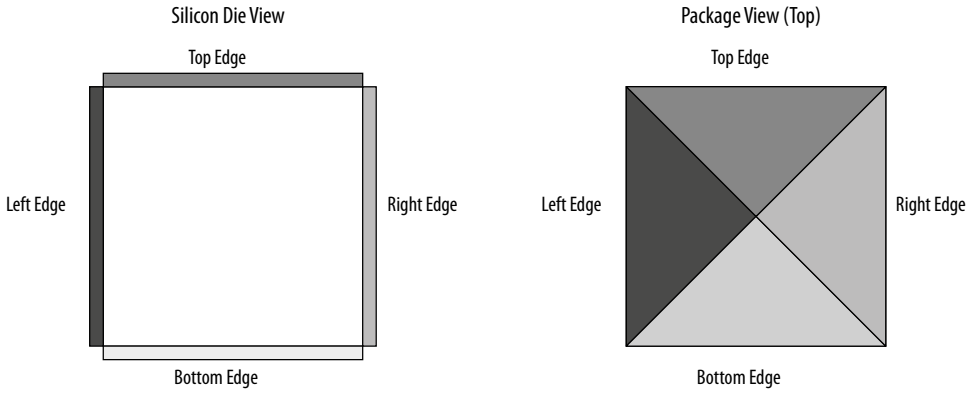
Figure 69. Creating a Differential Pin Pair in the Pin Planner



	Node Name	Differential Pair	I/O Standard	Direction
5	input_data[4]		3.3-V LVTTL (default)	Input
6	input_data[3]		3.3-V LVTTL (default)	Input
7	input_data[2]		3.3-V LVTTL (default)	Input
8	input_data[1]		3.3-V LVTTL (default)	Input
9	input_data[0]		3.3-V LVTTL (default)	Input
10	lvds_in	lvds_in(n)	LVDS	Input
11	output_data[7]		3.3-V LVTTL (default)	Output
12	output_data[6]		3.3-V LVTTL (default)	Output
13	output_data[5]		3.3-V LVTTL (default)	Output
14	output_data[4]		3.3-V LVTTL (default)	Output
15	output_data[3]		3.3-V LVTTL (default)	Output
16	output_data[2]		3.3-V LVTTL (default)	Output
17	output_data[1]		3.3-V LVTTL (default)	Output
18	output_data[0]		3.3-V LVTTL (default)	Output
19	reset		3.3-V LVTTL (default)	Input

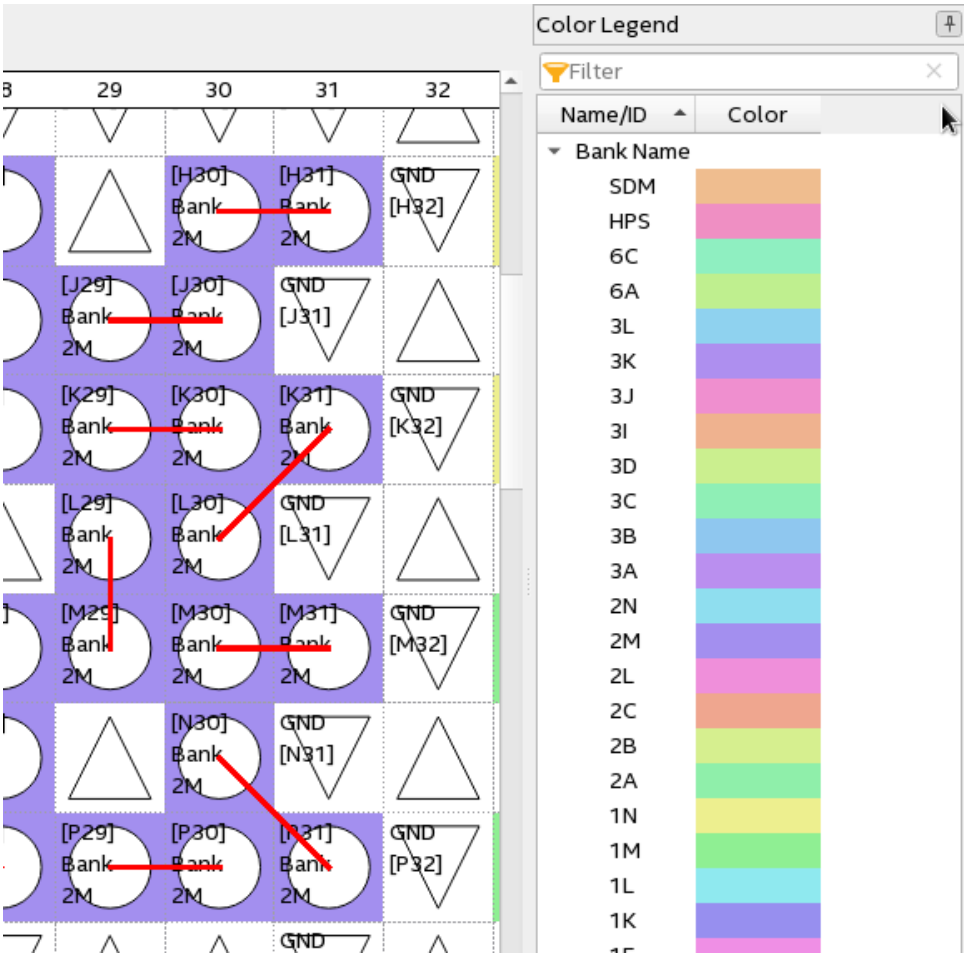
If your design contains a large bus that exceeds the pins available in a particular I/O bank, you can use edge location assignments to place the bus. Edge location assignments improve the circuit board routing ability of large buses, because they are close together near an edge. The following figure shows Intel device package edges.

Figure 70. Die View and Package View of the Four Edges on an Intel Device



When you assign differential pin pairs in **Package View**, a red connection line displays between the pair of differential pins. The Package View labels the positive and negative pins with the letters p and n, respectively.

Figure 71. Differential Pin Pair Color Coding



3.2.6.1. Overriding I/O Placement Rules on Differential Pins

I/O placement rules ensure that noisy signals do not corrupt neighboring signals. Each device family has predefined I/O placement rules.

I/O placement rules define, for example, the allowed placement of single-ended I/O with respect to differential pins, or how many output and bidirectional pins you can place within a VREF group when using voltage referenced input standards.

Use the **IO_MAXIMUM_TOGGLE_RATE** assignment to override I/O placement rules on pins, such as system reset pins that do not switch during normal design activity. Setting a value of 0 MHz for this assignment causes the Fitter to recognize the pin at a DC state throughout device operation. The Fitter excludes the assigned pin from placement rule analysis. Do not assign an **IO_MAXIMUM_TOGGLE_RATE** of 0 MHz to any actively switching pin, or your design may not function as you intend.

3.2.7. Entering Pin Assignments with Tcl Commands

You can apply pin assignments with Tcl scripts, by either entering individual Tcl commands in the Tcl Console, or creating a `.tcl` script and the typing the following in the command line:

Example 6. Applying Tcl Script Assignments

```
quartus_sh -t <my_tcl_script>.tcl
```

Example 7. Scripted Pin Assignment

The following example uses `set_location_assignment` and `set_instance_assignment` Tcl commands to assign a pin to a specific location, I/O standard, and drive strength.

```
set_location_assignment PIN_M20 -to address[10]
set_instance_assignment -name IO_STANDARD "2.5 V" -to address[10]
set_instance_assignment -name
        CURRENT_STRENGTH_NEW "MAXIMUM CURRENT" -to address[10]
```

Related Information

Quartus Prime Pro Edition User Guide: Scripting

3.2.8. Entering Pin Assignments in HDL Code

You can use synthesis attributes or low-level I/O primitives to embed I/O pin assignments directly in your HDL code. When you analyze and synthesize the HDL code, the information is converted into the appropriate I/O pin assignments. You can use either of the following methods to specify pin-related assignments with HDL code:

- Assigning synthesis attributes for signal names that are top-level pins
- Using low-level I/O primitives, such as ALT_BUF_IN, to specify input, output, and differential buffers, and for setting parameters or attributes

3.2.8.1. Using Low-Level I/O Primitives

You can alternatively enter I/O pin assignments using low-level I/O primitives. You can assign pin locations, I/O standards, drive strengths, slew rates, and on-chip termination (OCT) value assignments. You can also use low-level differential I/O primitives to define both positive and negative pins of a differential pair in the HDL code for your design.

Primitive-based assignments do not appear in the Pin Planner until after you perform a full compilation and back-annotate pin assignments (**Assignments > Back Annotate Assignments**).

3.3. Importing and Exporting I/O Pin Assignments

The Quartus Prime software supports transfer of I/O pin assignments across projects, or for analysis in third-party PCB tools. You can import or export I/O pin assignments in the following ways:

Table 31. Importing and Exporting I/O Pin Assignments

	Import Assignments	Export Assignments
Scenario	• From your PCB design tool or spreadsheet into Pin Planner during early pin planning or after optimization in PCB tool • From another Quartus Prime project with common constraints	• From Quartus Prime project for optimization in a PCB design tool • From Quartus Prime project for spreadsheet analysis or use in scripting assignments • From Quartus Prime project for import into another Quartus Prime project with similar constraints
Command	**Assignments ➤ Import Assignments**	**Assignments ➤ Export Assignments**
File formats	`.qsf`, `.csv`, `.txt`, `.sdc`	`.pin`, `.csv`, `.tcl`, `.qsf`
Notes	N/A	Exported `.csv` files retain column and row order and format. Do not modify the row of column headings if importing the `.csv` file

3.3.1. Importing and Exporting for PCB Tools

The Pin Planner supports import and export of assignments with PCB tools. You can export valid assignments as a `.pin` file for analysis in other supported PCB tools. You can also import optimized assignment from supported PCB tools. The `.pin` file contains pin name, number, and detailed properties.

Table 32. Contents of .pin File

File Column Name	Description
Pin Name/Usage	The name of the design pin, or whether the pin is GND or V_{CC} pin
Location	The pin number of the location on the device package
Dir	The direction of the pin
I/O Standard	The name of the I/O standard to which the pin is configured
Voltage	The voltage level that is required to be connected to the pin
I/O Bank	The I/O bank to which the pin belongs
User Assignment	Y or N indicating if the location assignment for the design pin was user assigned (Y) or assigned by the Fitter (N)

Related Information

PCB Design Tools Support
In *Quartus Prime Pro Edition User Guide: PCB Design Tools*

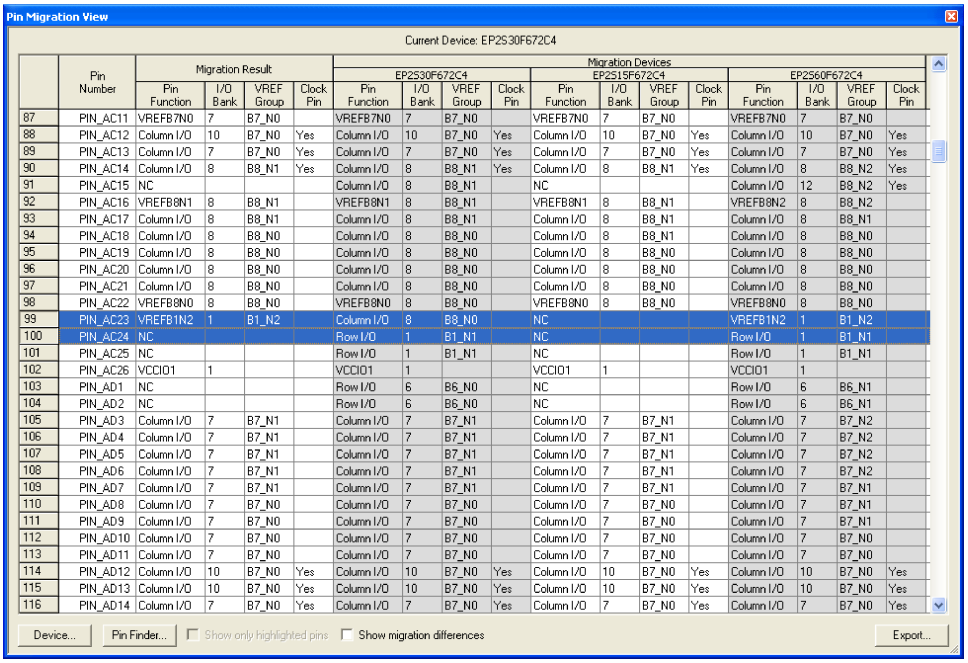
3.3.2. Migrating Assignments to Another Target Device

Click **View ➤ Pin Migration Window** to verify whether pin assignments are compatible with migration to a different Intel device.

You can migrate compatible pin assignments from one target device to another. You can migrate to a different density and the same device package. You can also migrate between device packages with different densities and pin counts.

The Quartus Prime software ignores invalid assignments and generates an error message during compilation. After evaluating migration compatibility, modify any incompatible assignments, and then click **Export** to export the assignments to another project.

Figure 72. Device Migration Compatibility (AC24 does not exist in migration device)



The migration result for the pin function of highlighted PIN_AC23 is not an NC but a voltage reference VREFB1N2 even though the pin is an NC in the migration device. VREF standards have a higher priority than an NC, thus the migration result displays the voltage reference. Even if you do not use that pin for a port connection in the design, you must use the VREF standard for I/O standards that require it on the actual board for the migration device.

If one of the migration devices has pins intended for connection to V_{CC} or GND and these same pins are I/O pins on a different device in the migration path, the Quartus Prime software ensures these pins are not used for I/O. Ensure that these pins are connected to the correct PCB plane.

When migrating between two devices in the same package, pins that are not connected to the smaller die may be intended to connect to V_{CC} or GND on the larger die. To facilitate migration, you can connect these pins to V_{CC} or GND in the original design because the pins are not physically connected to the smaller die.

3.4. Validating Pin Assignments

The Quartus Prime software validates I/O pin assignments against predefined I/O rules for your target device. You can use the following tools to validate your I/O pin assignments throughout the pin planning process:

Table 33. I/O Validation Tools

I/O Validation Tool	Description	Click to Run
Advanced I/O Timing	Fully validates I/O assignments against all I/O and timing checks during compilation	**Processing ➤ Start Compilation**

3.4.1. I/O Assignment Validation Rules

I/O Assignment Analysis validates your assignments against the following rules:

Table 34. Examples of I/O Rule Checks

Rule	Description	HDL Required?
I/O bank capacity	Checks the number of pins assigned to an I/O bank against the number of pins allowed in the I/O bank.	No
I/O bank VCCIO voltage compatibility	Checks that no more than one VCCIO is required for the pins assigned to the I/O bank.	No
I/O bank VREF voltage compatibility	Checks that no more than one VREF is required for the pins assigned to the I/O bank.	No
I/O standard and location conflicts	Checks whether the pin location supports the assigned I/O standard.	No
I/O standard and signal direction conflicts	Checks whether the pin location supports the assigned I/O standard and direction. For example, certain I/O standards on a particular pin location can only support output pins.	No
Differential I/O standards cannot have open drain turned on	Checks that open drain is turned off for all pins with a differential I/O standard.	No
I/O standard and drive strength conflicts	Checks whether the drive strength assignments are within the specifications of the I/O standard.	No
Drive strength and location conflicts	Checks whether the pin location supports the assigned drive strength.	No
BUSHOLD and location conflicts	Checks whether the pin location supports BUSHOLD. For example, dedicated clock pins do not support BUSHOLD.	No
WEAK_PULLUP and location conflicts	Checks whether the pin location supports WEAK_PULLUP (for example, dedicated clock pins do not support WEAK_PULLUP).	No
Electromigration check	Checks whether combined drive strength of consecutive pads exceeds a certain limit. For example, the total current drive for 10 consecutive pads on a Stratix II device cannot exceed 200 mA.	No
PCI_IO clamp diode, location, and I/O standard conflicts	Checks whether the pin location along with the I/O standard assigned supports PCI_IO clamp diode.	No
SERDES and I/O pin location compatibility check	Checks that all pins connected to a SERDES in your design are assigned to dedicated SERDES pin locations.	Yes
PLL and I/O pin location compatibility check	Checks whether pins connected to a PLL are assigned to the dedicated PLL pin locations.	Yes

Table 35. Signal Switching Noise Rules

Rule	Description	HDL Required?
I/O bank cannot have single-ended I/O when DPA exists	Checks that no single-ended I/O pin exists in the same I/O bank as a DPA.	No
A PLL I/O bank does not support both a single-ended I/O and a differential signal simultaneously	Checks that there are no single-ended I/O pins present in the PLL I/O Bank when a differential signal exists.	No
Single-ended output is required to be a certain distance away from a differential I/O pin	Checks whether single-ended output pins are a certain distance away from a differential I/O pin.	No
Single-ended output must be a certain distance away from a VREF pad	Checks whether single-ended output pins are a certain distance away from a VREF pad.	No
Single-ended input is required to be a certain distance away from a differential I/O pin	Checks whether single-ended input pins are a certain distance away from a differential I/O pin.	No
Too many outputs or bidirectional pins in a VREFGROUP when a VREF is used	Checks that there are no more than a certain number of outputs or bidirectional pins in a VREFGROUP when a VREF is used.	No
Too many outputs in a VREFGROUP	Checks whether too many outputs are in a VREFGROUP.	No

3.4.2. I/O Assignment Analysis

I/O assignment analysis validates I/O assignments against the complete set of I/O system and board layout rules. Full I/O assignment analysis validates blocks that directly feed or are fed by resources such as a PLL, LVDS, or gigabit transceiver blocks. In addition, the checker validates the legality of proper VREF pin use, pin locations, and acceptable mixed I/O standards

Run I/O assignment analysis during early pin planning to validate initial reserved pin assignments before compilation. Once you define design files, run I/O assignment analysis to perform more thorough legality checks with respect to the synthesized netlist. Run I/O assignment analysis whenever you modify I/O assignments.

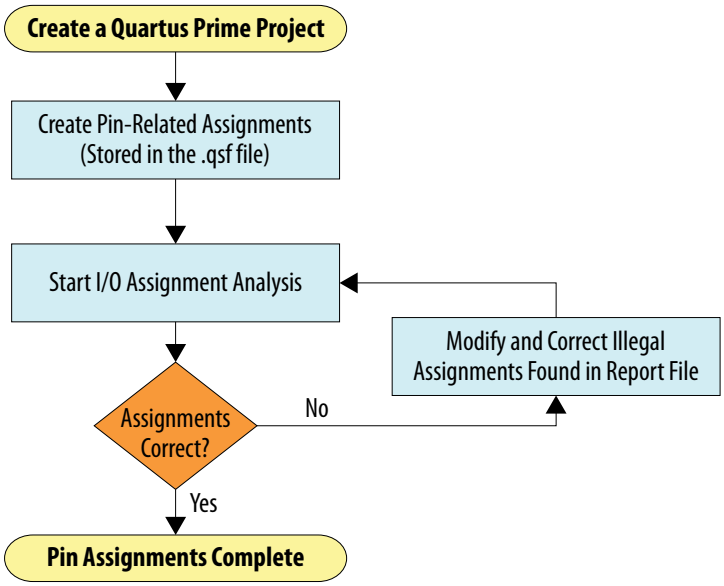
The Fitter assigns pins to accommodate your constraints. For example, if you assign an edge location to a group of LVDS pins, the Fitter assigns pin locations for each LVDS pin in the specified edge location and then performs legality checks. To display the Fitter-placed pins, click **Show Fitter Placements** in the Pin Planner. To accept these suggested pin locations, you must back-annotate your pin assignments.

View the I/O Assignment Warnings report to view and resolve all assignment warnings. For example, a warning that some design pins have undefined drive strength or slew rate. The Fitter recognizes undefined, single-ended output and bidirectional pins as non-calibrated OCT. To resolve the warning, assign the **Current Strength**, **Slew Rate** or **Slow Slew Rate** for the reported pin. Alternatively, can assign the **Termination** to the pin. You cannot assign drive strength or slew rate settings when a pin has an OCT assignment.

3.4.2.1. Early I/O Assignment Analysis Without Design Files

You can perform basic I/O legality checks before defining HDL design files. This technique produces a preliminary board layout. For example, you can specify a target device and enter pin assignments that correspond to PCB characteristics. You can reserve and assign I/O standards to each pin, and then run I/O assignment analysis to ensure that there are no I/O standard conflicts in each I/O bank.

Figure 73. Assigning and Analyzing Pin-Outs without Design Files

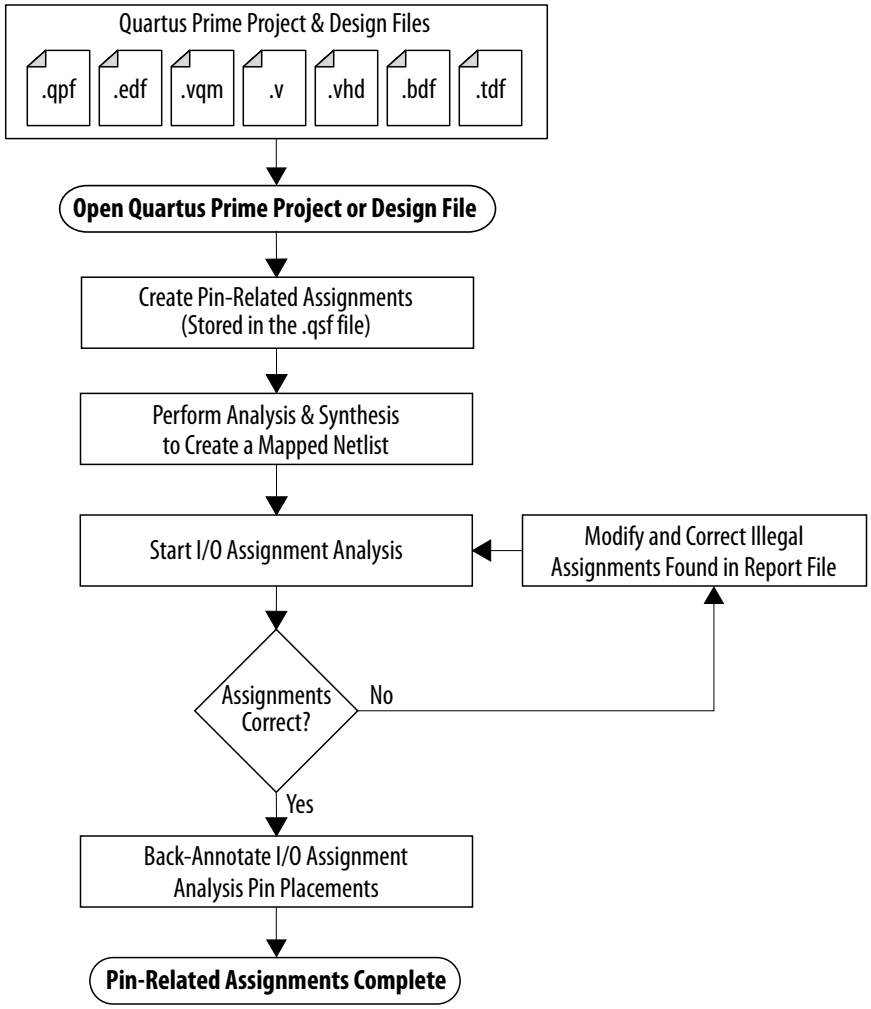


You must reserve all pins you intend to use as I/O pins, so that the Fitter can determine each pin type. After performing I/O assignment analysis, correct any errors reported by the Fitter and rerun I/O assignment analysis until all errors are corrected. A complete I/O assignment analysis requires all design files.

3.4.2.2. I/O Assignment Analysis With Design Files

I/O assignment analysis allows you to perform full I/O legality checks after fully defining HDL design files. When you run I/O assignment analysis on a complete design, the tool verifies all I/O pin assignments against all I/O rules. When you run I/O assignment analysis on a partial design, the tool checks legality only for defined portions of the design. The following figure shows the work flow for analyzing pin-outs with design files.

Figure 74. I/O Assignment Analysis Flow

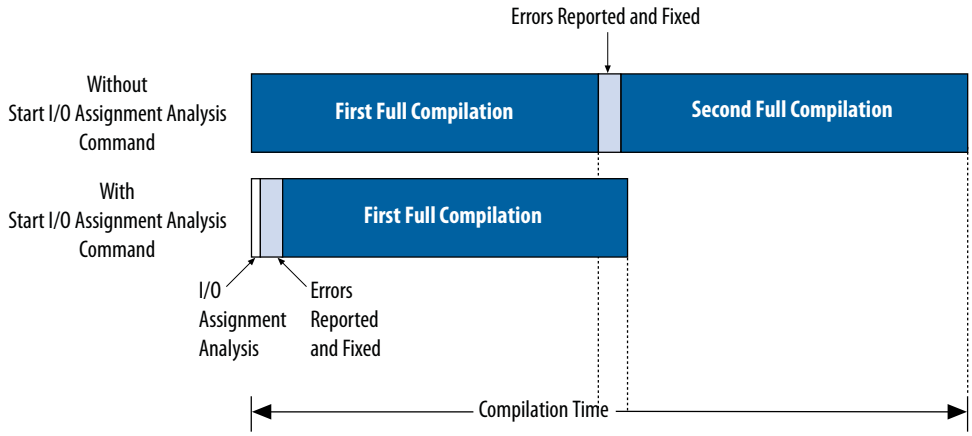


Even if I/O assignment analysis passes on incomplete design files, you may still encounter errors during full compilation. For example, you can assign a clock to a user I/O pin instead of assigning to a dedicated clock pin, or design the clock to drive a PLL that you have not yet instantiated in the design. This issues occur because I/O assignment analysis does not account for the logic that the pin drives and does not verify that only dedicated clock inputs can drive the a PLL clock port.

To obtain better coverage, analyze as much of the design as possible over time, especially logic that connects to pins. For example, if your design includes PLLs or LVDS blocks, define these files prior to full analysis. After performing I/O assignment analysis, correct any errors reported by the Fitter and rerun I/O assignment analysis until all errors are corrected.

The following figure shows the compilation time benefit of performing I/O assignment analysis before running a full compilation.

Figure 75. I/O Assignment Analysis Reduces Compilation Time



3.4.2.3. Overriding Default I/O Pin Analysis

You can override the default I/O analysis of pins to accommodate I/O rule exceptions, such as for analyzing VREF or inactive pins.

Each device contains VREF pins, each supporting one or more I/O pins. A VREF pin and its I/O pins comprise a VREF bank. The VREF pins are typically assigned inputs with VREF I/O standards, such as HSTL- and SSTL-type I/O standards. Conversely, VREF outputs do not require the VREF pin. When a voltage-referenced input is present in a VREF bank, only a certain number of outputs can be present in that VREF bank. I/O assignment analysis treats bidirectional signals controlled by different output enables as independent output enables.

To assign the **Output Enable Group** option to bidirectional signals to analyze the signals as a single output enable group, follow these steps:

1. To access this assignment in the Pin Planner, right-click the **All pins** list and click **Customize Columns**.
2. Under **Available columns**, add **Output Enable Group** to **Show these columns in this order**. The column appears in the **All Pins** list.
3. Enter the same integer value for the **Output Enable Group** assignment for all sets of signals that are driving in the same direction.

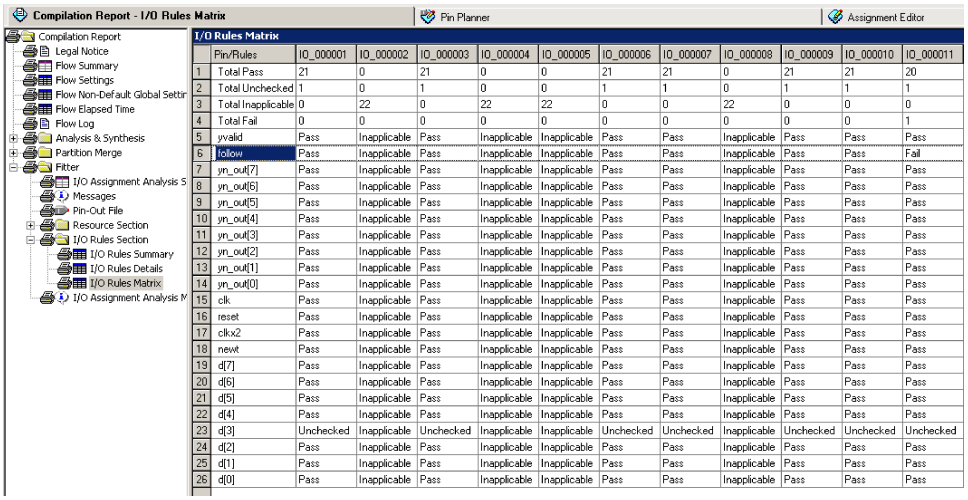
3.4.3. Understanding I/O Analysis Reports

The detailed I/O assignment analysis reports include the affected pin name and a problem description. The Fitter section of the Compilation report contains information generated during I/O assignment analysis, including the following reports:

- I/O Assignment Warnings—lists warnings generated for each pin
- Resource Section—quantifies use of various pin types and I/O banks
- I/O Rules Section—lists summary, details, and matrix information about the I/O rules tested

The **Status** column indicates whether rules passed, failed, or were not checked. A severity rating indicates the rule's importance for effective analysis. "Inapplicable" rules do not apply to the target device family.

Figure 76. I/O Rules Matrix



	Pin/Rules	IO_000001	IO_000002	IO_000003	IO_000004	IO_000005	IO_000006	IO_000007	IO_000008	IO_000009	IO_000010	IO_000011
1	Total Pass	21	0	21	0	0	21	21	0	21	21	20
2	Total Unchecked	1	0	1	0	0	1	1	0	1	1	1
3	Total Inapplicable	0	22	0	22	22	0	0	22	0	0	0
4	Total Fail	0	0	0	0	0	0	0	0	0	0	1
5	yvalid	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
6	follow	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Fail
7	yn_out[7]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
8	yn_out[6]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
9	yn_out[5]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
10	yn_out[4]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
11	yn_out[3]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
12	yn_out[2]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
13	yn_out[1]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
14	yn_out[0]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
15	clk	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
16	reset	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
17	clk:x2	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
18	newt	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
19	d[7]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
20	d[6]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
21	d[5]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
22	d[4]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
23	d[3]	Unchecked	Inapplicable	Unchecked	Inapplicable	Inapplicable	Unchecked	Unchecked	Inapplicable	Unchecked	Unchecked	Unchecked
24	d[2]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
25	d[1]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass
26	d[0]	Pass	Inapplicable	Pass	Inapplicable	Inapplicable	Pass	Pass	Inapplicable	Pass	Pass	Pass

3.5. Verifying I/O Timing

You must verify board-level signal integrity and I/O timing when assigning I/O pins. High-speed interface operation requires a quality signal and low propagation delay at the far end of the board route. Click **Tools ➤ Timing Analyzer** to confirm timing after making I/O pin assignments.

For example, if you change the slew rates or drive strengths of some I/O pins with ECOs, you can verify timing without recompiling the design. You must understand I/O timing and what factors affect I/O timing paths in your design. The accuracy of the output load specification of the output and bidirectional pins affects the I/O timing results.

The Quartus Prime software supports three different methods of I/O timing analysis:

Table 36. I/O Timing Analysis Methods

I/O Timing Analysis	Description
Advanced I/O timing analysis	Analyze I/O timing with your board trace model to report accurate, "board-aware" simulation models. Configures a complete board trace model for each I/O standard or pin. Timing Analyzer applies simulation results of the I/O buffer, package, and board trace model to generate accurate I/O delays and system level signal information. Use this information to improve timing and signal integrity.
I/O timing analysis	Analyze I/O timing with default or specified capacitive load without signal integrity analysis. Timing Analyzer reports tCO to an I/O pin using a default or user-specified value for a capacitive load.
Full board routing simulation	Use Intel-provided or Quartus Prime software-generated IBIS or HSPICE I/O models for simulation in Mentor Graphics* HyperLynx* and Synopsys HSPICE.

For more information about advanced I/O timing support, refer to the appropriate device handbook for your target device. For more information about board-level signal integrity and tips on how to improve signal integrity in your high-speed designs, refer to the Signal Integrity and Power Integrity – Support Center website.

For information about creating IBIS and HSPICE models with the Quartus Prime software and integrating those models into HyperLynx and HSPICE simulations, refer to the *Signal Integrity Analysis with Third Party Tools* chapter.

Related Information

Signal Integrity and Power Integrity – Support Center

3.5.1. Running Advanced I/O Timing

Advanced I/O timing analysis uses your board trace model and termination network specification to report accurate output buffer-to-pin timing estimates, FPGA pin and board trace signal integrity and delay values. Advanced I/O timing runs automatically for supported devices during compilation.

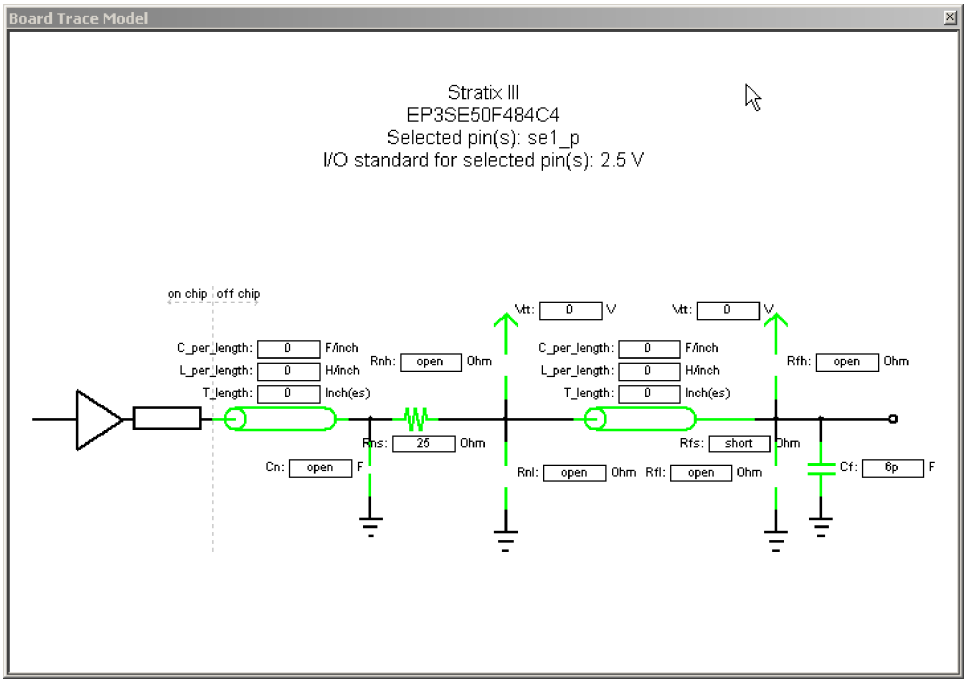
3.5.1.1. Board Trace Models

The Quartus Prime software provides board trace model templates for various I/O standards.

The following figure shows the template for a **2.5 V** I/O standard. This model consists of near-end and far-end board component parameters.

Near-end board trace modeling includes the elements which are close to the device. Far-end modeling includes the elements which are at the receiver end of the link, closer to the receiving device. Board trace model topology is conceptual and does not necessarily match the actual board trace for every component. For example, near-end model parameters can represent device-end discrete termination and breakout traces. Far-end modeling can represent the bulk of the board trace to discrete external memory components, and the far end termination network. You can analyze the same circuit with near-end modeling of the entire board, including memory component termination, and far-end modeling of the actual memory component.

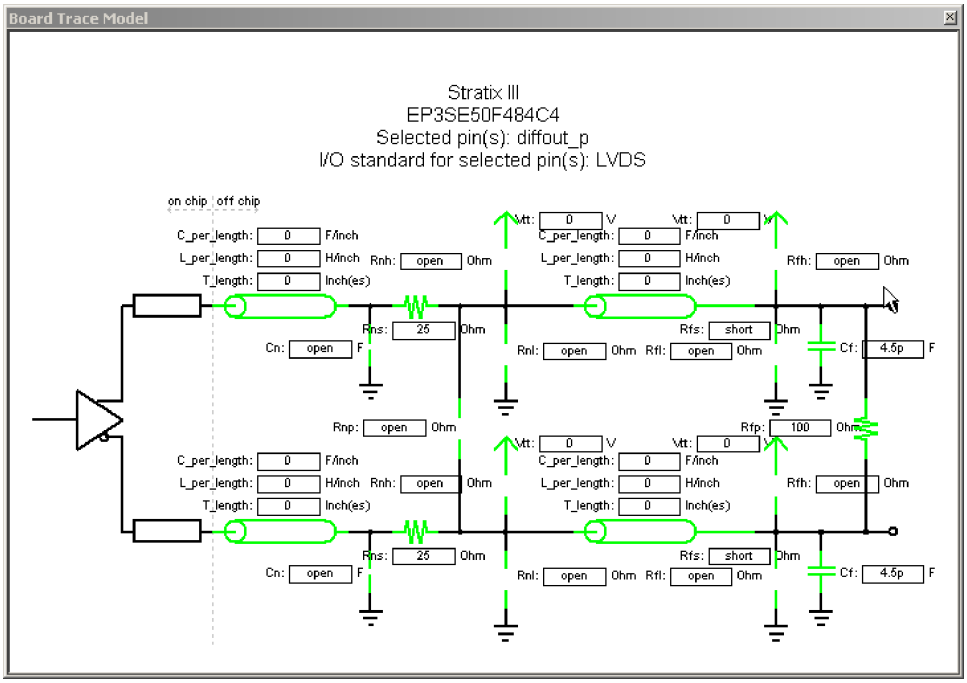
Figure 77. 2.5-V I/O Standard Board Trace Model



The following figure shows the template for the **LVDS** I/O standard. The far-end capacitance (Cf) represents the external-device or multiple-device capacitive load. If you have multiple devices on the far-end, you must find the equivalent capacitance at the far-end, taking into account all receiver capacitances. The far-end capacitance can be the sum of all the receiver capacitances.

The Quartus Prime software models of transmission lines do not consider transmission-line resistance (lossless models). You only need to specify distributed inductance (L) and capacitance (C) values on a per-inch basis, which you can obtain from the PCB vendor or manufacturer, the CAD Design tool, or a signal integrity tool, such as the Mentor Graphics HyperLynx software.

Figure 78. LVDS Differential Board Trace Model



3.5.1.2. Defining the Board Trace Model

The board trace model describes a board trace and termination network as a set of capacitive, resistive, and inductive parameters.

Advanced I/O Timing uses the model to simulate the output signal from the output buffer to the far end of the board trace. You can define the capacitive load, any termination components, and trace impedances in the board routing for any output pin or bidirectional pin in output mode. You can configure an overall board trace model for each I/O standard or for specific pins. Define an overall board trace model for each I/O standard in your design. Use that model for all pins that use the I/O standard. You can customize the model for specific pins using the **Board Trace Model** window in the Pin Planner.

1. Click **Assignments ➤ Device ➤ Device and Pin Options**.
2. Click **Board Trace Model** and define board trace model values for each I/O standard.
3. Click **I/O Timing** and define default I/O timing options at board trace near and far ends.
4. Click **Assignments ➤ Pin Planner** and assign board trace model values to individual pins.

Example 8. Specifying Board Trace Model

```
## setting the near end series resistance model of sel_p output pin to 25 ohms
set_instance_assignment -name BOARD_MODEL_NEAR_SERIES_R 25 -to se1_p
## Setting the far end capacitance model for sel_p output signal to 6 picofarads
set_instance_assignment -name BOARD_MODEL_FAR_C 6P -to se1_p
```

3.5.1.3. Modifying the Board Trace Model

To modify the board trace model, click **View ➤ Board Trace Model** in the Pin Planner.

You can modify any of the board trace model parameters within a graphical representation of the board trace model.

The **Board Trace Model** window displays the routing and components for positive and negative signals in a differential signal pair. Only modify the positive signal of the pair, as the setting automatically applies to the negative signal. Use standard unit prefixes such as *p*, *n*, and *k* to represent pico, nano, and kilo, respectively. Use the **short** or **open** value to designate a short or open circuit for a parallel component.

3.5.1.4. Specifying Near-End vs Far-End I/O Timing Analysis

You can select a near-end or far-end point for I/O timing analysis. Near-end timing analysis extends to the device pin. You can apply the `set_output_delay` constraint during near-end analysis to account for the delay across the board.

With far-end I/O timing analysis, the advanced I/O timing analysis extends to the external device input, at the far-end of the board trace. Whether you choose a near-end or far-end timing endpoint, the board trace models are taken into account during timing analysis.

3.5.1.5. Advanced I/O Timing Analysis Reports

The following reports show advanced I/O timing analysis information:

Table 37. Advanced I/O Timing Reports

I/O Timing Report	Description
Timing Analyzer Report	Reports signal integrity and board delay data.
Board Trace Model Assignments report	Summarizes the board trace model component settings for each output and bidirectional signal.
Signal Integrity Metrics report	Contains all the signal integrity metrics calculated during advanced I/O timing analysis based on the board trace model settings for each output or bidirectional pin. Includes measurements at both the FPGA pin and at the far-end load of board delay, steady state voltages, and rise and fall times.

Note: By default, the Timing Analyzer generates the Slow-Corner Signal Integrity Metrics report. To generate a Fast-Corner Signal Integrity Metrics report you must change the delay model by clicking **Tools ➤ Timing Analyzer**.

3.5.2. Adjusting I/O Timing and Power with Capacitive Loading

When calculating t_{CO} and power for output and bidirectional pins, the Timing Analyzer and the Power Analyzer use a bulk capacitive load. You can adjust the value of the capacitive load per I/O standard to obtain more precise t_{CO} and power measurements, reflecting the behavior of the output or bidirectional net on your PCB. The Quartus Prime software ignores capacitive load settings on input pins. You can adjust the capacitive load settings per I/O standard, in picofarads (pF), for your entire design. During compilation, the Compiler measures power and t_{CO} measurements based on your settings. You can also adjust the capacitive load on an individual pin with the **Output Pin Load** logic option.

3.6. Viewing Routing and Timing Delays

Right-click any node and click **Locate > Locate in Chip Planner** to visualize and adjust I/O timing delays and routing between user I/O pads and V_{CC}, GND, and V_{REF} pads. The Chip Planner graphically displays logic placement, Logic Lock regions, relative resource usage, detailed routing information, fan-in and fan-out, register paths, and high-speed transceiver channels. You can view physical timing estimates, routing congestion, and clock regions. Use the Chip Planner to change connections between resources and make post-compilation changes to logic cell and I/O atom placement. When you select items in the Pin Planner, the corresponding item is highlighted in Chip Planner.

3.7. Scripting API

The Quartus Prime software allows you to access I/O management functions through Tcl commands, rather than with the GUI. For detailed information about scripting command options and Tcl API packages, type the following at a system command prompt to view the Tcl API Help browser:

```
quartus_sh --qhelp
```

Related Information

Quartus Prime Pro Edition User Guide: Scripting

3.7.1. Generate Mapped Netlist

Enter the following in the Tcl console or in a Tcl script:

```
execute_module -tool map
```

The `execute_module` command is in the flow package.

Type the following at a system command prompt:

```
quartus_syn <project name>
```

3.7.2. Reserve Pins

Use the following Tcl command to reserve a pin:

```
set_instance_assignment -name RESERVE_PIN <value> -to <signal name>
```

Use one of the following valid reserved pin values:

- `"AS BIDIRECTIONAL"`
- `"AS INPUT TRI STATED"`
- `"AS OUTPUT DRIVING AN UNSPECIFIED SIGNAL"`
- `"AS OUTPUT DRIVING GROUND"`
- `"AS SIGNALPROBE OUTPUT"`

Note: You must include the quotation marks when specifying the reserved pin value.

3.7.3. Set Location

Use the following Tcl command to assign a signal to a pin or device location:

```
set_location_assignment <location> -to <signal name>
```

Valid locations are pin locations, I/O bank locations, or edge locations. Pin locations include pin names, such as PIN_A3. I/O bank locations include IOBANK_1 up to IOBANK_ *n*, where *n* is the number of I/O banks in the device.

Use one of the following valid edge location values:

- EDGE_BOTTOM
- EDGE_LEFT
- EDGE_TOP
- EDGE_RIGHT

3.7.4. Exclusive I/O Group

The following Tcl command creates an exclusive I/O group assignment:

```
set_instance_assignment -name "EXCLUSIVE_IO_GROUP" -to pin
```

3.7.5. Slew Rate and Current Strength

Use the following Tcl commands to create a slew rate and drive strength assignments:

```
set_instance_assignment -name CURRENT_STRENGTH_NEW 8MA -to e[0]
set_instance_assignment -name SLEW_RATE 2 -to e[0]
```

3.8. Managing Device I/O Pins Revision History

The following table shows the revision history for this chapter:

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied initial Altera rebranding throughout. • Updated Pin Planner screenshot in *Importing and Exporting I/O Pin Assignments* topic for current list of file formats supported.
2023.12.04	23.4	• Updated *Intel FPGA Device and I/O Terminology* topic for removal of VREF Groups and Edges highlight feature. • Updated Pin Planner screenshot in *Managing Device I/O Pins* topic for removal of VREF Groups and Edges highlight feature.
2023.10.02	23.3	• Added *Assigning I/O Banks* topic to describe color coding and saving changes. • Added *Changing Pin Planner Highlight Colors* topic to describe changing report highlight coloring. • Added *Showing I/O Lanes* topic to describe new Show I/O 12 Lanes report.
2022.04.27	22.1	Made a minor fix.

continued...

Document Version	Quartus Prime Version	Changes
2020.11.04	19.3	Removed references to obsolete FPGA Xchange file (.fx) support from "Integrating PCB Design Tools" and "Importing and Exporting I/O Pin Assignments" topics.
2018.05.07	18.0	• First release as part of the stand-alone *Design Constraints User Guide*
2017.11.06	17.1	• Revised topic: I/O Planning Overview. • Revised topic: Basic I/O Planning Flow with the Pin Planner and renamed to Basic I/O Planning Flow with the Pin Planner.
2017.05.08	17.0	• Renamed command: Run I/O Assignment Analysis to Start Fitter (Plan).
2016.10.31	16.1	• Implemented Intel rebranding.
2015.11.02	15.1	• Removed early pin planning and Live I/O Check support from Quartus Prime Pro Edition handbook • Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1	• Updated Live I/O check device support to include only limited device families.
2014.08.30	14.0a10	• Added link to information about special pin assignment features for Arria 10 SoC devices.
2014.06.30	14.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog.
November 2013	13.1	• Reorganization and conversion to DITA.
May 2013	13.0	• Added information about overriding I/O placement rules.
November 2012	12.1	• Updated Pin Planner description for new task and report windows.
June 2012	12.0.0	• Removed survey link.
November 2011	11.1	• Minor updates and corrections. • Updated the document template.
December 2010	10.0	Template update
July 2010	10.0	• Reorganized and edited the chapter • Added links to Help for procedural information previously included in the chapter • Added information on rules marked Inapplicable in the I/O Rules Matrix Report • Added information on assigning slew rate and drive strength settings to pins to fix I/O assignment warnings
November 2009	9.1	• Reorganized entire chapter to include links to Help for procedural information previously included in the chapter • Added documentation on near-end and far-end advanced I/O timing
March 2009	9.0	• Updated “Pad View Window” on page 5–20 • Added new figures: • Figure 5–15 • Figure 5–16 • Added new section “Viewing Simultaneous Switching Noise (SSN) Results” on page 5–17 • Added new section “Creating Exclusive I/O Group Assignments” on page 5–18

4. Quartus Prime Pro Edition User Guide: Design Constraints Document Archives

For the latest and previous versions of this user guide, refer to Quartus Prime Pro Edition User Guide: Design Constraints. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

PCB Design Tools

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

1. Signal Integrity Analysis with Third-Party Tools

1.1. Signal Integrity Analysis with Third-Party Tools

With the ever-increasing operating speed of interfaces in traditional FPGA design, the timing and signal integrity margins between the FPGA and other devices on the board must be within specification and tolerance before building a PCB.

If the board trace design is poor, or the route is too heavily loaded, noise in the signal can cause data corruption, while overshoot and undershoot can potentially damage input buffers over time.

As FPGA devices are used in high-speed applications, signal integrity and timing margin between the FPGA and other devices on the PCB are important for proper system operation. To avoid time-consuming and costly board respins, you must simulate the topology and routing of critical signals. You must accurately model the high-speed interfaces available on FPGA devices.

The Quartus® Prime software provides methodologies, resources, and tools to ensure good signal integrity and timing margin between Intel® FPGA devices and other components on the board. Three types of analysis are possible with the Quartus Prime software:

- I/O timing with a default or user-specified capacitive load and no signal integrity analysis (default)
- Full board routing simulation in third-party tools using Intel-provided or generated Input/Output Buffer Information Specification (IBIS) or HSPICE I/O models

I/O timing using a specified capacitive test load requires no special configuration other than setting the size of the load. The Quartus Prime Timing Analyzer generates I/O timing reports based only on point-to-point delays within the I/O buffer. The Timing Analyzer assumes the presence of the capacitive test load without specifying any other details about the board. The default size of the load derives from the I/O standard that you select for the pin. Timing analysis measures to the FPGA pin without signal integrity analysis details.

The signal integrity information in this chapter refers to board-level signal integrity based on I/O buffer configuration and board parameters, not simultaneous switching noise (SSN).(1) SSN is a product of multiple output drivers switching at the same time, causing an overall drop in the voltage of the chip's power supply. This condition can cause temporary glitches in the specified level of ground or V_{CC} for the device.

(1) Also known as ground bounce or V_{CC} sag.

This chapter provides FPGA and board designers with the concepts and steps necessary to perform signal integrity simulation and adjust designs to improve board-level timing and signal integrity. This chapter also includes information about how to obtain and customize simulation models, and how to use those models in simulation software.

1.1.1. What's New In This Version

- The current version of the Quartus Prime Pro Edition software supports generation of custom IBIS models for Agilex™ FPGA portfolio devices using the EDA Netlist Writer GUI, as Generate Custom IBIS Models with the EDA Netlist Writer GUI describes.
- The current version of the Quartus Prime Pro Edition software no longer includes the Enable Advanced I/O Timing option, as reflected throughout this document.

1.1.2. Signal Integrity Simulations with HSPICE and IBIS Models

The Quartus Prime software can export accurate HSPICE models with the built-in HSPICE Writer. You can run signal integrity simulations with these complete HSPICE models in Synopsys* HSPICE. You can also easily create and customize IBIS models of the FPGA I/O buffers in the Quartus Prime software.

You can run signal integrity simulations with these complete HSPICE models in Synopsys HSPICE.

You can integrate IBIS models into any third-party simulation tool that supports IBIS models. With the ability to create industry-standard model definition files quickly, you can build accurate simulations that can provide data to help improve board-level signal integrity.

Creating and running accurate simulations can be difficult and time consuming. The Quartus Prime software tools automate the I/O model setup and creation process by generating custom models for your design. These tools allow you to set up and run accurate simulations that guide FPGA and board design.

For a more information about SSN and ways to prevent it, refer to *AN 315: Guidelines for Designing High-Speed FPGA PCBs*.

For information about basic signal integrity concepts and signal integrity details pertaining to Intel FPGA devices, visit the Intel Signal & Power Integrity Center.

Related Information

- AN 315: Guidelines for Designing High-Speed FPGA PCBs
- Intel Signal & Power Integrity Center

1.2. I/O Model Selection: IBIS or HSPICE

The Quartus Prime software can export two different types of I/O models that are useful for different simulation situations, IBIS models and HSPICE models.

IBIS models define the behavior of input or output buffers through voltage-current (V-I) and voltage-time (V-t) data tables. HSPICE models, or decks, include complete physical descriptions of the transistors and parasitic capacitances that make up an I/O buffer along with all the parameter settings that you require to run a simulation.

The Quartus Prime software generates HSPICE decks, and adds preconfigured I/O standard, voltage, and pin loading settings for each pin in your design.

The choice of I/O model type is based on many factors.

Table 1. IBIS and HSPICE Model Comparison

Feature	IBIS Model	HSPICE Model
I/O Buffer Description	**Behavioral**—I/O buffers are described by voltage-current and voltage-time tables in typical, minimum, and maximum supply voltage cases.	**Physical**—I/O buffers and all components in a circuit are described by their physical properties, such as transistor characteristics and parasitic capacitances, as well as their connections to one another.
Model Customization	**Simple and limited**—The model completely describes the I/O buffer and does not usually have to be customized.	**Fully customizable**—Unless connected to an arbitrary board description, the description of the board trace model must be customized in the model file. All parameters of the simulation are also adjustable.
Simulation Set Up and Run Time	**Fast**—Simulations run quickly after set up correctly.	**Slow**—Simulations take time to set up and take longer to run and complete.
Simulation Accuracy	**Good**—For most simulations, accuracy is sufficient to make useful adjustments to the FPGA or board design to improve signal integrity.	**Excellent**—Simulations are highly accurate, making HSPICE simulation almost a requirement for any high-speed design where signal integrity and timing margins are tight.
Third-Party Tool Support	**Excellent**—Almost all third-party board simulation tools support IBIS.	**Good**—Most third-party tools that support SPICE support HSPICE. However, Synopsys HSPICE is required for simulations of Intel's encrypted HSPICE models.

Related Information

AN 283: Simulating Intel Devices with IBIS Models

1.3. FPGA to Board Signal Integrity Analysis Flow

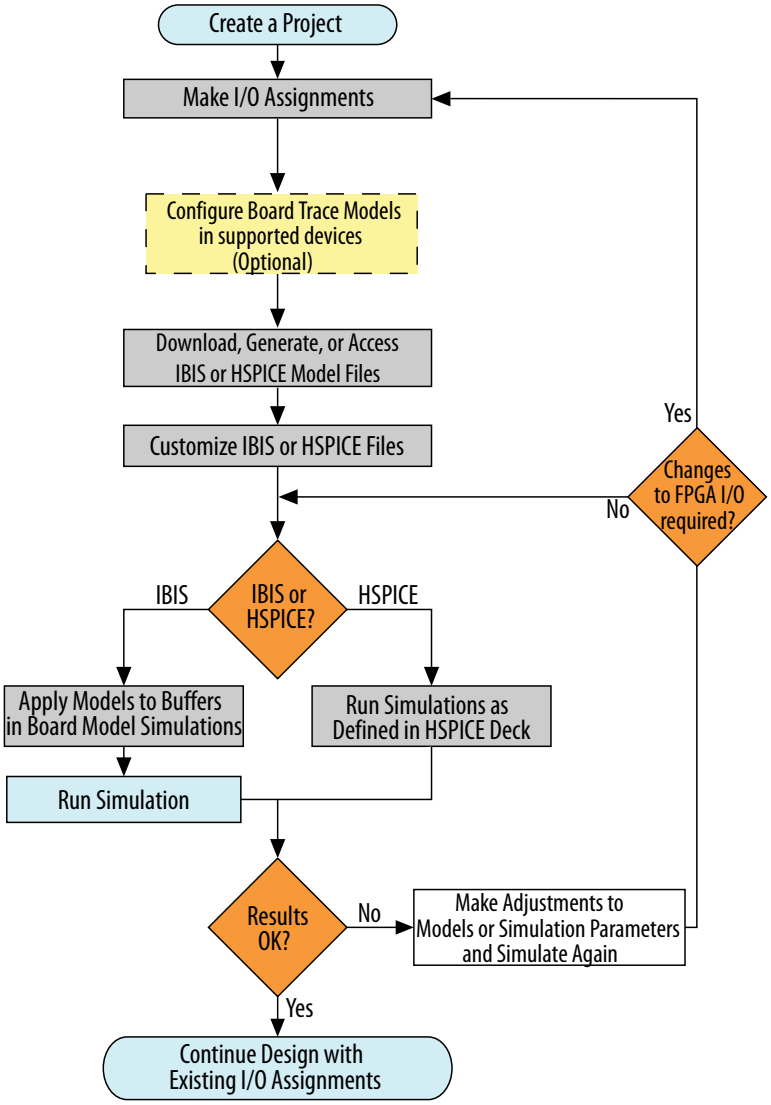
Board signal integrity analysis can take place at any point in the FPGA design process and is often performed before and after board layout. If it is performed early in the process as part of a pre-PCB layout analysis, the models used for simulations can be more generic.

These models can be changed as much as required to see how adjustments improve timing or signal integrity and help with the design and routing of the PCB. Simulations and the resulting changes made at this stage allow you to analyze "what if" scenarios to plan and implement your design better. To assist with early board signal integrity analysis, you can download generic IBIS model files for each device family and obtain HSPICE buffer simulation kits from the "Board Level Tools" section of the EDA Tool Support Resource Center.

Typically, if board signal integrity analysis is performed late in the design, it is used for a post-layout verification. The inputs and outputs of the FPGA are defined, and required board routing topologies and constraints are known. Simulations can help you find problems that might still exist in the FPGA or board design before fabrication and assembly. In either case, a simple process flow illustrates how to create accurate IBIS and HSPICE models from a design in the Quartus Prime software and transfer them to third-party simulation tools.

Your design depends on the type of model, IBIS or HSPICE, that you use for your simulations. When you understand the steps in the analysis flow, refer to the section of this chapter that corresponds to the model type you are using.

Figure 1. Third-Party Board Signal Integrity Analysis Flow



Related Information

EDA Tool Support Resource Center
For more information, generic IBIS model files for each device family, and to obtain HSPICE buffer simulation kits.

1.3.1. Create I/O and Board Trace Model Assignments

You can configure a board trace model for output signals or for bidirectional signals in output mode. You can then automatically transfer its description to HSPICE decks generated by the HSPICE Writer. This helps improve simulation accuracy.

To configure a board trace model, specify the board trace model assignment settings for each I/O standard used in your design. You can add series or parallel termination, specify the transmission line length, and set the value of the far-end capacitive load. You can configure these parameters in the Board Trace Model view of the Pin Planner (**Assignments ➤ Pin Planner**).

The Quartus Prime software can generate IBIS models and HSPICE decks without having to configure a board trace model. Generated IBIS models ignore any board trace model settings other than the far-end capacitive load. If any load value is set other than the default, the delay given by IBIS models generated by the IBIS Writer cannot be used to account correctly for the double counting problem.

The load value mismatch between the IBIS delay and the t_{CO} measurement of the Quartus Prime software prevents the delays from being safely added together. Warning messages displayed when the EDA Netlist Writer runs indicate when this mismatch occurs.

1.3.2. Customize the Output Files

You can readily customize the files that the IBIS Writer and HSPICE Writer generate. You must customize any generic IBIS files with the correct RLC values for your specific device package before running signal integrity simulations.

If you generate IBIS files with the EDA Netlist Writer or IBIS Writer script, the IBIS Writer automatically customizes the files with the RLC values for the current target device. For details, refer to Simulation with IBIS Models.

You can make additions or adjustments to the default simulation in the generated files to change the parameters of the default simulation or to perform additional measurements. For details, refer to Simulation with HSPICE Models.

1.3.3. Set Up and Run Simulations in Third-Party Tools

When you have generated the files, you can use them to perform simulations in your selected simulation tool.

With IBIS models, you can apply them to input, output, or bidirectional buffer entities and quickly set up and run simulations. For HSPICE decks, the simulation parameters are included in the files. Open the files in Synopsys HSPICE and run simulations for each pin as required.

With HSPICE decks generated from the HSPICE Writer, the double counting problem is accounted for, which ensures that your simulations are accurate.

Simulations that involve IBIS models created with anything other than the default loading settings in the Quartus Prime software must take into account the change in the size of the load between the IBIS delay and the Quartus Prime t_{CO} measurement. Warning messages during compilation alert you to this change.

1.3.4. Interpret Simulation Results

If you encounter timing or signal integrity issues with your high-speed signals after running simulations, you can make adjustments to I/O assignment settings in the Quartus Prime software.

You can adjust drive strength or I/O standard, or make changes to the board routing or topology. After regenerating models in the Quartus Prime software based on the changes you have made, rerun the simulations to check whether your changes corrected the problem.

1.4. Simulation with IBIS Models

IBIS models provide a way to run accurate signal integrity simulations quickly. IBIS models describe the behavior of I/O buffers with voltage-current and voltage-time data curves.

Because of their behavioral nature, IBIS models do not have to include any information about the internal circuit design of the I/O buffer. Intel provides free IBIS models for use with Intel FPGA designs in signal integrity analysis simulation tools. You can obtain and customize these models for signal integrity design simulation.

1.4.1. IBIS Model Access and Customization Flows

There are different methods of accessing and customizing the IBIS models, depending on your target device family.

Table 2. IBIS Model Access and Customization

IBIS Model Access and Customization Method	Stratix® 10 Devices Arria® 10 Devices Cyclone® 10 GX Devices	Agilex FPGA Portfolio Devices
Obtaining IBIS Models	• Download generic device family IBIS models from the Intel website to perform early simulations of the I/O buffers you expect to use in your design as part of a pre-layout analysis at: IBIS Models for Intel FPGA Devices. The downloaded models have the RLC package values set to one particular device in each device family. Or • Use the Quartus Prime EDA Netlist Writer GUI to generate custom IBIS models that accurately reflect your device and assignments.	• Quartus Prime Pro Edition installation includes the IBIS models and IBIS Writer script for Agilex FPGA portfolio devices in: `/common/misc/ibis_writer/` Or • Download generic device family IBIS models from the Intel website to perform early simulations of the I/O buffers you expect to use in your design as part of a pre-layout analysis at: IBIS Models for Intel FPGA Devices. The downloaded models have the RLC package values set to one particular device in each device family and require customization for accurate simulation.
Customizing IBIS Models	• For more accurate IBIS module simulation, you must first customize any generic IBIS files that you download from the Intel website with the correct RLC values for the specific device package you have selected for your design. Or • Generate custom IBIS files that with Quartus Prime EDA Netlist Writer. IBIS files that you generate with the EDA Netlist Writer automatically include the RLC values for your current target device. *Note:* The `ibis_writer.py` script does not support generation of IBIS model files for Stratix 10 devices, Arria 10 devices, or Cyclone 10 GX devices.	• For more accurate IBIS module simulation, you must first customize the installed or downloaded IBIS model files with the correct RLC values for the specific target device package using the IBIS Writer script in `/quartus/common/misc/ibis_writer/`. The `README.txt` file in this directory provides complete instructions for using the script. • Alternatively, you can generate custom IBIS files by using the Quartus Prime software EDA Netlist Writer GUI. IBIS files that you generate with the EDA Netlist Writer automatically include the RLC values for your current target device.

Related Information

Generate Custom IBIS Models with the EDA Netlist Writer GUI on page 11

1.4.2. Elements of an IBIS Model

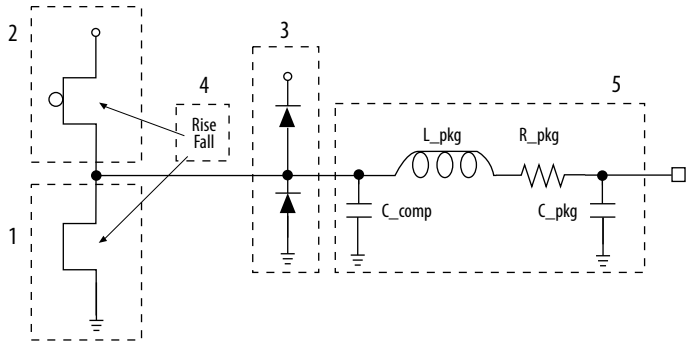
An IBIS model file (`.ibs`) is a text file that describes the behavior of an I/O buffer across minimum, typical, and maximum temperature and voltage ranges with a specified test load.

The tables and values specified in the IBIS file describe five basic elements of the I/O buffer.

The following elements correspond to each numbered block.

1. **Pulldown**—A voltage-current table describes the current when the buffer is driven low based on a pull-down voltage range of $-V_{CC}$ to 2 V_{CC}.
2. **Pullup**—A voltage-current table describes the current when the buffer is driven high based on a pull-up voltage range of $-V_{CC}$ to V_{CC}.
3. **Ground and Power Clamps**—Voltage-current tables describe the current when clamping diodes for electrostatic discharge (ESD) are present. The ground clamp voltage range is $-V_{CC}$ to V_{CC}, and the power clamp voltage range is $-V_{CC}$ to ground.
4. **Ramp and Rising/Falling Waveform**—A voltage-time (dv/dt) ratio describes the rise and fall time of the buffer during a logic transition. Optional rising and falling waveform tables can be added to more accurately describe the characteristics of the rising and falling transitions.
5. **Total Output Capacitance and Package RLC**—The total output capacitance includes the parasitic capacitances of the output pad, clamp diodes (if present), and input transistors. The package RLC is device package-specific and defines the resistance, inductance, and capacitance of the bond wire and pin of the I/O.

Figure 2. Five Basic Elements of an I/O Buffer in IBIS Models



Related Information

AN 283: Simulating Intel Devices with IBIS Models
For more information about IBIS models and Intel-specific features, including links to the official IBIS specification.

1.4.3. Customizing IBIS Models

There are different options for obtaining and customizing Intel FPGA IBIS models, depending on your target device family, as IBIS Model Access and Customization Flows describes. The following topics describe these different options for obtaining and customizing Intel FPGA IBIS models.

The IBIS file that the Quartus Prime EDA Netlist Writer GUI generates contains models of both input and output termination, and is supported for IBIS model versions of 4.2 and later.

The Quartus Prime IBIS dynamic OCT IBIS model names end in `g50c_r50c`. For example : `sstl15i_ctnio_g50c_r50c`.

In the simulation tool, the IBIS model is attached to a buffer.

- When the buffer is assigned as an output, use the series termination r50c.
- When the buffer is assigned as an input, use the parallel termination g50c.

1.4.3.1. Generate Custom IBIS Models with the EDA Netlist Writer GUI

You can use the Quartus Prime EDA Netlist Writer GUI to generate custom IBIS models.

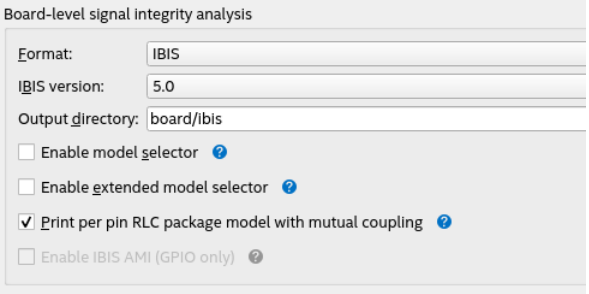
IBIS files that you generate with the EDA Netlist Writer automatically include the RLC values for your current target device.

Before generating the custom IBIS model, you can specify I/O constraints to define things like drive strength, enabling of clamping diodes for ESD protection, and other settings. The custom IBIS models that EDA Netlist Writer generates then reflect the I/O assignments.

To generate custom IBIS models with the EDA Netlist Writer GUI, follow these steps:

1. To specify the format, version, and output location of the generated model files, click **Assignments ➤ Settings ➤ EDA Tool Settings**.
2. Under **Board Level signal integrity analysis**, specify **IBIS** for the **Format**, the supported **IBIS version** that you want, and the location of the **Output directory** for the generated files.
3. Click **Assignments ➤ Device**. In the **Device** dialog box, click the **Device and Pin Options** button and review and specify any optional IBIS settings, as Board Level Signal Integrity Analysis Settings describes.
4. To run the EDA Netlist Writer to generate the custom IBIS model files, click **Processing ➤ Start ➤ Start EDA Netlist Writer**.

Figure 3. Board Level Signal Integrity Analysis Settings



Related Information

- Intel IBIS models
- Generating IBIS Output Files with the Quartus Prime Software
  In *Quartus Prime Help*
- AN 283: Simulating Intel Devices with IBIS Models

1.4.3.1.1. Board Level Signal Integrity Analysis Settings

The following settings are available for generation of custom IBIS models using the EDA Netlist Writer GUI:

Table 3. Board Level Signal Integrity Analysis Settings

Setting	Description
Format	Specifies **IBIS** as the format for output generation of custom IBIS models for board level signal integrity analysis in supported third-party tools.
IBIS version	Specifies the IBIS version **5.0** or **4.2** for the custom IBIS model you generate. Only version **5.0** is available for Agilex FPGA portfolio devices.
Output directory	Specifies the directory path for custom IBIS model generation. By default, the path is <*project*>`/board/ibis`.
Enable model selector	Enables the model selector feature that lists all the possible models for each I/O cell in the design. This setting is turned off by default.
Enable extended model selector	Enables the extended model selector feature. This setting is an extension of the **Enable model selector** setting. The extension lists additional models for I/O standards with Class I and II. This setting is turned off by default.
Enable per pin RLC package model with mutual coupling	Allows you to generate the per pin RLC package model with mutual coupling. The lumped RLC package model information appears in the IBIS output file. This setting is turned off by default except for Agilex FPGA portfolio devices.
Enable IBIS-AMI (GPIO Only)	Enables generation of IBIS-AMI models that you can use to model high-speed serial and parallel links that include transmitter and receiver equalization algorithms. This setting is available for only Agilex 5 devices and Agilex 7 M-Series devices. This setting is turned off by default for all applicable devices.

1.4.3.2. Customizing Downloaded or Installed IBIS Model Files for Agilex FPGA Portfolio Devices

The current Quartus Prime Pro Edition software installation includes the Intel FPGA IBIS models and IBIS writer script for Agilex FPGA portfolio devices in `/quartus/common/misc/ibis_writer/`. In addition, you can download the latest models (without the script) from: IBIS Models for Intel FPGA Devices.

The downloaded or installed IBIS models have the RLC package values set to only one particular device in each device family. These generic models describe the full set of models listed for and supported by each device family at: IBIS Models for Intel FPGA Devices.

Alternatively, you can generate custom IBIS models for Agilex FPGA portfolio devices by using the Quartus Prime EDA Netlist Writer GUI, as Generate Custom IBIS Models with the EDA Netlist Writer GUI describes.

To simulate your design with the most accurate model, you must customize the `.ibs` files to adjust the RLC values for accurate device package data using the IBIS Writer script. Use the script to customize the IBIS model file to match the values for your particular device package.

To use the IBIS Writer script, refer to the step by step instructions in the `/quartus/common/misc/ibis_writer/README.txt` file.

1.4.3.3. Customizing Downloaded IBIS Models for Stratix 10 Devices, Arria 10 Devices, and Cyclone 10 GX Devices

You can download the IBIS models for Stratix 10 devices, Arria 10 devices, and Cyclone 10 GX devices from the Intel website. You can use these IBIS models directly to perform early simulations of the I/O buffers that you expect to use in your design as part of a pre-layout analysis.

These downloaded IBIS models have the RLC package values set to only one particular device in each device family. These generic models describe only a certain set of models listed for each device at: IBIS Models for Intel FPGA Devices.

To simulate your design with the most accurate model, you must customize the `.ibs` files to adjust the RLC values for accurate device package data. Customize the IBIS model file to match the values for your particular device package by performing the following steps:

1. Download and expand the ZIP file (`.zip`) of the IBIS model for the device family you are using for your design. The `.zip` file contains the `.ibs` file along with an IBIS model user guide and a model data correlation report.
2. Download the Package RLC Values spreadsheet for the same device family.
3. Open the spreadsheet and locate the row that describes the device package used in your design.
4. From the package's **I/O** row, copy the minimum, maximum, and typical values of resistance, inductance, and capacitance for your device package.
5. Open the `.ibs` file in a text editor and locate the [Package] section of the file.
6. Overwrite the listed values copied with the values from the spreadsheet and save the file.

Related Information

IBIS Models for Intel FPGA Devices
For information about whether models for your selected device are available.

1.4.4. Design Simulation Using the Siemens EDA HyperLynx* Software

You must integrate IBIS models into board design simulations to accurately model timing and signal integrity.

The Siemens EDA HyperLynx* software is an industry standard tool for PCB analysis and simulation of high-speed designs. The HyperLynx software makes it easy to integrate IBIS models into simulations.

The HyperLynx software consists of the LineSim and BoardSim products. LineSim is an early simulation tool. Before any board routing takes place, you can use LineSim to simulate "what if" scenarios that assist in creating routing rules and defining board parameters.

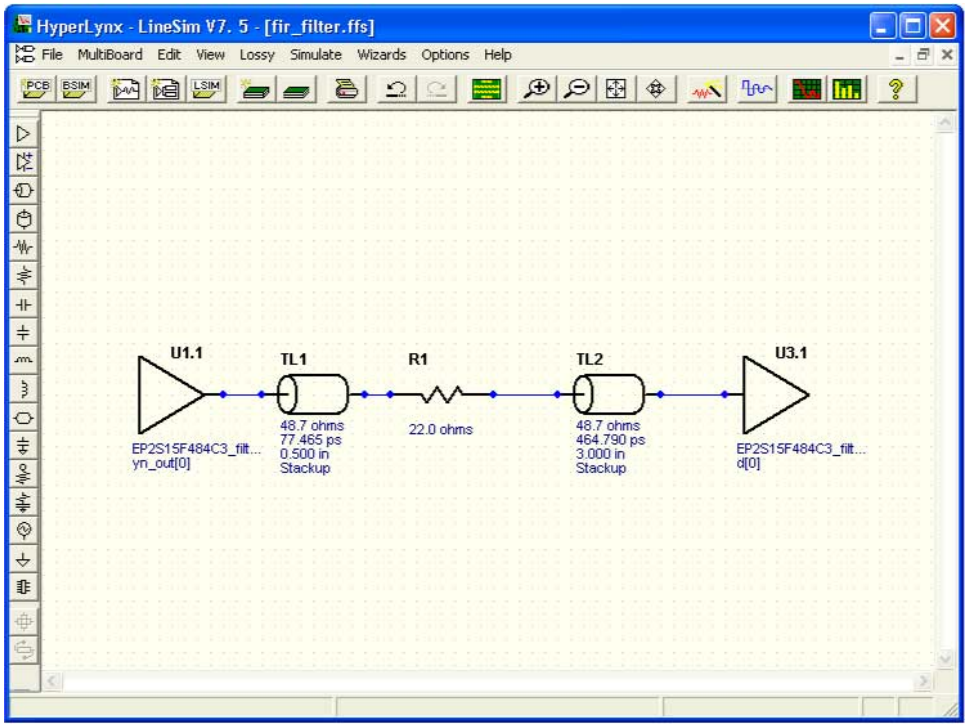
BoardSim is a post-layout tool that you can use to analyze existing board routing. You select one or more nets from a board layout file and BoardSim simulates those nets in a manner similar to LineSim. With board and routing parameters, and surrounding signal routing known, highly accurate simulations of the final fabricated PCB are possible.

This document section focuses on LineSim. Because the process of creating and running simulations is very similar for both LineSim and BoardSim, the details of IBIS model use in LineSim also apply to simulations in BoardSim.

You configure simulations in LineSim using a schematic GUI to create connections and topologies between I/O buffers, route trace segments, and termination components. LineSim provides two methods for creating routing schematics: cell-based and free-form. Cell-based schematics are based on fixed cells consisting of typical placements of buffers, trace impedances, and components. Parts of the grid-based cells are filled with the desired objects to create the topology. A topology in a cell-based schematic is limited by the available connections within and between the cells.

A more robust and expandable way to create a circuit schematic for simulation is to use the free-form schematic format in LineSim. The free-form schematic format makes it easy to place parts into any configuration and edit them as required. This section describes the use of IBIS models with free-form schematics, but the process is nearly identical for cell-based schematics.

Figure 4. HyperLynx LineSim Free-Form Schematic Editor



When you use HyperLynx software to perform simulations, you typically perform the following steps:

1. Create a new LineSim free-form schematic document and set up the board stackup for your PCB using the Stackup Editor. In this editor, specify board layer properties including layer thickness, dielectric constant, and trace width.
2. Create a circuit schematic for the net you want to simulate. The schematic represents all the parts of the routed net including source and destination I/O buffers, termination components, transmission line segments, and representations of impedance discontinuities such as vias or connectors.
3. Assign IBIS models to the source and destination I/O buffers to represent their behavior during operation.
4. Attach probes from the digital oscilloscope that is built in to LineSim to points in the circuit that you want to monitor during simulation. Typically, at least one probe is attached to the pin of a destination I/O buffer. For differential signals, you can attach a differential probe to both the positive and negative pins at the destination.
5. Configure and run the simulation. You can simulate a rising or falling edge and test the circuit under different drive strength conditions.
6. Interpret the results and make adjustments. Based on the waveforms captured in the digital oscilloscope, you can adjust anything in the circuit schematic to correct any signal integrity issues, such as overshoot or ringing. If necessary, you can make I/O assignment changes in the Quartus Prime software, regenerate the IBIS file with the IBIS Writer, and apply the updated IBIS model to the buffers in your HyperLynx software schematic.
7. Repeat the simulations and circuit adjustments until you are satisfied with the results.
8. When the operation of the net meets your design requirements, implement changes to your I/O assignments in the Quartus Prime software and optionally adjust your board routing constraints, component values, and placement to match the simulation.

For more information about HyperLynx software, including schematic creation, simulation setup, model usage, product support, licensing, and training, refer to the Siemens EDA webpage.

Related Information

eda.sw.siemens.com

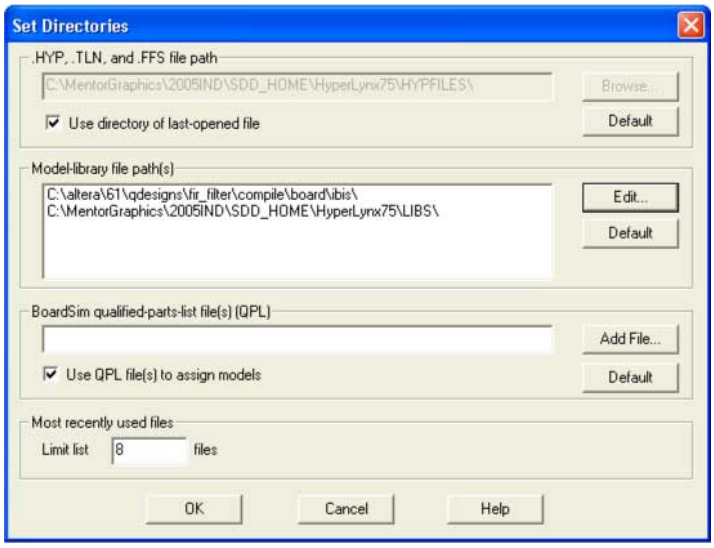
1.4.5. Configuring LineSim to Use Intel IBIS Models

You must configure LineSim to find and use the IBIS models for your design. To do this, add the location of your `.ibs` file or files to the LineSim Model Library search path. Next, you apply a selected model to a buffer in your schematic.

To add the Quartus Prime software's default IBIS model location, *<project directory>*`/board/ibis`, to the HyperLynx LineSim model library search path, perform the following steps in LineSim:

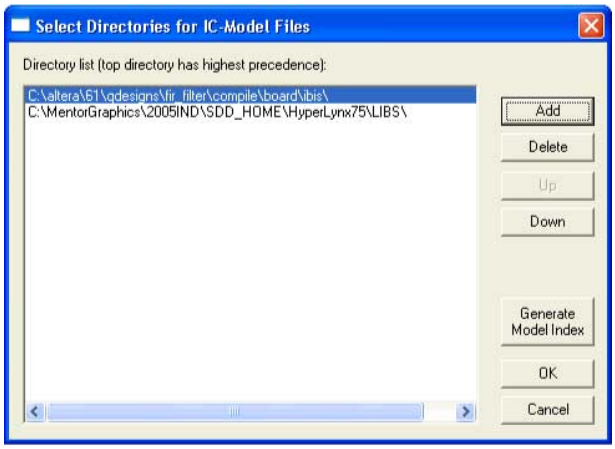
1. From the Options menu, click **Directories**. The **Set Directories** dialog box appears. The **Model-library file path(s)** list displays the order in which LineSim searches file directories for model files.

Figure 5. LineSim Set Directories Dialog Box



2. Click **Edit**. A dialog box appears where you can add directories and adjust the order in which LineSim searches them.

Figure 6. LineSim Select Directories Dialog Box



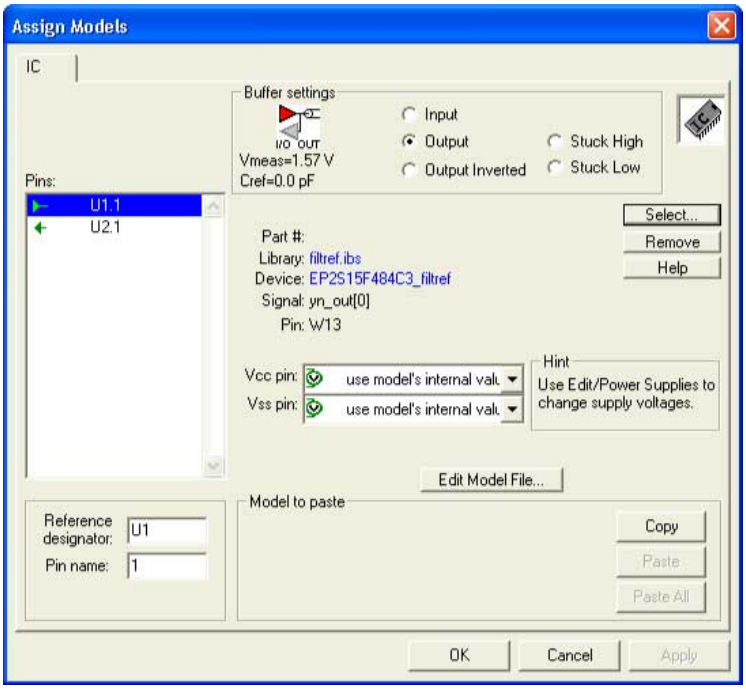
3. Click **Add**
4. Browse to the default IBIS model location, <*project directory*>`/board/ibis`. Click **OK**.
5. Click **Up** to move the IBIS model directory to the top of the list. Click **Generate Model Index** to update LineSim's model database with the models found in the added directory.
6. Click **OK**. The IBIS model directory for your project is added to the top of the Model-library file path(s) list.
7. To close the **Set Directories** dialog box, click **OK**.

1.4.6. Integrating Intel IBIS Models into LineSim Simulations

When the location for IBIS files has been set, you can assign the downloaded or generated IBIS models to the buffers in your schematic. To do this, perform the following steps:

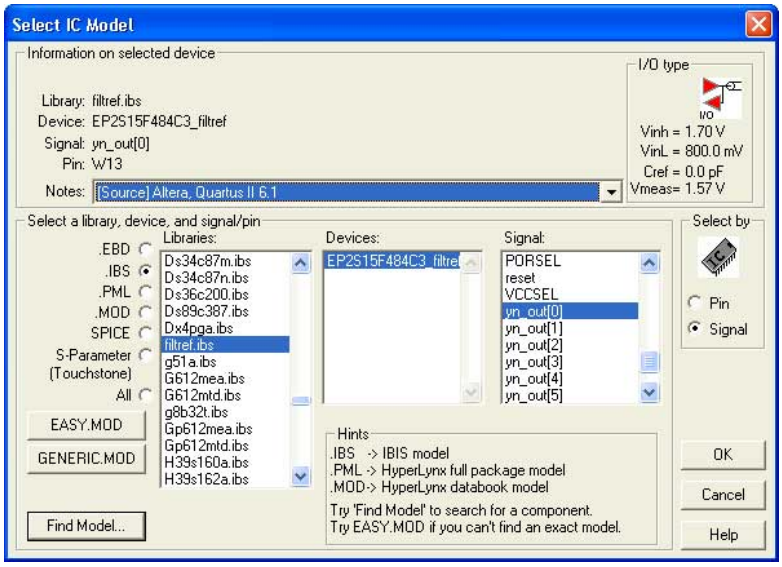
1. Double-click a buffer symbol in your schematic to open the **Assign Models** dialog box. You can also click **Assign Models** from the buffer symbol's right-click menu.

Figure 7. LineSim Assign Model Dialog Box



2. The pin of the buffer symbol you selected should be highlighted in the **Pins** list. If you want to assign a model to a different symbol or pin, select it from the list.
3. Click **Select**. The **Select IC Model** dialog box appears.

Figure 8. LineSim Select IC Model Dialog Box



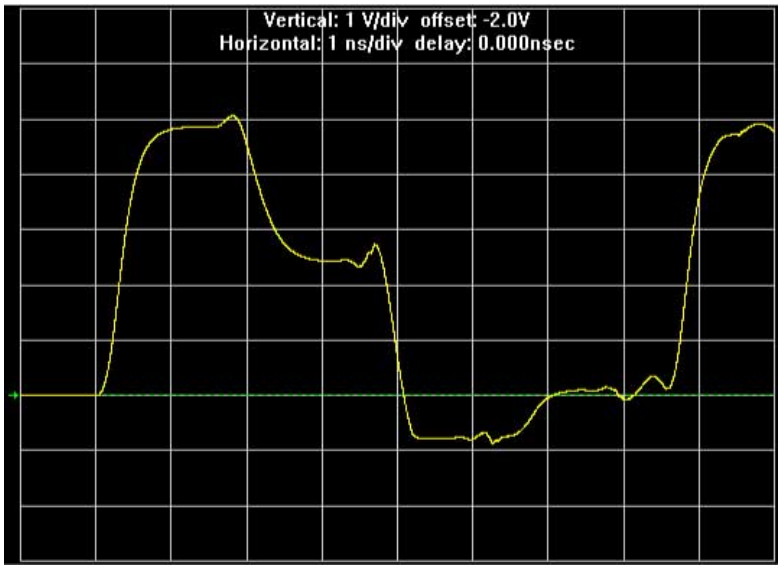
4. To filter the list of available libraries to display only IBIS models, select **.IBS**. Scroll through the **Libraries** list, and click the name of the library for your design. By default, this is *<project name>*`.ibs`.
5. The device for your design should be selected as the only item in the **Devices** list. If not, select your device from the list.
6. From the **Signal** list, select the name of the signal you want to simulate. You can also choose to select by device pin number.
7. Click **OK**. The **Assign Models** dialog box displays the selected `.ibs` file and signal.
8. If applicable to the signal you chose, adjust the buffer settings as required for the simulation.
9. Select and configure other buffer pins from the **Pins** list in the same manner.
10. Click **OK** when all I/O models are assigned.

1.4.7. Running and Interpreting LineSim Simulations

You can run any simulation and make adjustments to the I/O assignments or simulation parameters as required.

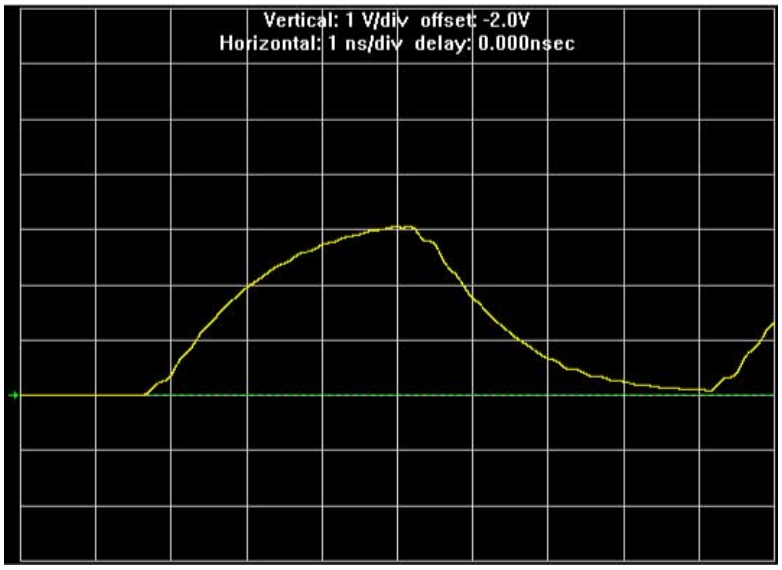
For example, if you see too much overshoot in the simulated signal at the destination buffer after running a simulation, you can adjust the drive strength I/O assignment setting to a lower value. Regenerate the `.ibs` file, and run the simulation again to verify whether the change fixes the problem.

Figure 9. Example of Overshoot in HyperLynx with IBIS Models



If you see a discontinuity or other anomalies at the destination, such as slow rise and fall times, adjust the termination scheme or termination component values. After making these changes, rerun the simulation to check whether your adjustments solved the problem. In this case, it is not necessary to regenerate the `.ibs` file.

Figure 10. Example of Signal Integrity Anomaly in HyperLynx with IBIS Models



For more information about board-level signal integrity, and to learn about ways to improve it with simple changes to your design, visit the Intel FPGA Signal & Power Integrity Support Center.

Related Information

Intel Signal & Power Integrity Center

1.5. Simulation with HSPICE Models

HSPICE decks are used to perform highly accurate simulations by describing the physical properties of all aspects of a circuit precisely. HSPICE decks describe I/O buffers, board components, and all the connections between them, as well as defining the parameters of the simulation to be run.

By their nature, HSPICE decks are highly customizable and require a detailed description of the circuit under simulation. The HSPICE decks generated by the Quartus Prime HSPICE Writer automatically include board components and topology defined in the Board Trace Model. Configure the board components and topology in the Pin Planner or in the **Board Trace Model** tab of the **Device and Pin Options** dialog box. All HSPICE decks generated by the Quartus Prime software include compensation for the double count problem. You can simulate with the default simulation parameters built in to the generated HSPICE decks or make adjustments to customize your simulation.

Related Information

The Double Counting Problem in HSPICE Simulations on page 22

1.5.1. Supported Devices and Signaling

The HSPICE Writer in the Quartus Prime software supports Arria, Cyclone, and Stratix devices for the creation of a board trace model in the Quartus Prime software for automatic inclusion in an HSPICE deck.

The HSPICE files include the board trace description you create in the Board Trace Model view in the Pin Planner or the **Board Trace Model** tab in the **Device and Pin Options** dialog box.

Note: Note that for Arria 10 devices, you may need to download the Encrypted HSPICE model from the Intel website.

Related Information

- I/O Management
  For information about how to use the **Enable Advanced I/O Timing** option and configure board trace models for the I/O standards used in your design.
- SPICE Models for Intel FPGAs
  For more information about the Encrypted HSPICE model.

1.5.2. Accessing HSPICE Simulation Kits

You can access the available HSPICE models with the Quartus Prime software's HSPICE Writer tool and also at the Spice Models for Intel Devices web page.

The Quartus Prime software HSPICE Writer tool removes many common sources of user error from the I/O simulation process. The HSPICE Writer tool automatically creates preconfigured I/O simulation spice decks that only require the addition of a user board model. All the difficult tasks required to configure the I/O modes and interpret the timing results are handled automatically by the HSPICE Writer tool.

Related Information

SPICE Models for Intel FPGAs
For more information about the Encrypted HSPICE model.

1.5.3. The Double Counting Problem in HSPICE Simulations

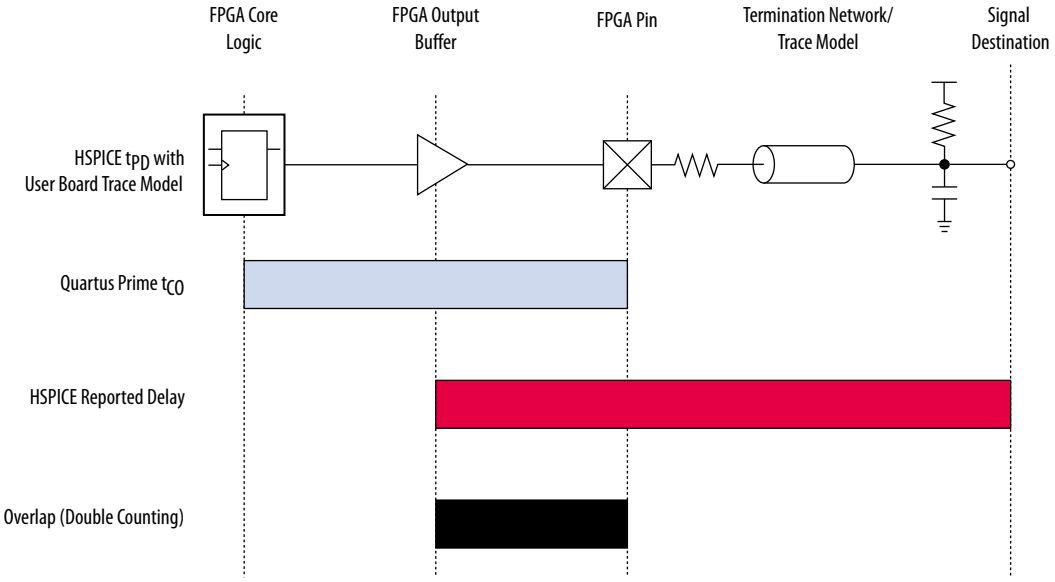
Simulating I/Os using accurate models is extremely helpful for finding and fixing FPGA I/O timing and board signal integrity issues before any boards are built. However, the usefulness of such simulations is directly related to the accuracy of the models used and whether the simulations are set up and performed correctly.

To ensure accuracy in models and simulations created for FPGA output signals you must consider the timing hand-off between t_{CO} timing in the Quartus Prime software and simulation-based board delay. If this hand-off is not handled correctly, the calculated delay could either count some of the delay twice or even miss counting some of the delay entirely.

1.5.3.1. Defining the Double Counting Problem

The double counting problem is inherent to the difference between the method to analyze output timing in the Quartus Prime software versus the method HSPICE models use. The timing analyzer tools in the Quartus Prime software measure delay timing for an output signal from the core logic of the FPGA design through the output buffer, ending at the FPGA pin with a default capacitive load or a specified value for the I/O standard you selected. This measurement is the t_{CO} timing variable.

Figure 11. Double Counting Problem



HSPICE models for board simulation measure t_{PD} (propagation delay) from an arbitrary reference point in the output buffer, through the device pin, out along the board routing, and ending at the signal destination.
If you add these two delays, the delay between the output buffer and the device pin appears twice in the calculation. A model or simulation that does not account for this double count creates overly pessimistic simulation results, because the double-counted delay can limit I/O performance artificially.

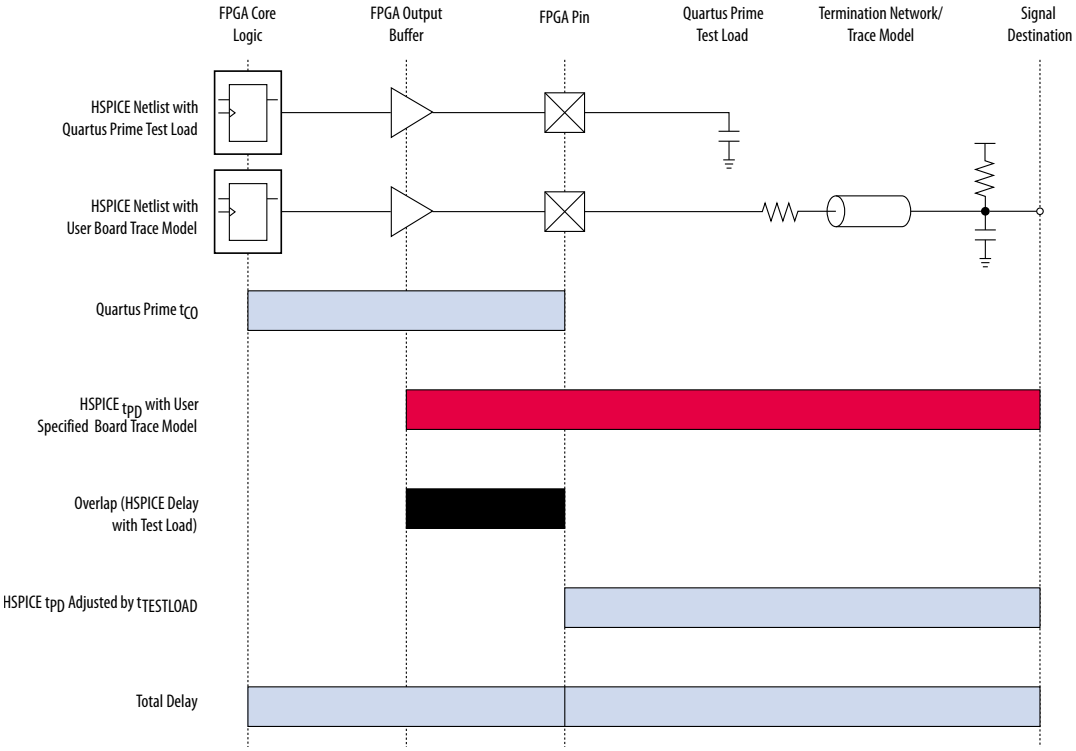
One approach to fix the problem is subtracting the overlap between t_{CO} and t_{PD} to account for the double count. However, this adjustment is not accurate, because each measurement considers a different load.

Note: Input signals do not exhibit this problem, because the HSPICE models for inputs stop at the FPGA pin instead of at the input buffer. In this case, adding the delays together produces an accurate measurement of delay timing.

1.5.3.2. The Solution to Double Counting

To adjust the measurements to account for the double-counting, the delay between the arbitrary point in the output buffer selected by the HSPICE model and the FPGA pin must be subtracted from either t_{CO} or t_{PD} before adding the results together. The subtracted delay must also be based on a common load between the two measurements. This is done by repeating the HSPICE model measurement, but with the same load used by the Quartus Prime software for the t_{CO} measurement.

Figure 12. Common Test Loads Used for Output Timing



With $t_{TESTLOAD}$ known, the total delay is calculated for the output signal from the FPGA logic to the signal destination on the board, accounting for the double count.

$$t_{delay} = t_{CO}+(t_{PD}-t_{TESTLOAD})$$

The preconfigured simulation files generated by the HSPICE Writer in the Quartus Prime software are designed to account for the double-counting problem based on this calculation automatically.

1.5.4. HSPICE Writer Tool Flow

This section includes information to help you get started using the Quartus Prime software HSPICE Writer tool. The information in this section assumes you have a basic knowledge of the standard Quartus Prime software design flow, such as project and assignment creation, compilation, and timing analysis.

1.5.4.1. Applying I/O Assignments

The first step in the HSPICE Writer tool flow is to configure the I/O standards and modes for each of the pins in your design properly. In the Quartus Prime software, these settings are represented by assignments that map I/O settings, such as pin selection, and I/O standard and drive strength, to corresponding signals in your design.

The Quartus Prime software provides multiple methods for creating these assignments:

- Using the Pin Planner
- Using the assignment editor
- Manually editing the `.qsf` file
- By making assignments in a scripted Quartus Prime flow using Tcl

1.5.4.2. Enabling HSPICE Writer Using Assignments

You can also use HSPICE Writer in conjunction with a scripted Tcl flow. To enable HSPICE Writer during a full compile, include the following lines in your Tcl script.

Enable HSPICE Writer

```
set_global_assignment -name EDA_BOARD_DESIGN_SIGNAL_INTEGRITY_TOOL \
     "HSPICE (Signal Integrity)"
set_global_assignment -name EDA_OUTPUT_DATA_FORMAT HSPICE \
     -section_id eda_board_design_signal_integrity
set_global_assignment -name EDA_NETLIST_WRITER_OUTPUT_DIR <output_directory> \
     -section_id eda_board_design_signal_integrity
```

As with command-line invocation, specifying the output directory is optional. If not specified, the output directory defaults to `board/hspice`.

1.5.4.3. Naming Conventions for HSPICE Files

HSPICE Writer automatically generates simulation files and names them using the following naming convention: *<device>_<pin #>_<pin_name>_<in/out>*`.sp`.

For bidirectional pins, two spice decks are produced; one with the I/O buffer configured as an input, and the other with the I/O buffer configured as an output.

The Quartus Prime software supports alphanumeric pin names that contain the underscore (_) and dash (-) characters. Any illegal characters used in file names are converted automatically to underscores.

Related Information

- Sample Output for I/O HSPICE Simulation Deck on page 35

- Sample Input for I/O HSPICE Simulation Deck on page 31

1.5.4.4. Invoking HSPICE Writer

After HSPICE Writer is enabled, the HSPICE simulation files are generated automatically each time the project is completely compiled. The Quartus Prime software also provides an option to generate a new set of simulation files without having to recompile manually. In the Processing menu, click **Start EDA Netlist Writer** to generate new simulation files automatically.

Note: You must perform both Analysis & Synthesis and Fitting on a design before invoking the HSPICE Writer tool.

1.5.4.5. Invoking HSPICE Writer from the Command Line

If you use a script-based flow to compile your project, you can create HSPICE model files by including the following commands in your Tcl script (`.tcl` file).

Create HSPICE Model Files

```
set_global_assignment -name EDA_BOARD_DESIGN_SIGNAL_INTEGRITY_TOOL \
     "HSPICE (Signal Integrity)"
set_global_assignment -name EDA_OUTPUT_DATA_FORMAT HSPICE \
     -section_ideda_board_design_signal_integrity
set_global_assignment -name EDA_NETLIST_WRITER_OUTPUT_DIR <output_directory> \
     -section_id eda_board_design_signal_integrity
```

The <*output_directory*> option specifies the location where HSPICE model files are saved. By default, the <*project directory*>`/board/hspice` directory is used.

Invoke HSPICE Writer

To invoke the HSPICE Writer tool through the command line, type:

```
quartus_eda.exe <project_name> --board_signal_integrity=on --format=HSPICE \
--output_directory=<output_directory>
```

<*output_directory*> specifies the location where the tool writes the generated spice decks, relative to the design directory. This is an optional parameter and defaults to `board/hspice`.

1.5.4.6. Customizing Automatically Generated HSPICE Decks

HSPICE models generated by the HSPICE Writer can be used for simulation as generated.

A default board description is included, and a default simulation is set up to measure rise and fall delays for both input and output simulations, which compensates for the double counting problem. However, Intel recommends that you customize the board description to more accurately represent your routing and termination scheme.

The sample board trace loading in the generated HSPICE model files must be replaced by your actual trace model before you can run a correct simulation. To do this, open the generated HSPICE model files for all pins you want to simulate and locate the following section.

Sample Board Trace Section

```
* I/O Board Trace and Termination Description
* - Replace this with your board trace and termination description
```

You must replace the example load with a load that matches the design of your PCB board. This includes a trace model, termination resistors, and, for output simulations, a receiver model. The spice circuit node that represents the pin of the FPGA package is called **pin**. The node that represents the far pin of the external device is called **load-in** (for output SPICE decks) and **source-in** (for input SPICE decks).

For an input simulation, you must also modify the stimulus portion of the spice file. The section of the file that must be modified is indicated in the following comment block.

Sample Source Stimulus Section

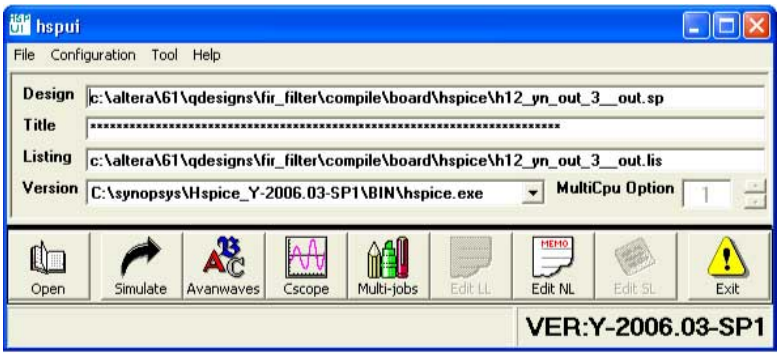
```
* Sample source stimulus placeholder
*  - Replace this with your I/O driver model
```

Replace the sample stimulus model with a model for the device that drives the FPGA.

1.5.5. Running an HSPICE Simulation

Because simulation parameters are configured directly in the HSPICE model files, running a simulation requires only that you open an HSPICE file in the HSPICE user interface and start the simulation.

Figure 13. HSPICE User Interface Window



Click **Open** and browse to the location of the HSPICE model files generated by the Quartus Prime HSPICE Writer. The default location for HSPICE model files is *<project directory>*`/board/hspice`. Select the `.sp` file generated by the HSPICE Writer for the signal you want to simulate. Click **OK**.

To run the simulation, click **Simulate**. The status of the simulation is displayed in the window and saved in an `.lis` file with the same name as the `.sp` file when the simulation is complete. Check the `.lis` file if an error occurs during the simulation requiring a change in the `.sp` file to fix.

1.5.6. Interpreting the Results of an Output Simulation

By default, the automatically generated output simulation spice decks are set up to measure three delays for both rising and falling transitions. Two of the measurements, `tpd_rise` and `tpd_fall`, measure the double-counting corrected delay from the FPGA pin to the load pin. To determine the complete clock-edge to load-pin delay, add these numbers to the Quartus Prime software reported default loading t_{CO} delay.

The remaining four measurements, `tpd_uncomp_rise`, `tpd_uncomp_fall`, `t_dblcnt_rise`, and `t_dblcnt_fall`, are required for the double-counting compensation process and are not required for further timing usage.

Related Information

Simulation Analysis on page 35

1.5.7. Interpreting the Results of an Input Simulation

By default, the automatically generated input simulation SPICE decks are set up to measure delays from the source's driver pin to the FPGA's input pin for both rising and falling transitions.

The propagation delay is reported by HSPICE measure statements as `tpd_rise` and `tpd_fall`. To determine the complete source driver pin-to-FPGA register delay, add these numbers to the Quartus Prime software reported T_H and T_{SU} input timing numbers.

1.5.8. Viewing and Interpreting Tabular Simulation Results

The `.lis` file stores the collected simulation data in tabular form. The default simulation configured by the HSPICE Writer produces delay measurements for rising and falling transitions on both input and output simulations.

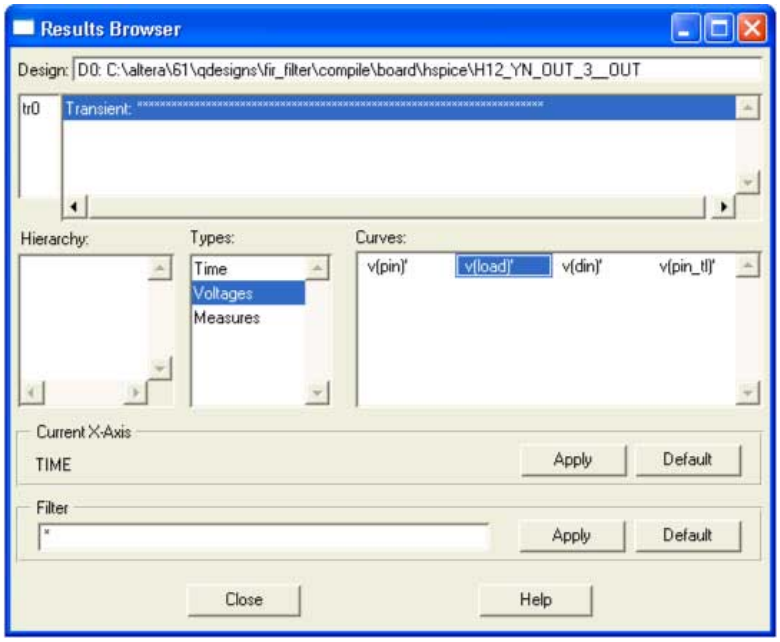
These measurements are found in the `.lis` file and named `tpd_rise` and `tpd_fall`. For output simulations, these values are already adjusted for the double count. To determine the complete delay from the FPGA logic to the load pin, add either of these measurements to the Quartus Prime t_{CO} delay. For input simulations, add either of these measurements to the Quartus Prime t_{SU} and t_H delay values to calculate the complete delay from the far end stimulus to the FPGA logic. Other values found in the `.lis` file, such as `tpd_uncomp_rise`, `tpd_uncomp_fall`, `t_dblcnt_rise`, and `t_dblcnt_fall`, are parts of the double count compensation calculation. These values are not necessary for further analysis.

1.5.9. Viewing Graphical Simulation Results

You can view the results of the simulation quickly as a graphical waveform display using the AvanWaves viewer included with HSPICE. With the default simulation configured by the HSPICE Writer, you can view the simulated waveforms at both the source and destination in input and output simulations.

To see the waveforms for the simulation, in the HSPICE user interface window, click **AvanWaves**. The AvanWaves viewer opens and displays the **Results Browser**.

Figure 14. HSPICE AvanWaves Results Browser



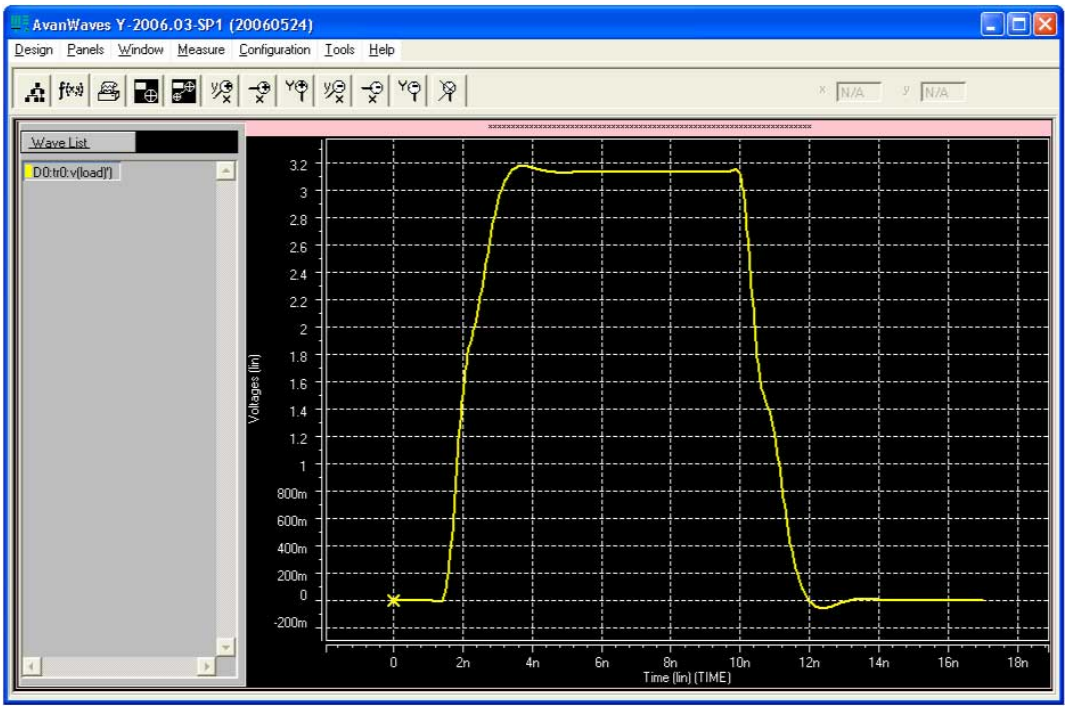
The **Results Browser** lets you select which waveform to view quickly in the main viewing window. If multiple simulations are run on the same signal, the list at the top of the **Results Browser** displays the results of each simulation. Click the simulation description to select which simulation to view. By default, the descriptions are derived from the first line of the HSPICE file, so the description might appear as a line of asterisks.

Select the type of waveform to view, by performing the following steps:

1. To see the source and destination waveforms with the default simulation, from the **Types** list, select **Voltages**.
2. On the **Curves** list, double-click the waveform you want to view. The waveform appears in the main viewing window.

You can zoom in and out and adjust the view as desired.

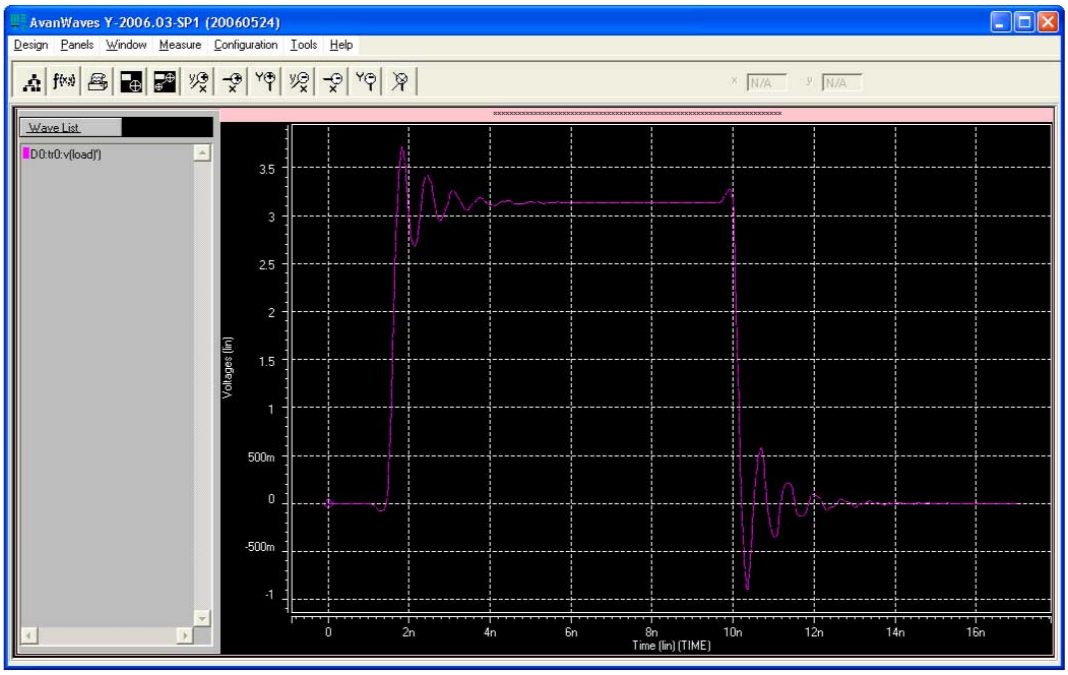
Figure 15. AvanWaves Waveform Viewer



1.5.10. Making Design Adjustments Based on HSPICE Simulations

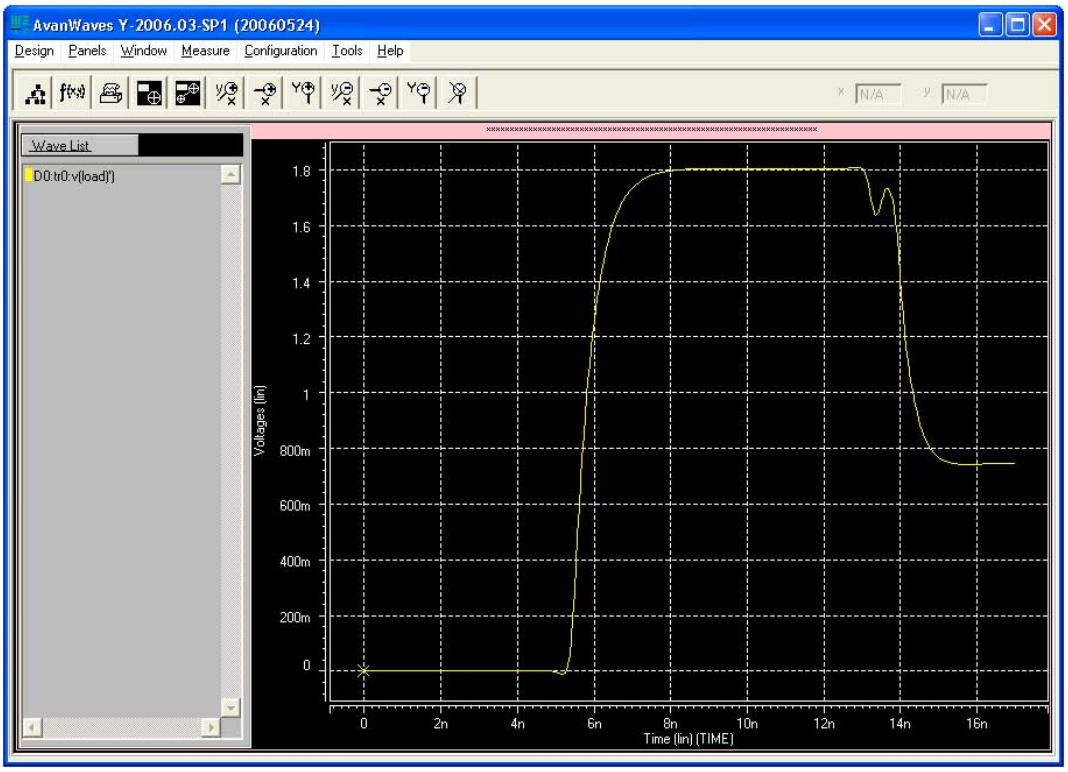
Based on the results of your simulations, you can make adjustments to the I/O assignments or simulation parameters if required. For example, after you run a simulation and see overshoot or ringing in the simulated signal at the destination buffer, you can adjust the drive strength I/O assignment setting to a lower value. Regenerate the HSPICE deck, and run the simulation again to verify that the change fixed the problem.

Figure 16. Example of Overshoot in the AvanWaves Waveform Viewer



If there is a discontinuity or any other anomalies at the destination, adjust the board description in the Quartus Prime Board Trace Model, or in the generated HSPICE model files to change the termination scheme or adjust termination component values. After making these changes, regenerate the HSPICE files if necessary, and rerun the simulation to verify whether your adjustments solved the problem.

Figure 17. Example of Signal Integrity Anomaly in the AvanWaves Waveform Viewer



For more information about board-level signal integrity and to learn about ways to improve it with simple changes to your FPGA design, visit the Intel Signal & Power Integrity Center

Related Information

Intel Signal & Power Integrity Center

1.5.11. Sample Input for I/O HSPICE Simulation Deck

The following sections examine a typical HSPICE simulation spice deck for an I/O of type input. Each section presents the simulation file one block at a time.

1.5.11.1. Header Comment

The first block of an input simulation spice deck is the header comment. The purpose of this block is to provide an easily readable summary of how the simulation file has been automatically configured by the Quartus Prime software.

This block has two main components: The first component summarizes the I/O configuration relevant information such as device, speed grade, and so on. The second component specifies the exact test condition that the Quartus Prime software assumes for the given I/O standard.

Sample Header Comment Block

```
* Intel Quartus Prime HSPICE Writer I/O Simulation Deck*
* This spice simulation deck was automatically generated by
* Quartus for the following IO settings:
*
* Device: EP2S60F1020C3
* Speed Grade: C3
* Pin: AA4 (out96)
* Bank: IO Bank 6 (Row I/O)
* I/O Standard: LVTTL, 12mA
* OCT: Off
*
* Intel Quartus Prime's default I/O timing delays assume the following slow
* corner simulation conditions.
*
* Specified Test Conditions For Intel Quartus Prime Tco
* Temperature: 85C (Slowest Temperature Corner)
* Transistor Model: TT (Typical Transistor Corner)
* Vccn: 3.135V (Vccn_min = Nominal - 5%)
* Vccpd: 2.97V (Vccpd_min = Nominal - 10%)
* Load: No Load
* Vtt: 1.5675V (Voltage reference is Vccn/2)
*
* Note: The I/O transistors are specified to operate at least as
* fast as the TT transistor corner, actual production
* devices can be as fast as the FF corner. Any simulations
* for hold times should be conducted using the fast process
* corner with the following simulation conditions.
* Temperature: 0C (Fastest Commercial Temperature Corner **)
* Transistor Model: FF (Fastest Transistor Corner)
* Vccn: 1.98V (Vccn_hold = Nominal + 10%)
* Vccpd: 3.63V (Vccpd_hold = Nominal + 10%)
* Vtt: 0.95V (Vtt_hold = Vccn/2 - 40mV)
* Vcc: 1.25V (Vcc_hold = Maximum Recommended)
* Package Model: Short-circuit from pad to pin (no parasitics)
*
* Warnings:
```

1.5.11.2. Simulation Conditions

The simulation conditions block loads the appropriate process corner models for the transistors. This condition is automatically set up for the slow timing corner and is modified only if other simulation corners are desired.

Simulation Conditions Block

```
* Process Settings
.options brief
.inc 'sii_tt.inc' * TT process corner
```

1.5.11.3. Simulation Options

The simulation options block configures the simulation temperature and configures HSPICE with typical simulation options.

Simulation Options Block

```
* Simulation Options
.options brief=0
.options badchr co=132 scale=1e-6 acct ingold=2 nomod dv=1.0
+ dcstep=1 absv=1e-3 absi=1e-8 probe csdf=2 accurate=1
+ converge=1
.temp 85
```

Note: For a detailed description of these options, consult your *HSPICE* manual.

1.5.11.4. Constant Definition

The constant definition block of the simulation file instantiates the voltage sources that controls the configuration modes of the I/O buffer.

Constant Definition Block

```
* Constant Definition
voeb oeb 0 vc * Set to 0 to enable buffer output
vopdrain opdrain 0 0 * Set to vc to enable open drain
vrambh rambh 0 0 * Set to vc to enable bus hold
vrpullup rpullup 0 0 * Set to vc to enable weak pullup
vpcdp5 rpcdp5 0 rp5 * Set the IO standard
vpcdp4 rpcdp4 0 rp4
vpcdp3 rpcdp3 0 rp3
vpcdp2 rpcdp2 0 rp2
vpcdp1 rpcdp1 0 rp1
vpcdp0 rpcdp0 0 rp0
vpcdn4 rpcdn4 0 rn4
vpcdn3 rpcdn3 0 rn3
vpcdn2 rpcdn2 0 rn2
vpcdn1 rpcdn1 0 rn1
vpcdn0 rpcdn0 0 rn0
vdin din 0 0
```

Where:

- Voltage source `voeb` controls the output enable of the buffer and is set to disabled for inputs.
- `vopdrain` controls the open drain mode for the I/O.
- `vrambh` controls the bus hold circuitry in the I/O.
- `vrpullup` controls the weak pullup.
- The next 11 voltages sources control the I/O standard of the buffer and are configured through a later library call.
- `vdin` is not used on input pins because it is the data pin for the output buffer.

1.5.11.5. Buffer Netlist

The buffer netlist block of the simulation spice deck loads all the load models required for the corresponding input pin.

Buffer Netlist Block

```
* IO Buffer Netlist
.include 'vio_buffer.inc'
```

1.5.11.6. Drive Strength

The drive strength block of the simulation SPICE deck loads the configuration bits necessary to configure the I/O into the proper I/O standard and drive strengths.

Although these settings are not relevant to an input buffer, they are provided to allow the SPICE deck to be modifiable to support bidirectional simulations.

Drive Strength Block

```
* Drive Strength Settings
.lib ‘drive_select_hio.lib’ 3p3ttl_12ma
```

1.5.11.7. I/O Buffer Instantiation

The I/O buffer instantiation block of the simulation SPICE deck instantiates the necessary power supplies and I/O model components that are necessary to simulate the given I/O.

I/O Buffer Instantiation

```
I/O Buffer Instantiation
* Supply Voltages Settings
.param vcn=3.135
.param vpd=2.97
.param vc=1.15
* Instantiate Power Supplies|
vvcc vcc 0 vc * FPGA core voltage
vvss vss 0 0 * FPGA core ground
vvccn vccn 0 vcn * IO supply voltage
vvssn vssn 0 0 * IO ground
vvccpd vccpd 0 vpd * Pre-drive supply voltage
* Instantiate I/O Buffer
xvio_buf din oeb opdrain die rambh
+ rpcdn4 rpcdn3 rpcdn2 rpcdn1 rpcdn0
+ rpcdp5 rpcdp4 rpcdp3 rpcdp2 rpcdp1 rpcdp0
+ rpullup vccn vccpd vcpad0 vio_buf
* Internal Loading on Pad
* - No loading on this pad due to differential buffer/support
* circuitry
* I/O Buffer Package Model
* - Single-ended I/O standard on a Row I/O
.lib ‘lib/package.lib’ hio
xpkg die pin hio_pkg
```

1.5.11.8. Board Trace and Termination

The board trace and termination block of the simulation SPICE deck is provided only as an example. Replace this block with your own board trace and termination models.

Board Trace and Termination Block

```
* I/O Board Trace and Termination Description
* - Replace this with your board trace and termination description
wtline pin vssn load vssn N=1 L=1 RLGCMODEL=tlinemodel
.MODEL tlinemodel W MODELTYPE=RLGC N=1 Lo=7.13n Co=2.85p
Rterm2 load vssn 1x
```

1.5.11.9. Stimulus Model

The stimulus model block of the simulation spice deck is provided only as a place holder example. Replace this block with your own stimulus model. Options for this include an IBIS or HSPICE model, among others.

Stimulus Model Block

```
* Sample source stimulus placeholder
* - Replace this with your I/O driver model
Vsource source 0 pulse(0 vcn 0s 0.4ns 0.4ns 8.5ns 17.4ns)
```

1.5.11.10. Simulation Analysis

The simulation analysis block of the simulation file is configured to measure the propagation delay from the source to the FPGA pin. Both the source and end point of the delay are referenced against the 50% V_{CCN} crossing point of the waveform.

Simulation Analysis Block

```
* Simulation Analysis Setup
* Print out the voltage waveform at both the source and the pin
.print tran v(source) v(pin)
.tran 0.020ns 17ns
* Measure the propagation delay from the source pin to the pin
* referenced against the 50% voltage threshold crossing point
.measure TRAN tpd_rise TRIG v(source) val='vcn*0.5' rise=1
+ TARG v(pin) val ='vcn*0.5' rise=1
.measure TRAN tpd_fall TRIG v(source) val='vcn*0.5' fall=1
+ TARG v(pin) val ='vcn*0.5' fall=1
```

1.5.12. Sample Output for I/O HSPICE Simulation Deck

A typical HSPICE simulation SPICE deck for an I/O-type output has several sections. Each section presents the simulation file one block at a time.

1.5.12.1. Header Comment

The first block of an output simulation SPICE deck is the header comment. The purpose of this block is to provide a readable summary of how the simulation file has been automatically configured by the Quartus Prime software.

This block has two main components:

- The first component summarizes the I/O configuration relevant information such as device, speed grade, and so on.
- The second component specifies the exact test condition that the Quartus Prime software assumes when generating t_{CO} delay numbers. This information is used as part of the double-counting correction circuitry contained in the simulation file.

The SPICE decks are preconfigured to calculate the slow process corner delay but can also be used to simulate the fast process corner as well. The fast corner conditions are listed in the header under the notes section.

The final section of the header comment lists any warning messages that you must consider when you use the SPICE decks.

Header Comment Block

```
* Intel Quartus Prime HSPICE Writer I/O Simulation Deck
*
* This spice simulation deck was automatically generated by
* Intel Quartus Prime for the following IO settings:
*
* Device: EP2S60F1020C3
* Speed Grade: C3
* Pin: AA4 (out96)
* Bank: IO Bank 6 (Row I/O)
* I/O Standard: LVTTL, 12mA
* OCT: Off
*
* Quartus' default I/O timing delays assume the following slow
* corner simulation conditions.
* Specified Test Conditions For Intel Quartus Prime Tco
```

```
* Temperature: 85C (Slowest Temperature Corner)
* Transistor Model: TT (Typical Transistor Corner)
* Vccn: 3.135V (Vccn_min = Nominal - 5%)
* Vccpd: 2.97V (Vccpd_min = Nominal - 10%)
* Load: No Load
* Vtt: 1.5675V (Voltage reference is Vccn/2)
* For C3 devices, the TT transistor corner provides an
* approximation for worst case timing. However, for functionality
* simulations, it is recommended that the SS corner be simulated
* as well.
*
* Note: The I/O transistors are specified to operate at least as
* fast as the TT transistor corner, actual production
* devices can be as fast as the FF corner. Any simulations
* for hold times should be conducted using the fast process
* corner with the following simulation conditions.
* Temperature: 0C (Fastest Commercial Temperature Corner **)
* Transistor Model: FF (Fastest Transistor Corner)
* Vccn: 1.98V (Vccn_hold = Nominal + 10%)
* Vccpd: 3.63V (Vccpd_hold = Nominal + 10%)
* Vtt: 0.95V (Vtt_hold = Vccn/2 - 40mV)
* Vcc: 1.25V (Vcc_hold = Maximum Recommended)
* Package Model: Short-circuit from pad to pin
* Warnings:
```

1.5.12.2. Simulation Conditions

The simulation conditions block loads the appropriate process corner models for the transistors. This condition is automatically set up for the slow timing corner and must be modified only if other simulation corners are desired.

Simulation Conditions Block

```
* Process Settings
.options brief
.inc ‘sii_tt.inc’ * typical-typical process corner
```

Note: Two separate corners cannot be simulated at the same time. Instead, simulate the base case using the Quartus corner as one simulation and then perform a second simulation using the desired customer corner. The results of the two simulations can be manually added together.

1.5.12.3. Simulation Options

The simulation options block configures the simulation temperature and configures HSPICE with typical simulation options.

Simulation Options Block

```
* Simulation Options
.options brief=0
.options badchr co=132 scale=1e-6 acct ingold=2 nomod dv=1.0
+ dcstep=1 absv=1e-3 absi=1e-8 probe csdf=2 accurate=1
+ converge=1
.temp 85
```

Note: For a detailed description of these options, consult your *HSPICE* manual.

1.5.12.4. Constant Definition

The constant definition block of the output simulation SPICE deck instantiates the voltage sources that controls the configuration modes of the I/O buffer.

Constant Definition Block

```
* Constant Definition
voeb oeb 0 0 * Set to 0 to enable buffer output
vopdrain opdrain 0 0 * Set to vc to enable open drain
vrambh rambh 0 0 * Set to vc to enable bus hold
vrpullup rpullup 0 0 * Set to vc to enable weak pullup
vpci rpci 0 0 * Set to vc to enable pci mode
vpcdp4 rpcdp4 0 rp4 * These control bits set the IO standard
vpcdp3 rpcdp3 0 rp3
vpcdp2 rpcdp2 0 rp2
vpcdp1 rpcdp1 0 rp1
vpcdp0 rpcdp0 0 rp0
vpcdn4 rpcdn4 0 rn4
vpcdn3 rpcdn3 0 rn3
vpcdn2 rpcdn2 0 rn2
vpcdn1 rpcdn1 0 rn1
vpcdn0 rpcdn0 0 rn0
vdin din 0 pulse(0 vc 0s 0.2ns 0.2ns 8.5ns 17.4ns)
```

Where:

- Voltage source `voeb` controls the output enable of the buffer.
- `vopdrain` controls the open drain mode for the I/O.
- `vrambh` controls the bus hold circuitry in the I/O.
- `vrpullup` controls the weak pullup.
- `vpci` controls the PCI clamp.
- The next ten voltage sources control the I/O standard of the buffer and are configured through a later library call.
- `vdin` is connected to the data input of the I/O buffer.
- The edge rate of the input stimulus is automatically set to the correct value by the Quartus Prime software.

1.5.12.5. I/O Buffer Netlist

The I/O buffer netlist block loads all of the models required for the corresponding pin. These include a model for the I/O output buffer, as well as any loads that might be present on the pin.

I/O Buffer Netlist Block

```
*IO Buffer Netlist
.include ‘hio_buffer.inc’
.include ‘lvds_input_load.inc’
.include ‘lvds_oct_load.inc’
```

1.5.12.6. Drive Strength

The drive strength block of the simulation spice deck loads the configuration bits for configuring the I/O to the proper I/O standard and drive strength. These options are set by the HSPICE Writer tool and are not changed for expected use.

Drive Strength Block

```
* Drive Strength Settings
.lib ‘drive_select_hio.lib’ 3p3ttl_12ma
```

1.5.12.7. Slew Rate and Delay Chain

Stratix and Cyclone devices have sections for configuring the slew rate and delay chain settings.

Slew Rate and Delay Chain Settings

```
* Programmable Output Delay Control Settings
.lib 'lib/output_delay_control.lib' no_delay
* Programmable Slew Rate Control Settings
.lib 'lib/slew_rate_control.lib' slow_slow
```

1.5.12.8. I/O Buffer Instantiation

The I/O buffer instantiation block of the output simulation spice deck instantiates the necessary power supplies and I/O model components that are necessary to simulate the given I/O.

I/O Buffer Instantiation Block

```
* I/O Buffer Instantiation
* Supply Voltages Settings
.param vcn=3.135
.param vpd=2.97
.param vc=1.15
* Instantiate Power Supplies
vvcc vcc 0 vc * FPGA core voltage
vvss vss 0 0 * FPGA core ground
vvccn vccn 0 vcn * IO supply voltage
vvssn vssn 0 0 * IO ground
vvccpd vccpd 0 vpd * Pre-drive supply voltage
* Instantiate I/O Buffer
xhio_buf din oeb opdrain die rambh
+ rpcdn4 rpcdn3 rpcdn2 rpcdn1 rpcdn0
+ rpcdp4 rpcdp3 rpcdp2 rpcdp1 rpcdp0
+ rpullup vccn vccpd vcpad0 hio_buf
* Internal Loading on Pad
* - This pad has an LVDS input buffer connected to it, along
* with differential OCT circuitry. Both are disabled but
* introduce loading on the pad that is modeled below.
xlvds_input_load die vss vccn lvds_input_load
xlvds_oct_load die vss vccpd vccn vcpad0 vccn lvds_oct_load
* I/O Buffer Package Model
* - Single-ended I/O standard on a Row I/O
.lib 'lib/package.lib' hio
xpkg die pin hio_pkg
```

1.5.12.9. Board and Trace Termination

The board trace and termination block of the simulation SPICE deck is provided only as an example. Replace this block with your specific board loading models.

Board Trace and Termination Block

```
* I/O Board Trace And Termination Description
* - Replace this with your board trace and termination description
wtline pin vssn load vssn N=1 L=1 RLGCMODEL=tlinemodel
.MODEL tlinemodel W MODELTYPE=RLGC N=1 Lo=7.13n Co=2.85p
Rterm2 load vssn 1x
```

1.5.12.10. Double-Counting Compensation Circuitry

The double-counting compensation circuitry block of the simulation SPICE deck instantiates a second I/O buffer that is used to measure double-counting. The buffer is configured identically to the user I/O buffer but is connected to the Quartus Prime software test load. The simulated delay of this second buffer can be interpreted as the amount of double-counting between the Quartus Prime software and HSPICE Writer simulated results.

As the amount of double-counting is constant for a given I/O standard on a given pin, consider separating the double-counting circuitry from the simulation file. In doing so, you can perform any number of I/O simulations while referencing the delay only once.

(Part of)Double-Counting Compensation Circuitry Block

```
* Double Counting Compensation Circuitry
*
* The following circuit is designed to calculate the amount of
* double counting between Intel Quartus Prime and the HSPICE models. If
* you have not changed the default simulation temperature or
* transistor corner this spice deck automatically compensates the double
counting.
* In the event you wish to
* simulate an IO at a different temperature or transistor corner
* you need to remove this section of code and manually
* account for double counting. A description of Intel's
* recommended procedure for this process can be found in the
* Intel Quartus Prime HSPICE Writer AppNote.
* Supply Voltages Settings
.param vcn_tl=3.135
.param vpd_tl=2.97
* Test Load Constant Definition
vopdrain_tl opdrain_tl 0 0
vrambh_tl rambh_tl 0 0
vrpullup_tl rpullup_tl 0 0
* Instantiate Power Supplies
vvccn_tl vccn_tl 0 vcn_tl
vvssn_tl vssn_tl 0 0
vvccpd_tl vccpd_tl 0 vpd_tl
* Instantiate I/O Buffer
xhio_testload din oeb opdrain_tl die_tl rambh_tl
+ rpcdn4 rpcdn3 rpcdn2 rpcdn1 rpcdn0
+ rpcdp4 rpcdp3 rpcdp2 rpcdp1 rpcdp0
+ rpullup_tl vccn_tl vccpd_tl vcpad0_tl hio_buf
* Internal Loading on Pad
xlvds_input_testload die_tl vss vccn_tl lvds_input_load
xlvds_oct_testload die_tl vss vccpd_tl vccn_tl vcpad0_tl vccn_tl
lvds_oct_load
* I/O Buffer Package Model
* - Single-ended I/O standard on a Row I/O
.lib 'lib/package.lib' hio
xpkg die pin hio_pkg
* Default Intel Test Load
* - 3.3V LVTTL default test condition is an open load
```

Related Information

The Double Counting Problem in HSPICE Simulations on page 22

1.5.12.11. Simulation Analysis

The simulation analysis block is set up to measure double-counting corrected delays. This is accomplished by measuring the uncompensated delay of the I/O buffer when connected to the user load, and when subtracting the simulated amount of double-counting from the test load I/O buffer.

Simulation Analysis Block

```
* Simulation Analysis Setup
*Print out the voltage waveform at both the pin and far end load
.print tran v(pin) v(load)
.tran 0.020ns 17ns
* Measure the propagation delay to the load pin. This value
* includes some double counting with Intel Quartus Prime's Tco
.measure TRAN tpd_uncomp_rise TRIG v(din) val='vc*0.5' rise=1+ TARG v(load)
val='vcn*0.5' rise=1
.measure TRAN tpd_uncomp_fall TRIG v(din) val='vc*0.5' fall=1
+ TARG v(load) val='vcn*0.5' fall=1
* The test load buffer can calculate the amount of double counting
.measure TRAN t_dblcnt_rise TRIG v(din) val='vc*0.5' rise=1
+ TARG v(pin_tl) val='vcn_tl*0.5' rise=1
.measure TRAN t_dblcnt_fall TRIG v(din) val='vc*0.5' fall=1
+ TARG v(pin_tl) val='vcn_tl*0.5' fall=1
* Calculate the true propagation delay by subtraction
.measure TRAN tpd_rise PARAM='tpd_uncomp_rise-t_dblcnt_rise'
.measure TRAN tpd_fall PARAM='tpd_uncomp_fall-t_dblcnt_fall'
```

1.5.13. Advanced Topics

The information in this section describes some of the more advanced topics and methods employed when setting up and running HSPICE simulation files.

1.5.13.1. PVT Simulations

The automatically generated HSPICE simulation files are set up to simulate the slow process corner using low voltage, high temperature, and slow transistors. To ensure a fully robust link, Intel recommends that you run simulations over all process corners.

To perform process, voltage, and temperature (PVT) simulations, manually modify the spice decks in a two step process:

1. Remove the double-counting compensation circuitry from the simulation file. This is required as the amount of double-counting is dependent upon how the Quartus Prime software calculates delays and is not based on which PVT corner is being simulated. By default, the Quartus Prime software provides timing numbers using the slow process corner.
2. Select the proper corner for the PVT simulation by setting the correct HSPICE temperature, changing the supply voltage sources, and loading the correct transistor models.

A more detailed description of HSPICE process corners can be found in the family-specific HSPICE model documentation.

Related Information

Accessing HSPICE Simulation Kits on page 21

1.5.13.2. Hold Time Analysis

Intel recommends performing worst-case hold time analysis using the fast corner models, which use fast transistors, high voltage, and low temperature. This involves modifying the SPICE decks to select the correct temperature option, change the supply voltage sources, and load the correct fast transistor models. The values of these parameters are located in the header comment section of the corresponding simulation deck files.

For a truly worst-case analysis, combine the HSPICE Writer hold time analysis results with the Quartus Prime software fast timing model. This requires that you change the double-counting compensation circuitry in the simulations files to also simulate the fast process corners, as this is what the Quartus Prime software uses for the fast timing model.

Note: This method of hold time analysis is recommended only for globally synchronous buses. Do not apply this method of hold-time analysis to source synchronous buses. This is because the source synchronous clocking scheme is designed to cancel out some of the PVT timing effects. If this is not taken into account, the timing results are not accurate. Proper source synchronous timing analysis is beyond the scope of this document.

1.5.13.3. I/O Voltage Variations

Use each of the FPGA family datasheets to verify the recommended operating conditions for supply voltages. For current FPGA families, the maximum recommended voltage corresponds to the fast corner, while the minimum recommended voltage corresponds to the slow corner. These voltage recommendations are specified at the power pins of the FPGA and are not necessarily the same voltage that are seen by the I/O buffers due to package IR drops.

The automatically generated HSPICE simulation files model this IR effect pessimistically by including a 50-mV IR drop on the V_{CCPD} supply when a high drive strength standard is being used.

1.5.13.4. Correlation Report

Correlation reports for the HSPICE I/O models are located in the family-specific HSPICE I/O buffer simulation kits.

Related Information

Accessing HSPICE Simulation Kits on page 21

1.6. Signal Integrity Analysis with Third-Party Tools Document Revision History

Table 4. Document Revision History

Date	Quartus Prime Version	Changes
2024.04.01	24.1	• Updated throughout for initial Altera rebranding. • Updated *What's New In This Version* topic for latest software changes impacting this document. • Updated *Signal Integrity Analysis with Third-Party Tools* topic for removal of Enable Advanced I/O Timing option. • Updated *Create I/O and Board Trace Model Assignments* topic for removal of Enable Advanced I/O Timing option. • Updated *Customize the Output Files* topic for removal of Enable Advanced I/O Timing option. • Updated *IBIS Model Access and Customization Flows* topic for EDA Netlist Writer and Agilex FPGA portfolio device support.

continued...

Date	Quartus Prime Version	Changes
		• Updated *Generate Custom IBIS Models with the EDA Netlist Writer GUI* topic to reflect Agilex FPGA portfolio device support. • Updated *Customizing Downloaded or Installed IBIS Model Files for Agilex FPGA Portfolio Devices* topic for EDA Netlist Writer IBIS model generation support. • Updated *Simulation with HSPICE Models* topic for removal of Enable Advanced I/O Timing option.
2023.08.01	23.1	• Corrected junk characters in all code samples of *Sample Input for I/O HSPICE Simulation Deck* section. • Corrected junk characters in all code samples of *Sample Output for I/O HSPICE Simulation Deck* section.
2023.04.03	23.1	• Added *Top FAQs* navigation to front cover. • Added *What's New In This Version* section. • Revised *Third-Party Board Signal Integrity Analysis Flow* diagram to reflect new IBIS file methods. • Revised *Customize the Output Files* topic to reflect new IBIS file methods. • Revised *Simulation with IBIS Models* topic to reflect new IBIS file methods. • Added *IBIS Model Access and Customization* topic. • Revised *Customizing IBIS Models* topic to reflect new IBIS file methods. • Added new *Customizing Downloaded IBIS Models for Stratix 10 Devices, Arria 10 Devices, and Cyclone 10 GX Devices* topic. • Added new *Generate Custom IBIS Models with the EDA Netlist Writer GUI for Stratix 10 Devices, Arria 10 Devices, and Cyclone 10 GX Devices* topic. • Added new *Customizing IBIS Model Files for Agilex 7 Devices* topic. • Update *Design Simulation Using the Siemens EDA HyperLynx Software* topic for vendor name.
2017.11.06	17.1	• Reorganized chapter introduction.
2016.10.31	16.1	• Implemented Intel rebranding. • Corrected statement about timing simulation and double counting.
2015.11.02	15.1	• Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0	Updated format.
December 2010	10.0	Template update.
July 2010	10.0	Updated device support.
November 2009	9.1	No change to content.
March 2009	9.0	• Was volume 3, chapter 12 in the 8.1.0 release. • No change to content.
November 2008	8.1	• Changed to 8-1/2 x 11 page size. • Added information for Stratix III devices. • Input signals for Cyclone III devices are supported.
May 2008	8.0	• Updated "Introduction" on page 12–1. • Updated Figure 12–1. • Updated Figure 12–3. • Updated Figure 12–13. • Updated "Output File Generation" on page 12–6. • Updated "Simulation with HSPICE Models" on page 12–17. • Updated "Invoking HSPICE Writer from the Command Line" on page 12–22.

continued...

Date	Quartus Prime Version	Changes
		• Added "Sample Input for I/O HSPICE Simulation Deck" on page 12–29. • Added "Sample Output for I/O HSPICE Simulation Deck" on page 12–33. • Updated "Correlation Report" on page 12–41. • Added hyperlinks to referenced documents and websites throughout the chapter. • Made minor editorial updates.

2. Reviewing Printed Circuit Board Schematics with the Quartus Prime Software

Intel FPGAs and CPLDs offer a multitude of configurable options to allow you to implement a custom application-specific circuit on your PCB.

Your Quartus Prime project provides important information specific to your programmable logic design, which you can use in conjunction with the device literature available on the Intel website to ensure that you implement the correct board-level connections in your schematic.

Refer to the **Settings** dialog box options, the Fitter report, and **Messages** window when creating and reviewing your PCB schematic. The Quartus Prime software also provides the Pin Planner to assist you during your PCB schematic review process.

Related Information

Schematic Review Worksheets

2.1. Reviewing Quartus Prime Software Settings

Review these settings in the Quartus Prime software to help you review your PCB schematic.

The **Device** dialog box in the Quartus Prime software allows you to specify device-specific assignments and settings. You can use the **Device** dialog box to specify general project-wide options, including specific device and pin options, which help you to implement correct board-level connections in your PCB schematic.

The **Device** dialog box provides project-specific device information, including the target device and any migration devices you specify. Using migration devices can impact the number of available user I/O pins and internal resources, as well as require connection of some user I/O pins to power/ground pins to support migration.

If you want to use vertical migration, which allows you to use different devices with the same package, you can specify your list of migration devices in the **Migration Devices** dialog box. The Fitter places the pins in your design based on your targeted migration devices, and allows you to use only I/O pins that are common to all the migration devices.

If a migration device has pins that are power or ground, but the pins are also user I/O pins on a different device in the migration path, the Fitter ensures that these pins are not used as user I/O pins. You must ensure that these pins are connected to the appropriate plane on the PCB.

If you are migrating from a smaller device with NC (no-connect) pins to a larger device with power or ground pins in the same package, you can safely connect the NC pins to power or ground pins to facilitate successful migration.

Related Information

Migration Devices Dialog Box
In Quartus Prime Help

2.1.1. Device and Pins Options Dialog Box Settings

You can set device and pin options and verify important design-specific data in the **Device and Pin Options** dialog box, including options found on the **General**, **Configuration**, **Unused Pin**, **Dual-Purpose Pins**, and **Voltage** pages.

2.1.1.1. Configuration Settings

The **Configuration** page of the **Device and Pin Options** dialog box specifies the configuration scheme and configuration device for the target device. Use the **Configuration** page settings to verify the configuration scheme with the MSEL pin settings used on your PCB schematic and the I/O voltage of the configuration scheme.

Your specific configuration settings may impact the availability of some dual-purpose I/O pins in user mode.

Related Information

Dual-Purpose Pins Settings on page 45

2.1.1.2. Unused Pin Settings

The **Unused Pin** page specifies the behavior of all unused pins in your design. Use the **Unused Pin** page to ensure that unused pin settings are compatible with your PCB.

For example, if you reserve all unused pins as outputs driving ground, you must ensure that you do not connect unused I/O pins to VCC pins on your PCB. Connecting unused I/O pins to VCC pins may result in contention that could lead to higher than expected current draw and possible device overstress.

The **Reserve all unused pins** list shows available unused pin state options for the target device. The default state for each pin is the recommended setting for each device family.

When you reserve a pin as output driving ground, the Fitter connects a ground signal to the output pin internally. You should connect the output pin to the ground plane on your PCB, although you are not required to do so. Connecting the output driving ground to the ground plane is known as creating a virtual ground pin, which helps to minimize simultaneous switching noise (SSN) and ground bounce effects.

2.1.1.3. Dual-Purpose Pins Settings

The **Dual-Purpose Pins** page specifies how configuration pins should be used after device configuration completes. You can set the function of the dual-purpose pins by selecting a value for a specific pin in the **Dual-purpose pins** list. Pin functions should match your PCB schematic. The available options on the **Dual-Purpose Pins** page may differ depending on the selected configuration mode.

2.1.1.4. Voltage Settings

The **Voltage** page specifies the default VCCIO I/O bank voltage and the default I/O bank voltage for the pins on the target device. VCCIO I/O bank voltage settings made in the **Voltage** page are overridden by I/O standard assignments made on I/O pins in their respective banks.

Ensure that the settings in the **Voltage** page match the settings in your PCB schematic, especially if the target device includes transceivers.

The **Voltage** page settings requirements differ depending on the settings of the transceiver instances in the design. Refer to the Fitter report for the required settings, and verify that the voltage settings are correctly set up for your PCB schematic.

After verifying your settings in the **Device** and **Settings** dialog boxes, you can verify your device pin-out with the Fitter report.

Related Information

Reviewing Device Pin-Out Information in the Fitter Report on page 46

2.1.1.5. Error Detection CRC Settings

The **Error Detection CRC** page specifies error detection cyclic redundancy check (CRC) use for the target device. When **Enable error detection CRC** is turned on, the device checks the validity of the programming data in the devices. Any changes made in the data while the device is in operation generates an error.

Turning on the **Enable open drain on CRC error pin** option allows the CRC ERROR pin to be set as an open-drain pin in some devices, which decouples the voltage level of the CRC ERROR pin from VCCIO voltage. You must connect a pull-up resistor to the CRC ERROR pin on your PCB if you turn on this option.

In addition to settings in the **Device** dialog box, you should verify settings in the **Voltage** page of the **Settings** dialog box.

Related Information

Device and Pin Options Dialog Box
In Quartus Prime Help

2.2. Reviewing Device Pin-Out Information in the Fitter Report

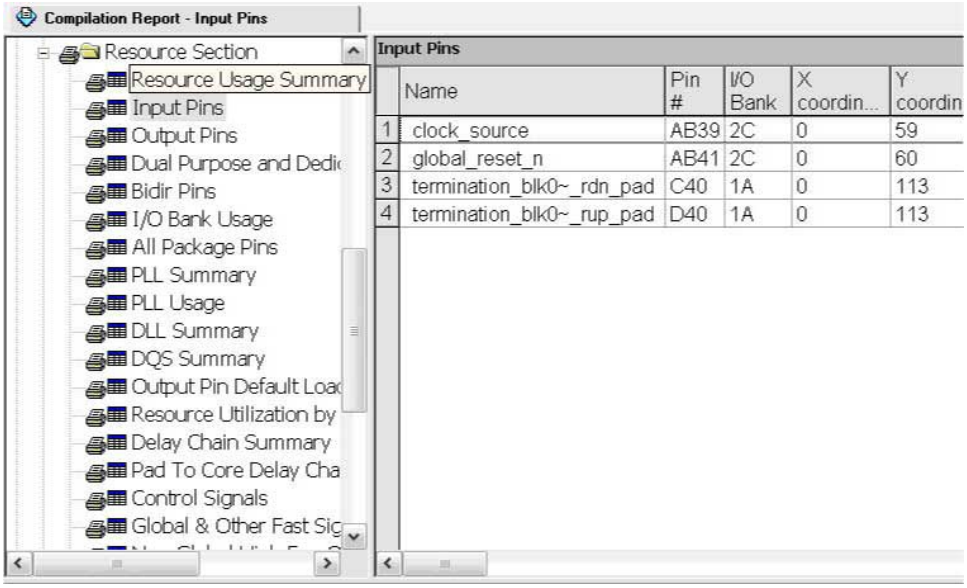
After you compile your design, you can use the reports in the Resource section of the Fitter report to check your device pin-out in detail.

The Input Pins, Output Pins, and Bidirectional Pins reports identify all the user I/O pins in your design and the features enabled for each I/O pin. For example, you can find use of weak internal pull-ups, PCI clamp diodes, and on-chip termination (OCT) pin assignments in these sections of the Fitter report. You can check the pin assignments reported in the Input Pins, Output Pins, and Bidirectional Pins reports against your PCB schematic to determine whether your PCB requires external components.

These reports also identify whether you made pin assignments or if the Fitter automatically placed the pins. If the Fitter changed your pin assignments, you should make these changes user assignments because the location of pin assignments made by the Fitter may change with subsequent compilations.

Figure 18. Resource Section Report

Open the **Compilation Report** tab with `Ctrl+R,`. In the **Plan Stage** folder under **Fitter** open the Input Pins report. The following figure shows the pins the Fitter chose for the OCT external calibration resistor connections (RUP/RDN) and the name of the associated termination block in the Input Pins report. You should make these types of assignments user assignments.



The I/O Bank Usage report provides a high-level overview of the VCCIO and VREF requirements for your design, based on your I/O assignments. Verify that the requirements in this report match the settings in your PCB schematic. All unused I/O banks, and all banks with I/O pins with undefined I/O standards, default the VCCIO voltage to the voltage defined in the **Voltage** page of the **Device and Pin Options** dialog box.

The All Package Pins report lists all the pins on your device, including unused pins, dedicated pins and power/ground pins. You can use this report to verify pin characteristics, such as the location, name, usage, direction, I/O standard and voltage for each pin with the pin information in your PCB schematic. In particular, you should verify the recommended voltage levels at which you connect unused dedicated inputs and I/O and power pins, especially if you selected a migration device. Use the All Package Pins report to verify that you connected all the device voltage rails to the voltages reported.

Errors commonly reported include connecting the incorrect voltage to the predriver supply (VCCPD) pin in a specific bank, or leaving dedicated clock input pins floating. Unused input pins that should be connected to ground are designated as **GND+** in the **Pin Name/Usage** column in the All Package Pins report.

You can also use the All Package Pins report to check transceiver-specific pin connections and verify that they match the PCB schematic. Unused transceiver pins have the following requirements, based on the pin designation in the Fitter report:

- GXB_GND—Unused GXB receiver or dedicated reference clock pin. This pin must be connected to GXB_GND through a 10k Ohm resistor.
- GXB_NC—Unused GXB transmitter or dedicated clock output pin. This pin must be disconnected.

Some transceiver power supply rails have dual voltage capabilities, such as VCCA_L/R and VCCH_L/R, that depend on the settings you created for the ALTGX parameter editor. Because these user-defined settings overwrite the default settings, you should use the All Package Pins report to verify that these power pins on the device symbol in the PCB schematics are connected to the voltage required by the transceiver. An incorrect connection may cause the transceiver to function not as expected.

If your design includes a memory interface, the DQS Summary report provides an overview of each DQ pin group. You can use this report to quickly confirm that the correct DQ/DQS pins are grouped together.

Finally, the Fitter Device Options report summarizes some of the settings made in the **Device and Pin Options** dialog box. Verify that these settings match your PCB schematics.

2.3. Reviewing Compilation Error and Warning Messages

If your project does not compile without error or warning messages, you should resolve the issues identified by the Compiler before signing off on your pin-out or PCB schematic. Error messages often indicate illegal or unsupported use of the device resources and IP.

Additionally, you should cross-reference fitting and timing analysis warnings with the design implementation. Timing may be constrained due to nonideal pin placement. You should investigate if you can reassign pins to different locations to prevent fitting and timing analysis warnings. Ensure that you review each warning and consider its potential impact on the design.

2.4. Using Additional Quartus Prime Software Features

You can generate IBIS files, which contain models specific to your design and selected I/O standards and options, with the Quartus Prime software.

Because board-level simulation is important to verify, you should check for potential signal integrity issues. You can turn on the **Board-Level Signal Integrity** feature in the **EDA Tool Settings** page of the **Settings** dialog box.

Additionally, using advanced I/O timing allows you to enter physical PCB information to accurately model the load seen by an output pin. This feature facilitates accurate I/O timing analysis.

Related Information

- Signal Integrity Analysis with Third-Party Tools on page 4
- Managing Device I/O Pins

2.5. Using Additional Quartus Prime Software Tools

Use the Pin Planner to assist you with reviewing your PCB schematics.

2.5.1. Pin Planner

The Quartus Prime Pin Planner helps you visualize, plan, and assign device I/O pins in a graphical view of the target device package. You can quickly locate various I/O pins and assign them design elements or other properties to ensure compatibility with your PCB layout.

You can use the Pin Planner to verify the location of clock inputs, and whether they have been placed on dedicated clock input pins, which is recommended when your design uses PLLs.

You can also use the Pin Planner to verify the placement of dedicated SERDES pins. SERDES receiver inputs can be placed only on DIFFIO_RX pins, while SERDES transmitter outputs can be placed only on DIFFIO_TX pins.

The Pin Planner gives a visual indication of signal-to-signal proximity in the **Pad View** window, and also provides information about differential pin pair placement, such as the placement of pseudo-differential signals.

Related Information

Managing Device I/O Pins

2.6. Reviewing Printed Circuit Board Schematics with the Quartus Prime Software Revision History

Table 5. Document Revision History

Date	Quartus Prime Version	Changes
2018.09.24	18.1	• First release as part of the stand-alone *Quartus Prime Standard Edition User Guide*
2016.10.31	16.1	• Implemented Intel rebranding.
2015.11.02	15.1	Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0	Template update.
November 2012	12.1	Minor update of Pin Planner description for task and report windows.
June 2012	12.0	Removed survey link.
November 2011	10.0	Template update.
December 2010	10.0	Changed to new document template. No change to content.
July 2010	10.0	Initial release.

3. Siemens EDA PCB Design Tools Support

You can integrate the Siemens EDA DxDesigner* PCB design tool into the Quartus Prime design flow.

With today's large, high-pin-count and high-speed FPGA devices, good and correct PCB design practices are essential to ensure correct system operation. The PCB design takes place concurrently with the design and programming of the FPGA. The FPGA or ASIC designer initially creates signal and pin assignments, and the board designer must correctly transfer these assignments to the symbols in their system circuit schematics and board layout. As the board design progresses, Intel recommends reassigning pins to optimize the PCB layout. Ensure that you inform the FPGA designer of the pin reassignments so that the new assignments are included in an updated placement and routing of the design.

This chapter covers the following topics:

- Siemens EDA and Quartus Prime software integration flow
- Generating supporting files
- Creating DxDesigner symbols from the Quartus Prime output files

This chapter is intended for board design and layout engineers who want to start the FPGA board integration while the FPGA is still in the design phase. Alternatively, the board designer can plan the FPGA pin-out and routing requirements in the Siemens EDA tools and pass the information back to the Quartus Prime software for placement and routing. Part librarians can also benefit from this chapter by learning how to use output from the Quartus Prime software to create new library parts and symbols.

The procedures in this chapter require the following software:

- The Quartus Prime software version 5.1 or later
- DxDesigner software version 2004 or later

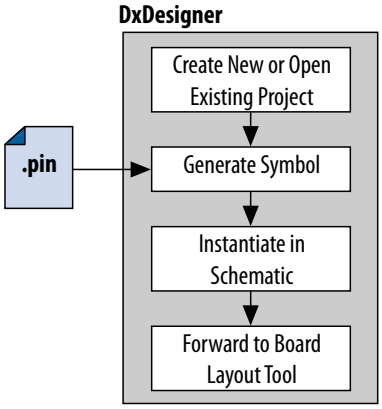
Note: To obtain and license the Siemens EDA tools and for product information, support, and training, refer to the Siemens EDA website.

3.1. Integrating with DxDesigner

You can integrate the Siemens EDA DxDesigner schematic capture tool into the Quartus Prime design flow. Use DxDesigner to create flat circuit schematics or to create hierarchical schematics that facilitate design reuse and a team-based design for all PCB types.

If you use DxDesigner without the I/O Designer software, the design flow is one-way, using only the **.pin** generated by the Quartus Prime software. You can only make signal and pin assignment changes in the Quartus Prime software. You cannot back-annotate changes made in a board layout tool or in a DxDesigner symbol to the Quartus Prime software.

Figure 19. DxDesigner-only Flow



3.1.1. DxDesigner Project Settings

DxDesigner new projects automatically create FPGA symbols by default. To enable the DxBoardLink flow design configuration when creating a new DxDesigner project, follow these steps:

1. Start the DxDesigner software.
2. Click **File ➤ New**, and then click the **Project** tab.
3. Click **More**. Turn on **DxBoardLink**. To enable the DxBoardLink Flow design configuration for an existing project, click **Design Configurations** in the Design Configuration toolbar and turn on **DxBoardLink**.

3.1.2. Creating Schematic Symbols in DxDesigner

You can create schematic symbols in the DxDesigner software manually or with the Symbol wizard. The DxDesigner Symbol wizard is similar to the I/O Designer Symbol wizard, but with fewer fracturing options. The DxDesigner Symbol wizard creates, fractures, and edits FPGA symbols based on the specified Intel FPGA device. To create a symbol with the Symbol wizard, follow these steps;

1. Start the DxDesigner software.
2. Click **Symbol Wizard** in the toolbar.
3. Type the new symbol name in the name field and click **OK**.
4. Specify creation of a new symbol or modification of an existing symbol. To modify an existing symbol, specify the library path or alias, and select the existing symbol. To create a new symbol, select DxBoardLink for the symbol source. The DxDesigner block type defaults to Module because the FPGA design does not have an underlying DxDesigner schematic. Choose whether or not to fracture the symbol. Click **Next**.
5. Type a name for the symbol, an overall part name for all the symbol fractures, and a library name for the new library created for this symbol. By default, the part and library names are the same as the symbol name. Click **Next**.
6. Specify the appearance of the generated symbol in your DxDesigner project schematic. After making your selections. Click **Next**.

7. In the **FPGA vendor** list, select **Intel Quartus**. In the **Pin-Out file to import** field, select the **.pin** from your Quartus Prime project directory. You can also specify Fracturing Scheme, Bus pin, and Power pin options. Click **Next**.
8. Select to create or modify symbol attributes for use in the DxDesigner software. Click **Next**.
9. On the **Pin Settings** page, make any final adjustments to pin and label location and information. Each tabbed spreadsheet represents a fracture of your symbol. Click **Save Symbol**.
   After creating the symbol, you can examine and place any fracture of the symbol in your schematic. You can locate separate files of all the fractures you created in the library you specified or created in the **/sym** directory in your DxDesigner project. You can add the symbols to your schematics or you can manually edit the symbols or with the Symbol wizard.

3.2. Siemens EDA PCB Design Tools Support Revision History

Table 6. Document Revision History

Date	Quartus Prime Version	Changes
2020.11.04	18.1	Removed references to unsupported `.fx` files and related tools.
2016.10.31	16.1	• Implemented Intel rebranding.
2015.11.02	15.1	• Changed instances of *Quartus II* to *Quartus Prime*.
2014.06.30	14.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog. • Added standard information about upgrading IP cores. • Added standard installation and licensing information. • Removed outdated device support level information. IP core device support is now available in IP Catalog and parameter editor.
June 2012	12.0	• Removed survey link.
December 2010	10.1	• Changed to new document template.

4. Cadence Board Design Tools Support

4.1. Cadence PCB Design Tools Support

The Quartus Prime software interacts with the following software to provide a complete FPGA-to-board integration design workflow: the Cadence Allegro Design Entry HDL software and the Cadence Allegro Design Entry CIS (Component Information System) software (also known as OrCAD Capture CIS). The information is useful for board design and layout engineers who want to begin the FPGA board integration process while the FPGA is still in the design phase. Part librarians can also benefit by learning the method to use output from the Quartus Prime software to create new library parts and symbols.

With today's large, high-pin-count and high-speed FPGA devices, good PCB design practices are important to ensure the correct operation of your system. The PCB design takes place concurrently with the design and programming of the FPGA. An FPGA or ASIC designer initially creates the signal and pin assignments and the board designer must transfer these assignments to the symbols used in their system circuit schematics and board layout correctly. As the board design progresses, you must perform pin reassignments to optimize the layout. You must communicate pin reassignments to the FPGA designer to ensure the new assignments are processed through the FPGA with updated placement and routing.

You require the following software:

- The Quartus Prime software version 5.1 or later
- The Cadence Allegro Design Entry HDL software or the Cadence Allegro Design Entry CIS software version 15.2 or later
- The OrCAD Capture software with the optional CIS option version 10.3 or later (optional)

Note: These programs are very similar because the Cadence Allegro Design Entry CIS software is based on the OrCAD Capture software. Any procedural information can also apply to the OrCAD Capture software unless otherwise noted.

Related Information

- www.cadence.com
  For more information about obtaining and licensing the Cadence tools and for product information, support, and training
- www.orcad.com
  For more information about the OrCAD Capture software and the CIS option
- www.ema-eda.com
  For more information about Cadence and OrCAD support and training.

4.2. Product Comparison

Table 7. Cadence and OrCAD Product Comparison

Description	Cadence Allegro Design Entry HDL	Cadence Allegro Design Entry CIS	OrCAD Capture CIS
Former Name	Concept HDL Expert	Capture CIS Studio	—
History	More commonly known by its former name, Cadence renamed all board design tools in 2004 under the Allegro name.	Based directly on OrCAD Capture CIS, the Cadence Allegro Design Entry CIS software is still developed by OrCAD but sold and marketed by Cadence. EMA provides support and training.	The basis for Design Entry CIS is still developed by OrCAD for continued use by existing OrCAD customers. EMA provides support and training for all OrCAD products.
Vendor Design Flow	Cadence Allegro 600 series, formerly known as the Expert Series, for high-end, high-speed design.	Cadence Allegro 200 series, formerly known as the Studio Series, for small- to medium-level design.	—

Related Information

- www.cadence.com
- www.ema-eda.com

4.3. FPGA-to-PCB Design Flow

You can create a design flow integrating an Intel FPGA design from the Quartus Prime software through a circuit schematic in the Cadence Allegro Design Entry HDL software or the Cadence Allegro Design Entry CIS software.

Figure 20. Design Flow with the Cadence Allegro Design Entry HDL Software

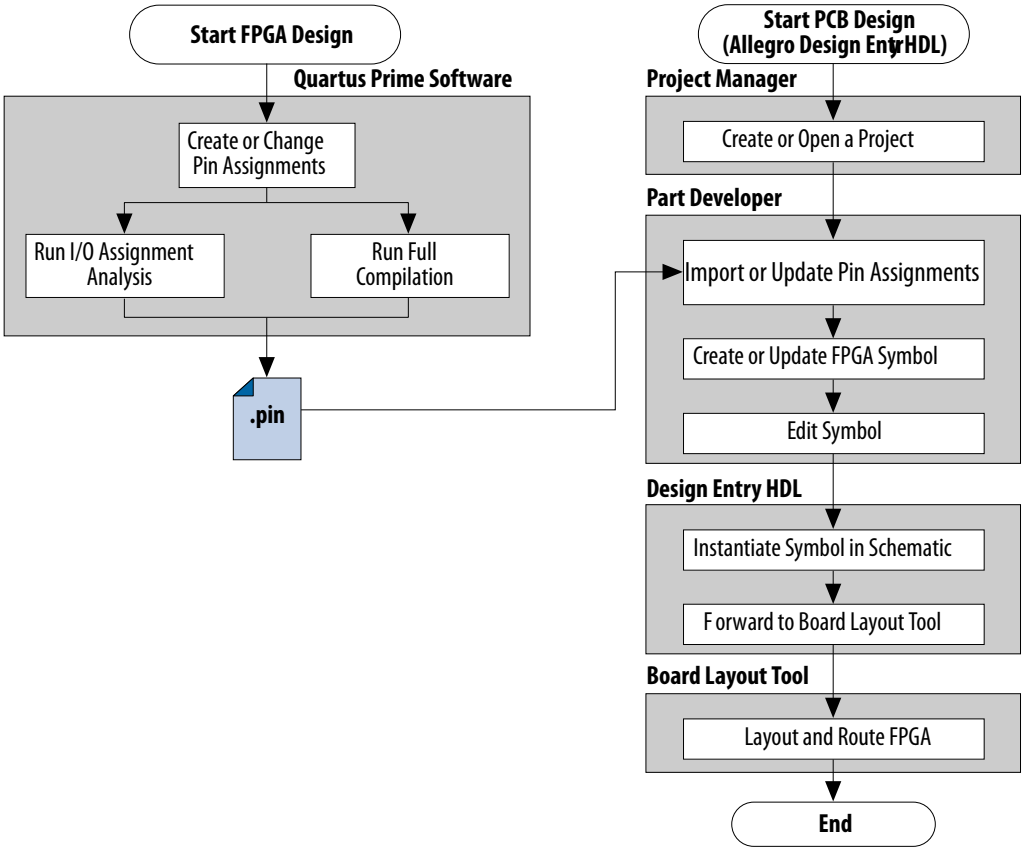
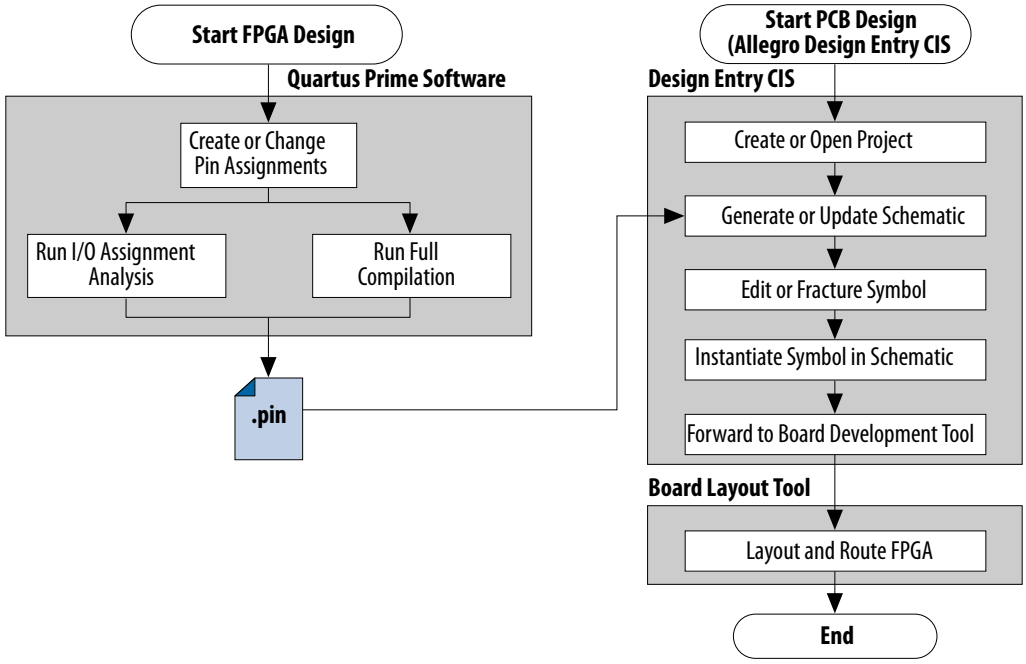


Figure 21. Design Flow with the Cadence Allegro Design Entry CIS Software



To create FPGA symbols using the Cadence Allegro PCB Librarian Part Developer tool, you must obtain the Cadence PCB Librarian Expert license. You can update symbols with changes made to the FPGA design using any of these tools.

4.3.1. Integrating Intel FPGA Designs

To integrate an Intel FPGA design starting in the Quartus Prime software through to a circuit schematic in the Cadence Allegro Design Entry HDL software or the Cadence Allegro Design Entry CIS software, follow these steps:

1. In the Quartus Prime software, compile your design to generate a Pin-Out File (**.pin**) to transfer the assignments to the Cadence software.
2. If you are using the Cadence Allegro Design Entry HDL software for your schematic design, follow these steps:
   a. Open an existing project or create a new project in the Cadence Allegro Project Manager tool.
   b. Construct a new symbol or update an existing symbol using the Cadence Allegro PCB Librarian Part Developer tool.
   c. With the Cadence Allegro PCB Librarian Part Developer tool, edit your symbol or fracture it into smaller parts (optional).
   d. Instantiate the symbol in your Cadence Allegro Design Entry HDL software schematic and transfer the design to your board layout tool.
3. If you are using the Cadence Allegro Design Entry CIS software for your schematic design, follow these steps:

a. Generate a new part in a new or existing Cadence Allegro Design Entry CIS project, referencing the **.pin** output file from the Quartus Prime software. You can also update an existing symbol with a new **.pin**.

b. Split the symbol into smaller parts as necessary.

c. Instantiate the symbol in your Cadence Allegro Design Entry CIS schematic and transfer the design to your board layout tool.

4.4. Setting Up the Quartus Prime Software

You can transfer pin and signal assignments from the Quartus Prime software to the Cadence design tools by generating the Quartus Prime project **.pin**. The **.pin** is an output file generated by the Quartus Prime Fitter containing pin assignment information. You can use the Quartus Prime Pin Planner to set and change the assignments in the **.pin** and then transfer the assignments to the Cadence design tools. You cannot, however, import pin assignment changes from the Cadence design tools into the Quartus Prime software with the **.pin**.

The **.pin** lists all used and unused pins on your selected Intel device. The **.pin** also provides the following basic information fields for each assigned pin on the device:

- Pin signal name and usage
- Pin number
- Signal direction
- I/O standard
- Voltage
- I/O bank
- User or Fitter-assigned

4.4.1. Generating a .pin File

To generate a **.pin**, follow these steps:

1. Compile your design.
2. Locate the **.pin** in your Quartus Prime project directory with the name <project name>**.pin**.

4.5. FPGA-to-Board Integration with the Cadence Allegro Design Entry HDL Software

The Cadence Allegro Design Entry HDL software is a schematic capture tool and is part of the Cadence 600 series design flow. Use the Cadence Allegro Design Entry HDL software to create flat circuit schematics for all types of PCB design. The Cadence Allegro Design Entry HDL software can also create hierarchical schematics to facilitate design reuse and team-based design. With the Cadence Allegro Design Entry HDL software, the design flow from FPGA-to-board is one-way, using only the **.pin** generated by the Quartus Prime software. You can only make signal and pin assignment changes in the Quartus Prime software and these changes reflect as updated symbols in a Cadence Allegro Design Entry HDL project.

For more information about the design flow with the Cadence Allegro Design Entry HDL software, refer to FPGA-to-PCB Design Flow.

Note: Routing or pin assignment changes made in a board layout tool or a Cadence Allegro Design Entry HDL software symbol cannot be back-annotated to the Quartus Prime software.

Related Information

www.cadence.com
Provides information about the Cadence Allegro Design Entry HDL software and the Cadence Allegro PCB Librarian Part Developer tool, including licensing, support, usage, training, and product updates.

4.5.1. Creating Symbols

In addition to circuit simulation, circuit board schematic creation is one of the first tasks required when designing a new PCB. Schematics must understand how the PCB works, and to generate a netlist for a board layout tool for board design and routing. The Cadence Allegro PCB Librarian Part Developer tool allows you to create schematic symbols based on FPGA designs exported from the Quartus Prime software.

You can create symbols for the Cadence Allegro Design Entry HDL project with the Cadence Allegro PCB Librarian Part Developer tool, which is available in the Cadence Allegro Project Manager tool. Intel recommends using the Cadence Allegro PCB Librarian Part Developer tool to import FPGA designs into the Cadence Allegro Design Entry HDL software.

You must obtain a PCB Librarian Expert license from Cadence to run the Cadence Allegro PCB Librarian Part Developer tool. The Cadence Allegro PCB Librarian Part Developer tool provides a GUI with many options for creating, editing, fracturing, and updating symbols. If you do not use the Cadence Allegro PCB Librarian Part Developer tool, you must create and edit symbols manually in the Symbol Schematic View in the Cadence Allegro Design Entry HDL software.

Note: If you do not have a PCB Librarian Expert license, you can automatically create FPGA symbols using the programmable IC (PIC) design flow found in the Cadence Allegro Project Manager tool.

Before creating a symbol from an FPGA design, you must open a Cadence Allegro Design Entry HDL project with the Cadence Allegro Project Manager tool. If you do not have an existing Cadence Allegro Design Entry HDL project, you can create one with the Cadence Allegro Design Entry HDL software. The Cadence Allegro Design Entry HDL project directory with the name <project name>`.cpm` contains your Cadence Allegro Design Entry HDL projects.

While the Cadence Allegro PCB Librarian Part Developer tool refers to symbol fractures as slots, the other tools use different names to refer to symbol fractures.

Table 8. Symbol Fracture Naming Conventions

	Cadence Allegro PCB Librarian Part Developer Tool	Cadence Allegro Design Entry HDL Software	Cadence Allegro Design Entry CIS Software
During symbol generation	Slots	—	Sections
During symbol schematic instantiation	—	Versions	Parts

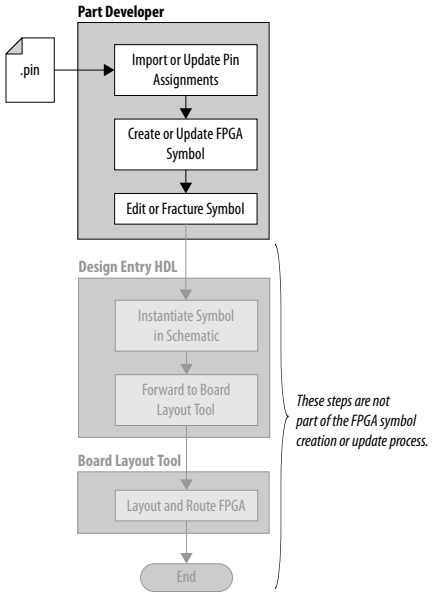
Related Information

www.cadence.com
Provides information about using the PIC design flow.

4.5.1.1. Cadence Allegro PCB Librarian Part Developer Tool

You can create, fracture, and edit schematic symbols for your designs using the Cadence Allegro PCB Librarian Part Developer tool. Symbols designed in the Cadence Allegro PCB Librarian Part Developer tool can be split or fractured into several functional blocks called slots, allowing multiple smaller part fractures to exist on the same schematic page or across multiple pages.

4.5.1.1.1. Cadence Allegro PCB Librarian Part Developer Tool in the Design Flow



To run the Cadence Allegro PCB Librarian Part Developer tool, you must open a Cadence Allegro Design Entry HDL project in the Cadence Allegro Project Manager tool. To open the Cadence Allegro PCB Librarian Part Developer tool, on the Flows menu, click **Library Management**, and then click **Part Developer**.

Related Information

FPGA-to-PCB Design Flow on page 54

4.5.1.1.2. Import and Export Wizard

After starting the Cadence Allegro PCB Librarian Part Developer tool, use the **Import and Export** wizard to import your pin assignments from the Quartus Prime software.

Note: Intel recommends using your PCB Librarian Expert license file. To point to your PCB Librarian Expert license file, on the File menu, click **Change Product** and then select the correct product license.

To access the Import and Export wizard, follow these steps:

1. On the File menu, click **Import and Export**.
2. Select **Import ECO-FPGA**, and then click **Next**.
3. In the **Select Source** page of the **Import and Export** wizard, specify the following settings:
   a. In the **Vendor** list, select **Altera**.
   b. In the **PnR Tool** list, select **quartusII**.
   c. In the **PR File** box, browse to select the **.pin** in your Quartus Prime project directory.
   d. Click **Simulation Options** to select simulation input files.
   e. Click **Next.**
4. In the **Select Destination** dialog box, specify the following settings:
   a. Under **Select Component**, click **Generate Custom Component** to create a new component in a library,

      or

      Click **Use standard component** to base your symbol on an existing component.

      Note: Intel recommends creating a new component if you previously created a generic component for an FPGA device. Generic components can cause some problems with your design. When you create a new component, you can place your pin and signal assignments from the Quartus Prime software on this component and reuse the component as a base when you have a new FPGA design.

   b. In the **Library** list, select an existing library. You can select from the cells in the selected library. Each cell represents all the symbol versions and part fractures for a particular part. In the **Cell** list, select the existing cell to use as a base for your part.
   c. In the **Destination Library** list, select a destination library for the component. Click **Next**.
   d. Review and edit the assignments you import into the Cadence Allegro PCB Librarian Part Developer tool based on the data in the **.pin** and then click **Finish**. The location of each pin is not included in the **Preview of Import Data** page of the **Import and Export** wizard, but input pins are on the left side of the created symbol, output pins on the right, power pins on the top, and ground pins on the bottom.

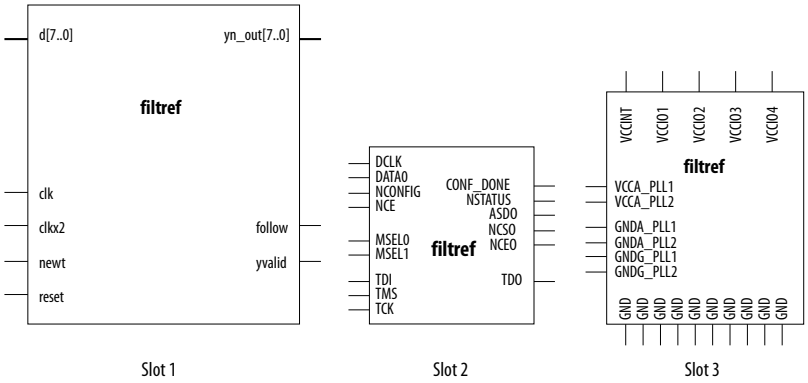
4.5.1.1.3. Editing and Fracturing Symbol

After creating your new symbol in the Cadence Allegro PCB Librarian Part Developer tool, you can edit the symbol graphics, fracture the symbol into multiple slots, and add or change package or symbol properties.

The Part Developer Symbol Editor contains many graphical tools to edit the graphics of a particular symbol. To edit the symbol graphics, select the symbol in the cell hierarchy. The **Symbol Pins** tab appears. You can edit the preview graphic of the symbol in the **Symbol Pins** tab.

Fracturing a Cadence Allegro PCB Librarian Part Developer package into separate symbol slots is useful for FPGA designs. A single symbol for most FPGA packages might be too large for a single schematic page. Splitting the part into separate slots allows you to organize parts of the symbol by function, creating cleaner circuit schematics. For example, you can create one slot for an I/O symbol, a second slot for a JTAG symbol, and a third slot for a power/ground symbol.

Figure 22. Splitting a Symbol into Multiple Slots



- This diagram represents a Cyclone device with JTAG or passive serial (PS) mode configuration option settings. Symbols created for other devices or other configuration modes may have dif erent sets of configuration pins, but can be fractured in a similar manner.
- The power/ground slot shows only a representation of power and ground pins because the device contains a large number of power and ground pins.

To fracture a part into separate slots, or to modify the slot locations of pins on parts fractured in the Cadence Allegro PCB Librarian Part Developer tool, follow these steps:

1. Start the Cadence Allegro Design Project Manager.
2. On the Flows menu, click **Library Management**.
3. Click **Part Developer**.
4. Click the name of the package you want to change in the cell hierarchy.
5. Click **Functions/Slots**. If you are not creating new slots but want to change the slot location of some pins, proceed to Step 6. If you are creating new slots, click **Add**. A dialog box appears, allowing you to add extra symbol slots. Set the number of extra slots you want to add to the existing symbol, not the total number of desired slots for the part. Click **OK**.
6. Click **Distribute Pins**. Specify the slot location for each pin. Use the checkboxes in each column to move pins from one slot to another. Click **OK**.
7. After distributing the pins, click the **Package Pin** tab and click **Generate Symbol(s)**.
8. Select whether to create a new symbol or modify an existing symbol in each slot. Click **OK**.

   The newly generated or modified slot symbols appear as separate symbols in the cell hierarchy. Each of these symbols can be edited individually.

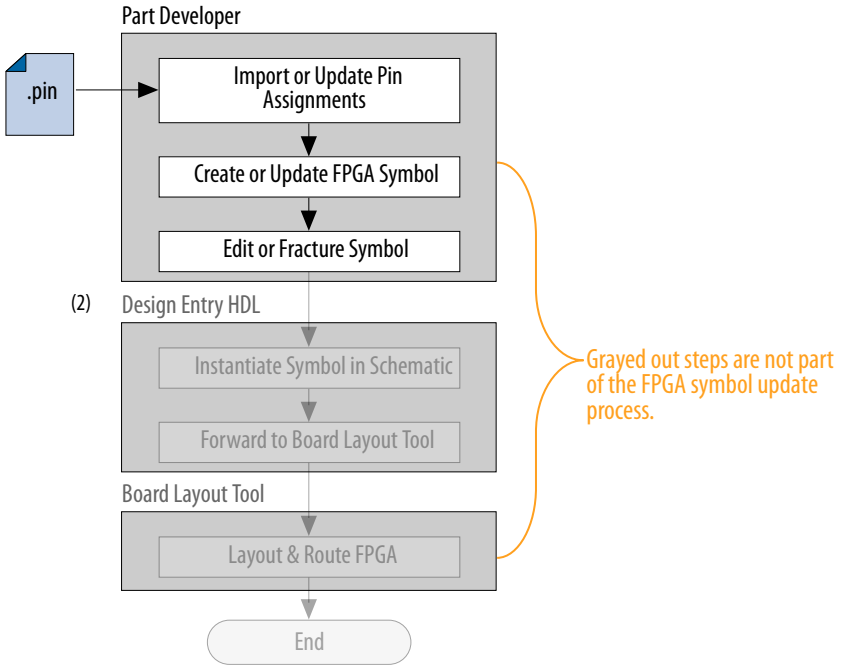
Caution: The Cadence Allegro PCB Librarian Part Developer tool allows you to remap pin assignments in the **Package Pin** tab of the main Cadence Allegro PCB Librarian Part Developer window. If signals remap to different pins in the Cadence Allegro PCB Librarian Part Developer tool, the changes reflect only in regenerated symbols for use in your schematics. You cannot transfer pin assignment changes to the Quartus Prime software from the Cadence Allegro PCB Librarian Part Developer tool, which creates a potential mismatch of the schematic symbols and assignments in the FPGA design. If pin assignment changes are necessary, make the changes in the Quartus Prime Pin Planner instead of the Cadence Allegro PCB Librarian Part Developer tool, and update the symbol as described in the following sections.

For more information about creating, editing, and organizing component symbols with the Cadence Allegro PCB Librarian Part Developer tool, refer to the Part Developer Help.

4.5.1.1.4. Updating FPGA Symbols

As the design process continues, you must make logic changes in the Quartus Prime software, placing signals on different pins after recompiling the design, or use the Quartus Prime Pin Planner to make changes manually. The board designer can request such changes to improve the board routing and layout. To ensure signals connect to the correct pins on the FPGA, you must carry forward these types of changes to the circuit schematic and board layout tools. Updating the `.pin` in the Quartus Prime software facilitates this flow.

Figure 23. Updating the FPGA Symbol in the Design Flow



To update the symbol using the Cadence Allegro PCB Librarian Part Developer tool after updating the `.pin`, follow these steps:

1. On the File menu, click **Import and Export**. The Import and Export wizard appears.
2. In the list of actions to perform, select **Import ECO - FPGA**. Click **Next**. The **Select Source** dialog box appears.
3. Select the updated source of the FPGA assignment information. In the **Vendor** list, select **Altera**. In the **PnR Tool** list, select **quartusII**. In the **PR File** field, click **browse** to specify the updated **.pin** in your Quartus Prime project directory. Click **Next**. The Select Destination window appears.
4. Select the source component and a destination cell for the updated symbol. To create a new component based on the updated pin assignment data, select **Generate Custom Component**. Selecting **Generate Custom Component** replaces the cell listed under the **Specify Library and Cell** name header with a new, nonfractured cell. You can preserve these edits by selecting **Use standard component and select the existing library and cell**. Select the destination library for the component and click **Next**. The **Preview of Import Data** dialog box appears.
5. Make any additional changes to your symbol. Click **Next**. A list of ECO messages appears summarizing the changes made to the cell. To accept the changes and update the cell, click **Finish**.
6. The main Cadence Allegro PCB Librarian Part Developer window appears. You can edit, fracture, and generate the updated symbols as usual from the main Cadence Allegro PCB Librarian Part Developer window.

Note: If the Cadence Allegro PCB Librarian Part Developer tool is not set up to point to your PCB Librarian Expert license file, an error message appears in red at the bottom of the message text window of the Part Developer when you select the **Import and Export** command. To point to your PCB Librarian Expert license, on the File menu, click **Change Product**, and select the correct product license.

Related Information

FPGA-to-PCB Design Flow on page 54

4.5.2. Instantiating the Symbol in the Cadence Allegro Design Entry HDL Software

To instantiate the symbol in your Cadence Allegro Design Entry HDL schematic after saving the new symbol in the Cadence Allegro PCB Librarian Part Developer tool, follow these steps:

1. In the Cadence Allegro Project Manager tool, switch to the board design flow.
2. On the Flows menu, click **Board Design**.
3. To start the Cadence Allegro Design Entry HDL software, click **Design Entry**.
4. To add the newly created symbol to your schematic, on the Component menu, click **Add**. The **Add Component** dialog box appears.
5. Select the new symbol library location, and select the name of the cell you created from the list of cells.

The symbol attaches to your cursor for placement in the schematic. To fracture the symbol into slots, right-click the symbol and choose **Version** to select one of the slots for placement in the schematic.

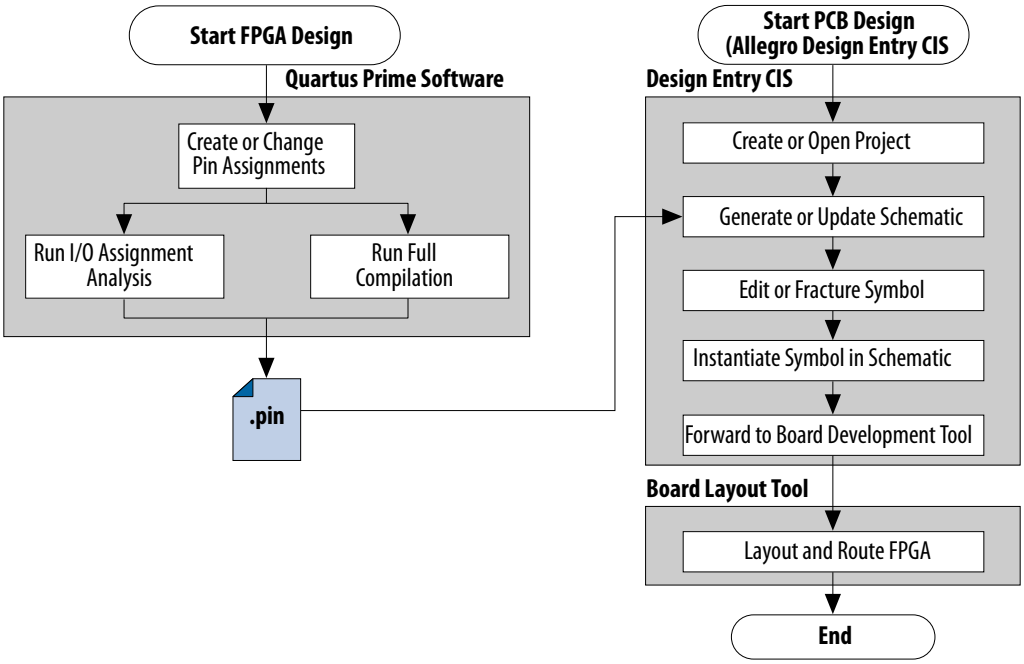
Related Information

www.cadence.com
Provides more information about the Cadence Allegro Design Entry HDL software, including licensing, support, usage, training, and product updates.

4.6. FPGA-to-Board Integration with Cadence Allegro Design Entry CIS Software

The Cadence Allegro Design Entry CIS software is a schematic capture tool (part of the Cadence 200 series design flow based on OrCAD Capture CIS). Use the Cadence Allegro Design Entry CIS software to create flat circuit schematics for all types of PCB design. You can also create hierarchical schematics to facilitate design reuse and team-based design using the Cadence Allegro Design Entry CIS software. With the Cadence Allegro Design Entry CIS software, the design flow from FPGA-to-board is unidirectional using only the **`.pin`** generated by the Quartus Prime software. You can only make signal and pin assignment changes in the Quartus Prime software. These changes reflect as updated symbols in a Cadence Allegro Design Entry CIS schematic project.

Figure 24. Design Flow with the Cadence Allegro Design Entry CIS Software



Note: Routing or pin assignment changes made in a board layout tool or a Cadence Allegro Design Entry CIS symbol cannot be back-annotated to the Quartus Prime software.

Related Information

- www.cadence.com
  For more information about the Cadence Allegro Design Entry CIS software, including licensing, support, usage, training, and product updates.
- www.ema-eda.com
  For more information about the Cadence Allegro Design Entry CIS software, including licensing, support, usage, training, and product updates.

4.6.1. Creating a Cadence Allegro Design Entry CIS Project

The Cadence Allegro Design Entry CIS software has built-in support for creating schematic symbols using pin assignment information imported from the Quartus Prime software.

To create a new project in the Cadence Allegro Design Entry CIS software, follow these steps:

1. On the File menu, point to **New** and click **Project.** The New Project wizard starts.

   When you create a new project, you can select the PC Board wizard, the Programmable Logic wizard, or a blank schematic.
2. Select the PC Board wizard to create a project where you can select which part libraries to use, or select a blank schematic.

The Programmable Logic wizard only builds an FPGA logic design in the Cadence Allegro Design Entry CIS software.

Your new project is in the specified location and consists of the following files:

- OrCAD Capture Project File (**`.opj`**)
- Schematic Design File (**`.dsn`**)

4.6.2. Generating a Part

After you create a new project or open an existing project in the Cadence Allegro Design Entry CIS software, you can generate a new schematic symbol based on your Quartus Prime FPGA design. You can also update an existing symbol. The Cadence Allegro Design Entry CIS software stores component symbols in OrCAD Library File (**`.olb`**). When you place a symbol in a library attached to a project, it is immediately available for instantiation in the project schematic.

You can add symbols to an existing library or you can create a new library specifically for the symbols generated from your FPGA designs. To create a new library, follow these steps:

1. On the File menu, point to **New** and click **Library** in the Cadence Allegro Design Entry CIS software to create a default library named **library1.olb**. This library appears in the **Library** folder in the Project Manager window of the Cadence Allegro Design Entry CIS software.
2. To specify a desired name and location for the library, right-click the new library and select **Save As**. Saving the new library creates the library file.

4.6.3. Generating Schematic Symbol

You can now create a new symbol to represent your FPGA design in your schematic.

To generate a schematic symbol, follow these steps:

1. Start the Cadence Allegro Design Entry CIS software.
2. On the Tools menu, click **Generate Part**. The **Generate Part** dialog box appears.
3. To specify the **.pin** from your Quartus Prime design, in the **Netlist/source file type** field, click **Browse**.
4. In the **Netlist/source file type** list, select **Altera Pin File**
5. Type the new part name.
6. Specify the **Destination part library** for the symbol. Failing to select an existing library for the part creates a new library with a default name that matches the name of your Cadence Allegro Design Entry CIS project.
7. To create a new symbol for this design, select **Create new part**. If you updated your **.pin** in the Quartus Prime software and want to transfer any assignment changes to an existing symbol, select **Update pins on existing part in library**.
8. Select any other desired options and set **Implementation type** to **<none>**. The symbol is for a primitive library part based only on the **.pin** and does not require special implementation. Click **OK**.
9. Review the Undo warning and click **Yes** to complete the symbol generation.

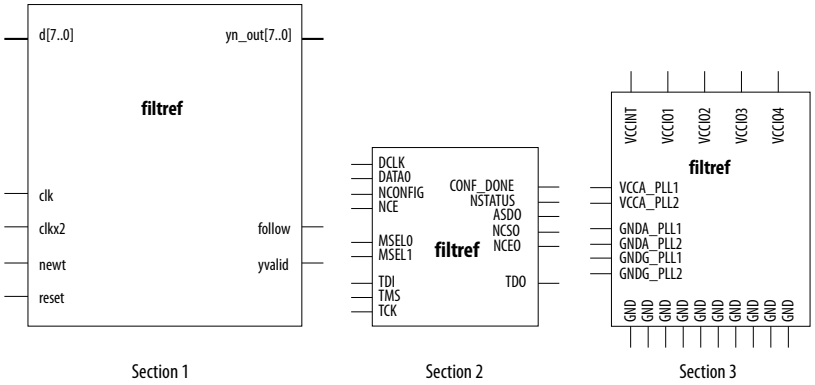
   You can locate the generated symbol in the selected library or in a new library found in the **Outputs** folder of the design in the Project Manager window. Double-click the name of the new symbol to see its graphical representation and edit it manually using the tools available in the Cadence Allegro Design Entry CIS software.

   Note: For more information about creating and editing symbols in the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

4.6.4. Splitting a Part

After saving a new symbol in a project library, you can fracture the symbol into multiple parts called sections. Fracturing a part into separate sections is useful for FPGA designs. A single symbol for most FPGA packages might be too large for a single schematic page. Splitting the part into separate sections allows you to organize parts of the symbol by function, creating cleaner circuit schematics. For example, you can create one slot for an I/O symbol, a second slot for a JTAG symbol, and a third slot for a power/ground symbol.

Figure 25. Splitting a Symbol into Multiple Sections

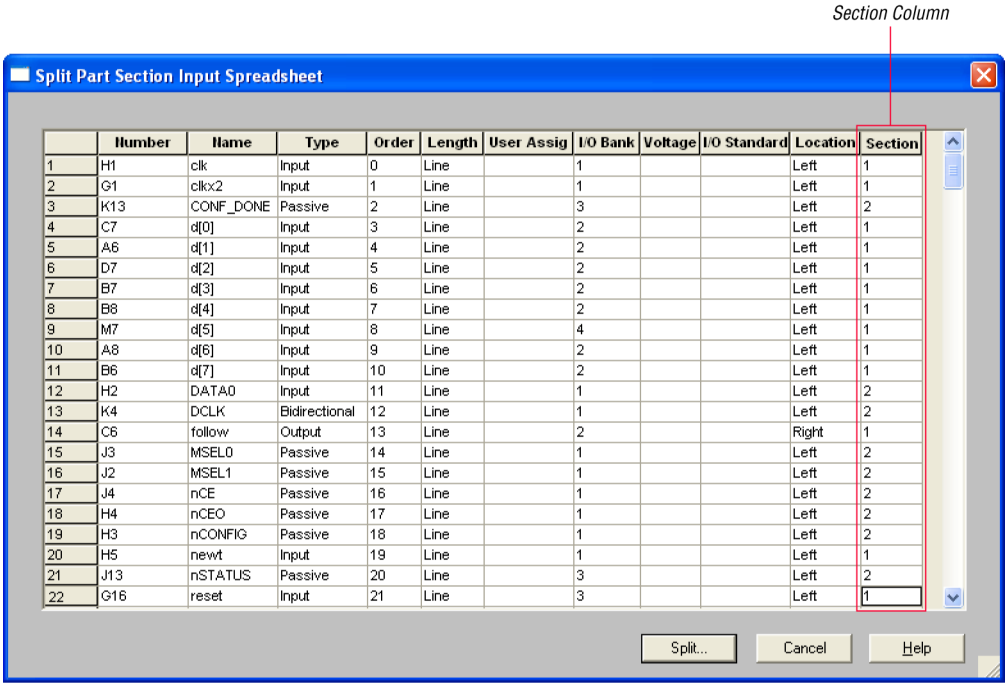


- This diagram represents a Cyclone device with JTAG or passive serial (PS) mode configuration option settings. Symbols created for other devices or other configuration modes might have dif erent sets of configuration pins, but can be fractured in a similar manner.
- The power/ground section shows only a representation of power and ground pins because the device contains a high number of power and ground pins.

Note: Although symbol generation in the Design Entry CIS software refers to symbol fractures as sections, other tools use different names to refer to symbol fractures.

To split a part into sections, select the part in its library in the Project Manager window of the Cadence Allegro Design Entry CIS software. On the Tools menu, click **Split Part** or right-click the part and choose **Split Part**. The **Split Part Section Input Spreadsheet** appears.

Figure 26. Split Part Section Input Spreadsheet



	Number	Name	Type	Order	Length	User Assig	I/O Bank	Voltage	I/O Standard	Location	Section
1	H1	clk	Input	0	Line		1			Left	1
2	G1	clkx2	Input	1	Line		1			Left	1
3	K13	CONF_DONE	Passive	2	Line		3			Left	2
4	C7	d[0]	Input	3	Line		2			Left	1
5	A6	d[1]	Input	4	Line		2			Left	1
6	D7	d[2]	Input	5	Line		2			Left	1
7	B7	d[3]	Input	6	Line		2			Left	1
8	B8	d[4]	Input	7	Line		2			Left	1
9	M7	d[5]	Input	8	Line		4			Left	1
10	A8	d[6]	Input	9	Line		2			Left	1
11	B6	d[7]	Input	10	Line		2			Left	1
12	H2	DATA0	Input	11	Line		1			Left	2
13	K4	DCLK	Bidirectional	12	Line		1			Left	2
14	C6	follow	Output	13	Line		2			Right	1
15	J3	MSEL0	Passive	14	Line		1			Left	2
16	J2	MSEL1	Passive	15	Line		1			Left	2
17	J4	nCE	Passive	16	Line		1			Left	2
18	H4	nCEO	Passive	17	Line		1			Left	2
19	H3	nCONFIG	Passive	18	Line		1			Left	2
20	H5	newt	Input	19	Line		1			Left	1
21	J13	nSTATUS	Passive	20	Line		3			Left	2
22	G16	reset	Input	21	Line		3			Left	1

Each row in the spreadsheet represents a pin in the symbol. The **Section** column indicates the section of the symbol to which each pin is assigned. You can locate all pins in a new symbol in section 1. You can change the values in the **Section** column to assign pins to various sections of the symbol. You can also specify the side of a section on the location of the pin by changing the values in the **Location** column. When you are ready, click **Split**. A new symbol appears in the same library as the original with the name <original part name>_**Split1.**

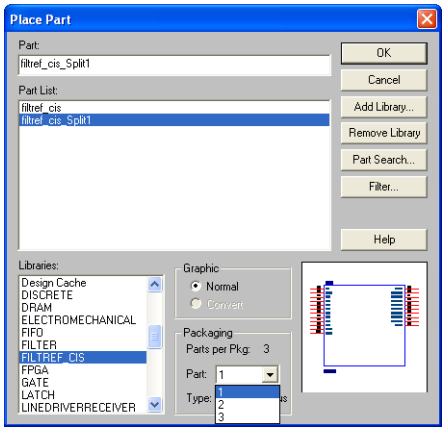
View and edit each section individually. To view the new sections of the part, double-click the part. The Part Symbol Editor window appears and the first section of the part displays for editing. On the View menu, click **Package** to view thumbnails of all the part sections. To edit the section of the symbol, double-click the thumbnail.

For more information about splitting parts into sections and editing symbol sections in the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

4.6.5. Instantiating a Symbol in a Design Entry CIS Schematic

After saving a new symbol in a library in your Cadence Allegro Design Entry CIS project, you can instantiate the new symbol on a page in your schematic. Open a schematic page in the Project Manager window of the Cadence Allegro Design Entry CIS software. To add the new symbol to your schematic on the schematic page, on the Place menu, click **Part**. The **Place Part** dialog box appears.

Figure 27. Place Part Dialog Box



Select the new symbol library location and the newly created part name. If you select a part that is split into sections, you can select the section to place from the **Part** menu. Click **OK**. The symbol attaches to your cursor for placement in the schematic. To place the symbol, click the schematic page.

For more information about using the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

4.6.6. Intel Libraries for the Cadence Allegro Design Entry CIS Software

Intel provides downloadable **`.olb`** for many of its device packages. You can add these libraries to your Cadence Allegro Design Entry CIS project and update the symbols with the pin assignments contained in the **`.pin`** generated by the Quartus Prime software. You can use the downloaded library symbols as a base for creating custom

schematic symbols with your pin assignments that you can edit or fracture. This method increases productivity by reducing the amount of time it takes to create and edit a new symbol.

4.6.6.1. Using the Intel-provided Libraries with your Cadence Allegro Design Entry CIS Project

To use the Intel-provided libraries with your Cadence Allegro Design Entry CIS project, follow these steps:

1. Download the library of your target device from the Download Center page found through the Support page on the Intel website.
2. Create a copy of the appropriate **.olb** to maintain the original symbols. Place the copy in a convenient location, such as your Cadence Allegro Design Entry CIS project directory.
3. In the Project Manager window of the Cadence Allegro Design Entry CIS software, click once on the **Library** folder to select it. On the Edit menu, click **Project** or right-click the **Library** folder and choose **Add File** to select the copy of the downloaded **.olb** and add it to your project. You can locate the new library in the list of part libraries for your project.
4. On the Tools menu, click **Generate Part.** The **Generate Part** dialog box appears.
5. In the **Netlist/source file** field, click **Browse** to specify the **.pin** in your Quartus Prime design.
6. From the **Netlist/source file type** list, select **Altera Pin File**.
7. For **Part name**, type the name of the target device the same as it appears in the downloaded library file. For example, if you are using a device from the **CYCLONE06.OLB** library, type the part name to match one of the devices in this library such as `ep1c6f256`. You can rename the symbol in the Project Manager window after updating the part.
8. Set the **Destination part library** to the copy of the downloaded library you added to the project.
9. Select **Update pins on existing part in library**. Click **OK.**
10. Click **Yes**.

    The symbol is updated with your pin assignments. Double-click the symbol in the Project Manager window to view and edit the symbol. On the View menu, click **Package** if you want to view and edit other sections of the symbol. If the symbol in the downloaded library is fractured into sections, you can edit each section but you cannot further fracture the part. You can generate a new part without using the downloaded part library if you require additional sections.

    For more information about creating, editing, and fracturing symbols in the Cadence Allegro Design Entry CIS software, refer to the Help in the software.

4.7. Cadence Board Design Tools Support Revision History

Table 9. Document Revision History

Date	Quartus Prime Version	Changes
2023.08.01	18.1	• Corrected junk characters in *Cadence and OrCAD Product Comparison* table. • Corrected junk characters in *Symbol Fracture Naming Conventions* table. • Corrected image scaling problem in *Updating the FPGA Symbol in the Design Flow* figure.
2020.11.04	18.1	• Added "Intel" to "Quartus Prime" software reference in a figure.
2019.07.15	18.1	• Retitled Cadence chapter to "Cadence Board Tools Support." • Restructured steps in "Integrating Intel FPGA Designs." • Added document archive.
2018.09.24	18.1	• Document title renamed • Other minor edits
2018.05.07	18.0	• First release as part of the stand-alone *PCB Design Tools User Guide*
2016.10.31	16.1	• Implemented Intel rebranding.
2015.11.02	15.1	• Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0	Converted to DITA format.
June 2012	12.0	Removed survey link.
November 2011	10.0	Template update.
December 2010	10.0	Template update.
July 2010	10.0	• General style editing. • Removed Referenced Document Section. • Added a link to Help in "Performing Simultaneous Switching Noise (SSN) Analysis of Your FPGA" on page 9–5.
November 2009	9.1	• Added "Performing Simultaneous Switching Noise (SSN) Analysis of Your FPGA" on page 9–5. • General style editing. • Edited Figure 9–4 on page 9–10 and Figure 9–8 on page 9–16.
March 2009	9.0	• Chapter 9 was previously Chapter 7 in the 8.1 software release. • No change to content.
November 2008	8.1	Changed to 8-1/2 x 11 page size.
May 2008	8.0	Updated references.

5. Quartus Prime Pro Edition User Guide: PCB Design Tools Document Archives

For the latest and previous versions of this user guide, refer to Quartus Prime Pro Edition User Guide: PCB Design Tools. If an IP or software version is not listed, the user guide for the previous IP or software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel FPGA devices, and program CPLD and configuration devices, via connection with an Intel FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Quartus® Prime Pro Edition User Guide

Scripting

Updated for Quartus® Prime Design Suite: **24.1**

This document is part of a collection - Quartus® Prime Pro Edition User Guides - Combined PDF link

Answers to Top FAQs:

Contents

1. Command Line Scripting

FPGA design software that easily integrates into your design flow saves time and improves productivity. The Quartus® Prime software provides you with a command-line executable for each step of the FPGA design flow to make the design process customizable and flexible.

The command-line executables are completely interchangeable with the Quartus Prime GUI, allowing you to use the exact combination of tools that best suits your needs.

Related Information

- Tcl Design Examples
- TCL Commands and Packages on page 52

1.1. Benefits of Command-Line Executables

Quartus Prime command-line executables give you precise control over each step of the design flow, reduce memory requirements, and improve performance.

You can group Quartus Prime executable files into a script, batch file, or a makefile to automate design flows. These scripting capabilities facilitate the integration of Quartus Prime software and other EDA synthesis, simulation, and verification software. Automatic design flows can perform on multiple computers simultaneously and easily archive and restore projects.

Command-line executables add flexibility without sacrificing the ease-of-use of the Quartus Prime GUI. You can use the Quartus Prime GUI and command-line executables at different stages in the design flow. For example, you might use the Quartus Prime GUI to edit the floorplan for the design, use the command-line executables to perform place-and-route, and return to the Quartus Prime GUI to perform debugging.

Command-line executables reduce the amount of memory required during each step in the design flow. Since each executable targets only one step in the design flow, the executables themselves are relatively compact, both in file size and the amount of memory used during processing. This memory usage reduction improves performance, and is particularly beneficial in design environments where heavy usage of computing resources results in reduced memory availability.

Related Information

About Command-Line Executables
in Quartus Prime Help

1.2. Command-Line Scripting Help

Help for command-line executables is available through different methods. You can access help built into the executables with command-line options. You can use the Quartus Prime Command-Line and Tcl API Help browser for an easy graphical view of the help information.

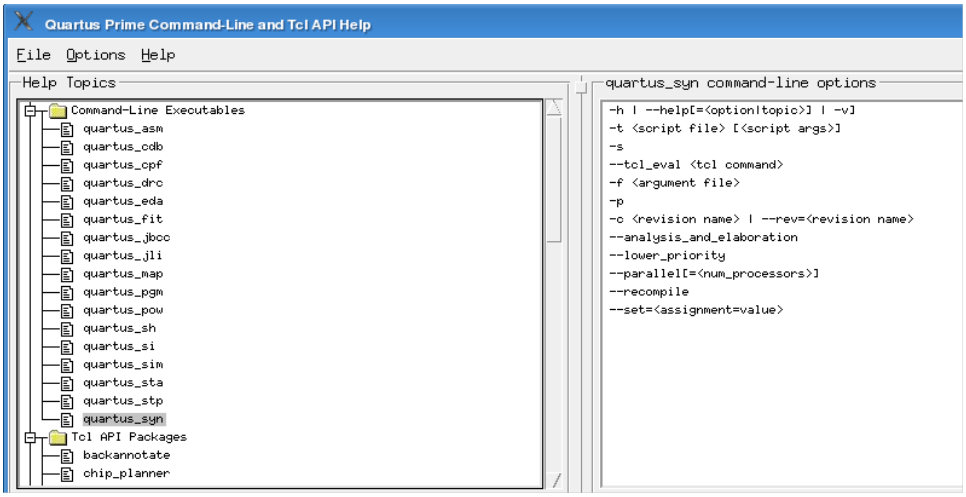
To use the Quartus Prime Command-Line and Tcl API Help browser, type the following command:

```
quartus_sh --qhelp
```

This command starts the Quartus Prime Command-Line and Tcl API Help browser, a viewer for information about the Quartus Prime Command-Line executables and Tcl API.

Use the `-h` option with any of the Quartus Prime Command-Line executables to get a description and list of supported options. Use the `--help=`<*option name*> option for detailed information about each option.

Figure 1. Quartus Prime Command-Line and Tcl API Help Browser



1.3. Project Settings with Command-Line Options

The Quartus Prime software command-line executables accept arguments to set project variables and access common settings.

To make assignments to an individual entity you can use the Quartus Prime Tcl scripting API. On existing projects, you can also open the project in the Quartus Prime GUI, change the assignment, and close the project. The changed assignment is updated in the `.qsf`. Any command-line executables that are run after this update use the updated assignment.

Related Information

- Compilation with quartus_sh --flow on page 8
- Quartus Prime Settings File (.qsf) Definition
  in Quartus Prime Help

- Quartus Prime Pro Edition Settings File Reference Manual

1.3.1. Option Precedence

Project assignments follow a set of precedence rules. Assignments for a project can exist in three places:

- Quartus Prime Settings File (`.qsf`)
- The compiler database
- Command-line options

The `.qsf` file contains all the project-wide and entity-level assignments and settings for the current revision for the project. The compiler database contains the result of the last compilation in the `/db` directory, and reflects the assignments at the moment when the project was compiled. Updated assignments first appear in the compiler database and later in the `.qsf` file.

Command-line options override any conflicting assignments in the `.qsf` file or the compiler database files. To specify whether the `.qsf` or compiler database files take precedence for any assignments not specified in the command-line, use the option `--read_settings_files`.

Table 1. Precedence for Reading Assignments

Option Specified	Precedence for Reading Assignments
`--read_settings_files = on` (Default)	1. Command-line options 2. The `.qsf` for the project 3. Project database (db directory, if it exists) 4. Quartus Prime software defaults
`--read_settings_files = off`	1. Command-line options 2. Project database (db directory, if it exists) 3. Quartus Prime software defaults

The `--write_settings_files` command-line option lists the locations to which assignments are written..

Table 2. Location for Writing Assignments

Option Specified	Location for Writing Assignments
`--write_settings_files = on` (Default)	`.qsf` file and compiler database
`--write_settings_files = off`	Compiler database

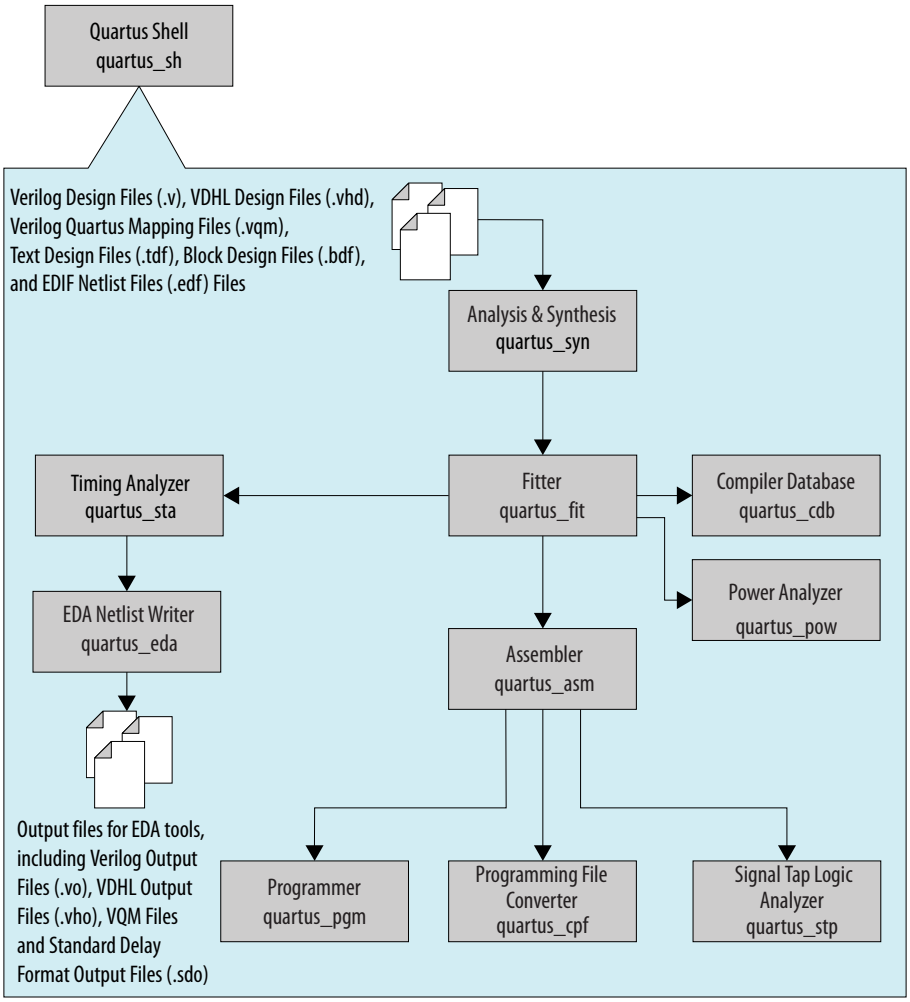
Any assignment not specified as a command-line option or found in the `.qsf` file or compiler database file is set to its default value.

Use the options `--read_settings_files=off` and `--write_settings_files=off` (where appropriate) to optimize the way that the Quartus Prime software reads and updates settings files.

1.4. Compilation with quartus_sh --flow

The figure shows a typical Quartus Prime FPGA design flow using command-line executables.

Figure 2. Typical Design Flow



Use the `quartus_sh` executable with the `--flow` option to perform a complete compilation flow with a single command. The `--flow` option supports the smart recompile feature and efficiently sets command-line arguments for each executable in the flow.

You can resume an interrupted compilation with the `-resume` argument of the `--flow` option.

After you start a compilation flow with the `quartus_sh` executable, you can monitor the progress of the compilation flow in the Quartus Prime Pro Edition GUI.

The following example runs analysis & synthesis, fitter, timing analysis, and programming file generation with a single command:

```
quartus_sh --flow compile filtref
```

Tip: For information about specialized flows, type `quartus_sh --help=flow` at a command prompt.

Related Information

- Resuming a Compilation with quartus_sh --flow on page 9
- Compilation Monitoring in *Intel Quartus Prime Pro Edition User Guide: Design Compilation*

1.4.1. Resuming a Compilation with quartus_sh --flow

You can resume a compilation flow for a project from the last valid step completed in the flow with the `-resume` option of the `quartus_sh --flow` command. If you want to resume a compilation flow, ensure that no settings that affect the subsequent compilation stages have changed from initial start of the compilation.

You can also use the `-start` and `-stop` options of `quartus_sh --flow` command to start and stop a compilation flow at specific compilation tasks.

Resuming a compilation flow also updates the Compilation Dashboard to show how the flow is progressing.

For command syntax and example of how to use the flow resume feature, run `quartus_sh --help=flow` at a command prompt.

Related Information

Using the Compilation Dashboard

1.4.2. Temporarily Overriding the Compiler Optimization Mode

You can run a full compilation flow that temporarily overrides the compiler optimization mode set in the Quartus settings file (`.qsf`) for your design. The optimization mode set for your project in your settings file does not change and remains the default compilation strategy for your project.

Overriding the compiler optimization mode set in your project can be helpful when your project has long compile times and you want to quickly produce a bitstream for on-chip testing.

The following temporary optimization mode compilation flow options are available for the `quartus_sh --flow`, `quartus_fit`, and `quartus_syn` commands:

- `-aggressive_compile_time`

  In the Aggressive Compile Time optimization mode, the Compiler reduces its performance optimization efforts and performs minimal reporting to provide a shorter compilation time.

- `-fast_functional_test`

  In Fast Functional Test optimization mode, the Compiler minimizes its setup-timing optimization efforts to provide an even shorter compilation time.

For example, to compile revision `rev1` of project `top` with a temporary Fast Functional Test optimization mode that overrides the compiler optimization mode set in the project `.qsf` file, issue the following command:

```
quartus_sh --flow compile top -c rev1 -fast_functional_test
```

The selected optimization mode is enabled only for the duration of the compilation. After the compilation completes, Quartus Prime returns to the compilation optimization strategy that is set in the project settings.

Important: With these optimization modes, the clocks in the resulting compilation might not meet setup. You might need to slow down the clocks on your design, such as by using PLL ECOs post-compile, before generating the bitstream.

Related Information

- Full Compilation Flow with Temporary Optimization in *Intel Quartus Prime Pro Edition User Guide: Design Compilation*
- Compiler Optimization Modes in *Intel Quartus Prime Pro Edition User Guide: Design Compilation*

1.5. Text-Based Report Files

Each command-line executable creates a text report file when it is run. These files report success or failure, and contain information about the processing performed by the executable.

Report file names contain the revision name and the short-form name of the executable that generated the report file, in the format `<revision>.<executable>.rpt`. For example, using the `quartus_fit` executable to place and route a project with the revision name **design_top** generates a report file named `design_top.fit.rpt`. Similarly, using the `quartus_sta` executable to perform timing analysis on a project with the revision name **fir_filter** generates a report file named `fir_filter.sta.rpt`.

As an alternative to parsing text-based report files, you can use the `::quartus::report` Tcl package.

Related Information

- Text-Format Report File (.rpt) Definition
  in Quartus Prime Help
- ::quartus::report
  in Quartus Prime Help

1.6. Using Command-Line Executables in Scripts

You can use command-line executables in scripts that control other software, in addition to Quartus Prime software. For example, if your design flow uses third-party synthesis or simulation software, and you can run this other software at the command prompt, you can group those commands with Quartus Prime executables in a single script.

To set up a new project and apply individual constraints, such as pin location assignments and timing requirements, you must use a Tcl script or the Quartus Prime GUI.

Command-line executables are very useful for working with existing projects, for making common global settings, and for performing common operations. For more flexibility in a flow, use a Tcl script. Additionally, Tcl scripts simplify passing data between different stages of the design flow.

For example, you can create a UNIX shell script to run a third-party synthesis software, place-and-route the design in the Quartus Prime software, and generate output netlists for other simulation software.

1.7. Common Scripting Examples

You can create scripts including command line executable to control common Quartus Prime processes.

1.7.1. Create a Project and Apply Constraints

The command-line executables include options for common global project settings and commands. You can use a Tcl script to apply constraints such as pin locations and timing assignments. You can write a Tcl constraint file, or generate one for an existing project by clicking **Project ➤ Generate Tcl File for Project**.

The example creates a project with a Tcl script and applies project constraints using the tutorial design files in the <Quartus Prime *installation directory*>/qdesigns/fir_filter/ directory.

```
project_new filtref -overwrite
# Assign family, device, and top-level file
set_global_assignment -name FAMILY "Arria 10"
set_global_assignment -name DEVICE <Device>
set_global_assignment -name VERILOG_FILE filtref.v
# Assign pins
set_location_assignment -to clk Pin_28
set_location_assignment -to clkx2 Pin_29
set_location_assignment -to d[0] Pin_139
set_location_assignment -to d[1] Pin_140
#
project_close
```

Save the script in a file called `setup_proj.tcl` and type the commands illustrated in the example at a command prompt to create the design, apply constraints, compile the design, and perform fast-corner and slow-corner timing analysis. Timing analysis results are saved in two files, `filtref_sta_1.rpt` and `filtref_sta_2.rpt`.

```
quartus_sh -t setup_proj.tcl
quartus_syn filtref
quartus_fit filtref
quartus_asm filtref
quartus_sta filtref --model=fast --export_settings=off
mv filtref_sta.rpt filtref_sta_1.rpt
quartus_sta filtref --export_settings=off
mv filtref_sta.rpt filtref_sta_2.rpt
```

Type the following commands to create the design, apply constraints, and compile the design, without performing timing analysis:

```
quartus_sh -t setup_proj.tcl
quartus_sh --flow compile filtref
```

The `quartus_sh --flow compile` command performs a full compilation, and is equivalent to clicking the **Start Compilation** button in the toolbar.

1.7.2. Check Design File Syntax

The .tcl script example below assumes the Quartus Prime software **fir_filter** tutorial project exists in the current directory. You can find the **fir_filter** project in the *<Quartus Prime directory>*`/qdesigns/fir_filter` directory unless the Quartus Prime software tutorial files are not installed.

When options are not specified, the executable uses the project database values. If not specified in the project database, the executable uses the Quartus Prime software default values.

To run this script, save this script to a file such as `check_syntax.tcl` and then run the following command from a command prompt: `quartus_syn -t check_syntax.tcl`.

```
set dir [pwd]; # set dir to current working directory

# assign quartus_files variable to all files within current working directory
# asterisk (*) may be changed to specific file extensions (i.e. *.v, *.vhdl,
*.etc)
set quartus_files [glob -directory $dir *]

# open project fir_filter with revision name filtref
project_open fir_filter -revision filtref

foreach file $quartus_files {
      post_message $file;         # echo which file was analyzed
      analyze_files -files $file -library work; # analyze file for syntax
}

project_close; # close project
```

1.7.3. Create a Project and Synthesize a Netlist Using Netlist Optimizations

This example creates a new Quartus Prime project with a file `top.edf` as the top-level entity. The `--enable_register_retiming=on` and `--enable_wysiwyg_resynthesis=on` options cause `quartus_map` to optimize the design using gate-level register retiming and technology remapping.

The `--part` option causes `quartus_syn` to target a device. To create the project and synthesize it using the netlist optimizations described above, type the command shown in this example at a command prompt.

```
quartus_syn top --source=top.edf --enable_register_retiming=on
    --enable_wysiwyg_resynthesis=on --part=<part>
```

1.7.4. Archive and Restore Projects

You can archive or restore an Quartus Prime Archive File (`.qar`) with a single command. This makes it easy to take snapshots of projects when you use batch files or shell scripts for compilation and project management.

Use the `--archive` or `--restore` options for `quartus_sh` as appropriate. Type the command shown in the example at a command prompt to archive your project.

```
quartus_sh --archive <project name>
```

The archive file is automatically named `<project name>.qar`. If you want to use a different name, type the command with the `-output` option as shown in example the example.

```
quartus_sh --archive <project name> -output <filename>
```

To restore a project archive, type the command shown in the example at a command prompt.

```
quartus_sh --restore <archive name>
```

The command restores the project archive to the current directory and overwrites existing files.

Related Information

Managing Quartus Prime Projects

1.7.5. Update Memory Contents Without Recompiling

You can use two commands to update the contents of memory blocks in your design without recompiling. Use the `quartus_cdb` executable with the `--update_mif` option to update memory contents from `.mif` or `.hexout` files. Then, rerun the assembler with the `quartus_asm` executable to regenerate the `.sof`, `.pof`, and any other programming files.

```
quartus_cdb --update_mif <project name> [--rev=<revision name>]
quartus_asm <project name> [--rev=<revision name>]
```

The example shows the commands for a DOS batch file for this example. With a DOS batch file, you can specify the project name and the revision name once for both commands. To create the DOS batch file, paste the following lines into a file called `update_memory.bat`.

```
quartus_cdb --update_mif %1 --rev=%2
quartus_asm %1 --rev=%2
```

To run the batch file, type the following command at a command prompt:

```
update_memory.bat <project name> <revision name>
```

1.7.6. Create Device Configuration Files

You can use the `quartus_cpf` or `quartus_pfg` command line executables to generate different types of device configuration files at the command line, depending on your target device.

- `quartus_pfg`—controls the same programming file generation functions as the **Programming File Generator** dialog box in the Quartus Prime software GUI, and supports programming file generation for Stratix® 10 and Intel Agilex® 7 device families.

Table 3. `quartus_pfg` Command Examples

Command Function	Command Syntax
Specify the ASX4 operation mode, convert `.sof` to `.jic`	`quartus_pfg -c  -o device=MT25QU512 -o mode=ASX4 -o flash_loader=1SG280HN3S3 \ project.sof project.jic`
Access full command-line syntax help	`quartus_pfg --help`

- `quartus_cpf`—controls the same functions as the **Convert Programming Files** dialog box in the Quartus Prime software GUI, and supports programming file generation for all device families prior to the Stratix 10 device family.

Table 4. `quartus_cpf` Command Examples

Command Function	Command Syntax
Create an option file that turns on compression, type the following command at a command prompt	`quartus_cpf -w <filename>.opt`
Create a compressed `.pof` that targets an EPCS64 device	`quartus_cpf --convert --option=<filename>.opt --device=EPCS64 \ <file>.sof <file>.pof`
Save configuration options in a conversion setup file (`.cof`)	`quartus_cpf --convert <file>.cof`
Convert a .sof programming file to CvP periphery image (*.jam) file	`quartus_cpf -c <filename>.sof <filename>.jam --cvp`
Access full command-line syntax help	`quartus_cpf --help`

Note: For complete Quartus Prime command line executable syntax and examples, refer to Command-Line Scripting Help on page 6.

1.7.7. Fit a Design Using Multiple Seeds

This shell script example assumes that the Quartus Prime software tutorial project called **fir_filter** exists in the current directory (defined in the file `fir_filter.qpf`). If the tutorial files are installed on your system, this project exists in the <*Quartus Prime directory*>`/qdesigns`<*quartus_version_number*>` /fir_filter` directory.

Because the top-level entity in the project does not have the same name as the project, you must specify the revision name for the top-level entity with the `--rev` option. The `--seed` option specifies the seeds to use for fitting.

A seed is a parameter that affects the random initial placement of the Quartus Prime Fitter. Varying the seed can result in better performance for some designs.

After each fitting attempt, the script creates new directories for the results of each fitting attempt and copies the complete project to the new directory so that the results are available for viewing and debugging after the script has completed.

```
#!/bin/sh
ERROR_SEEDS=""
quartus_syn fir_filter --rev=filtref
# Iterate over a number of seeds
for seed in 1 2 3 4 5
do
echo "Starting fit with seed=$seed"
# Perform a fitting attempt with the specified seed
quartus_fit fir_filter --seed=$seed --rev=filtref
# If the exit-code is non-zero, the fitting attempt was
# successful, so copy the project to a new directory
if [ $? -eq 0 ]
then
           mkdir ../fir_filter-seed_$seed
           mkdir ../fir_filter-seed_$seed/db
           cp * ../fir_filter-seed_$seed
           cp db/* ../fir_filter-seed_$seed/db
else
           ERROR_SEEDS="$ERROR_SEEDS $seed"
fi
done
if [ -z "$ERROR_SEEDS" ]
then
echo "Seed sweeping was successful"
exit 0
else
echo "There were errors with the following seed(s)"
echo $ERROR_SEEDS
exit 1
fi
```

Tip: Use Design Space Explorer II (DSE) included with the Quartus Prime software script (by typing `quartus_dse` at a command prompt) to improve design performance by performing automated seed sweeping.

1.8. The QFlow Script

A Tcl/Tk-based graphical interface called QFlow is included with the command-line executables. You can use the QFlow interface to open projects, launch some of the command-line executables, view report files, and make some global project assignments.

The QFlow interface can run the following command-line executables:

- `quartus_syn` (Analysis and Synthesis)
- `quartus_fit` (Fitter)
- `quartus_sta` (Timing Analyzer)
- `quartus_asm` (Assembler)
- `quartus_eda` (EDA Netlist Writer)

  To view floorplans or perform other GUI-intensive tasks, launch the Quartus Prime software.

  Start QFlow by typing the following command at a command prompt:

  ```
  quartus_sh -g
  ```

  Tip: The QFlow script is located in the *<Quartus Prime directory>*`/common/tcl/apps/qflow/` directory.

1.8.1. --partition Option

The `--partition` option is for the `--simulation` top-level argument for `quartus_eda`. This option selects an individual partition as the netlist output.

`--exclude_sub_partitions`

The `--exclude_sub_partitions` flag limits the output to the netlist of this partition only. This flag is only valid when you use the `--partition` option, this flag outputs the netlist belonging to the partition specified. The software instantiates sub-partitions as separate modules.

For no partition argument, the software writes the entire design out to a single file. The partition argument takes a name of a partition in the design. The sub-option is a flag only and takes no arguments.

When you specify the `--exclude_sub_partitions` flag, the software only writes out the contents of the selected partition. Sub-partitions are instantiated as separate modules. Each call of `quartus_eda` writes one netlist. If you write out the design one partition at a time, excluding sub partitions, they need to call `quartus_eda` for each partition in the design including the root.

`root_partition`

You can specify the `root_partition` flag to get the full design. You can provide the partition option to write to a netlist file (`.vo` or `.vho` file). The file contains all the logic and atoms corresponding the contents of the specified partition along with its sub-partition.

`--rename`

Additionally, you have the option to rename a module name in the generated netlist file using the `--rename` option. By default, the software uses the partition name as the module name in the netlist file. This option is only valid when you use the partition option. You can elect to rename any module using `--module_name=abc=xyz --module_name=def=prq`. The generated file names format is: *<revision>.<module*

or partition name>.<vo or vho> By default, the software writes the netlist file to the `simulation/<3rd party simulation tool>` directory (for example, `simulation/modelsim`), unless you specify an `output_directory` (using a command line option or `.qsf` assignment).

1.9. Command-Line Scripting Revision History

The following revision history applies to this chapter:

Table 5. Document Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied phase I Altera rebranding throughout.
2022.04.03	23.1	• Updated name of the Agilex 7 device family.
2022.09.26	22.3	• Added Top FAQs navigation to document cover.
2022.03.28	22.1	• Added "Temporarily Overriding the Compiler Optimization Mode" • Added information about monitoring a compilation to "Compilation with quartus_sh --flow"
2021.03.29	21.1	• Added "Resuming a Compilation with `quartus_sh --flow`" • Revised "Check Design File Syntax".
2020.12.14	20.4	• Updated "Create Device Configuration Files" to include references to the cvp command option and the quartus_pfg command.
2020.04.13	20.1	Added *--partition option*.
2017.05.08	17.0.0	• Reorganized content on topics: Benefits of Command-Line Executables and Project Settings with Command-Line Options. • Removed mentions to unsupported executables and options. • Removed topics: Introductory Example and Common Scripting Examples
2016.10.31	16.1.0	• Implemented Intel rebranding.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Remove descriptions of makefile support that was removed from software in 14.0.
December 2014	14.1.0	Updated DSE II commands.
June 2014	14.0.0	Updated formatting.
November 2013	13.1.0	Removed information about -silnet `qmegawiz` command
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	Corrected quartus_qpf example usage. Updated examples.
December 2010	10.1.0	Template update. Added section on using a script to regenerate megafunction variations. Removed references to the Classic Timing Analyzer (quartus_tan). Removed Qflow illustration.
July 2010	10.0.0	Updated script examples to use quartus_sta instead of quartus_tan, and other minor updates throughout document.

continued...

Document Version	Quartus Prime Version	Changes
November 2009	9.1.0	Updated Table 2–1 to add quartus_jli and quartus_jbcc executables and descriptions, and other minor updates throughout document.
March 2009	9.0.0	No change to content.
November 2008	8.1.0	Added the following sections: • "The MegaWizard Plug-In Manager" on page 2–11 "Command-Line Support" on page 2–12 "Module and Wizard Names" on page 2–13 "Ports and Parameters" on page 2–14 "Invalid Configurations" on page 2–15 "Strategies to Determine Port and Parameter Values" on page 2–15 "Optional Files" on page 2–15 "Parameter File" on page 2–16 "Working Directory" on page 2–17 "Variation File Name" on page 2–17 • "Create a Compressed Configuration File" on page 2–21 • Updated "Option Precedence" on page 2–5 to clarify how to control precedence • Corrected Example 2–5 on page 2–8 • Changed Example 2–1, Example 2–2, Example 2–4, and Example 2–7 to use the EP1C12F256C6 device • Minor editorial updates • Updated entire chapter using 8½" × 11" chapter template
May 2008	8.0.0	• Updated "Referenced Documents" on page 2–20. • Updated references in document.

2. Tcl Scripting

You can use Tcl scripts, as an alternative to the GUI, to control the function and operation of Quartus Prime software.

For example, you can use Tcl scripts to perform the following tasks:

- Manage Quartus Prime projects
- Specify assignments and constraints
- Compile your design
- Perform timing analysis
- Generate and view reports about your project

You can also use Tcl scripts to migrate a project or project settings. For example, when working with different projects targeting the same prototype or development board, you can define a Tcl script to automate pin assignments for each project, rather than entering the assignments individually in the GUI. You can automatically generate a Tcl script based on current project assignments, which simplifies transferring the assignments to another project.

The Quartus Prime software Tcl commands follow familiar EDA industry Tcl application programming interface (API) standards for command-line options. If you encounter an error with a command argument, the Tcl interpreter includes help information showing correct usage.

This chapter includes sample Tcl scripts for automating tasks in the Quartus Prime software, along with a complete reference of all supported Tcl commands and arguments. You can modify the example scripts for use with your own designs. Refer to Design Examples section of the Support area on the Intel website.

Related Information

- Tcl Design Examples
- TCL Commands and Packages on page 52

2.1. Tool Command Language

Tcl (pronounced “tickle”) stands for Tool Command Language, and is the industry-standard scripting language. Tcl supports control structures, variables, network socket access, and APIs.

With Tcl, you can work seamlessly across most development platforms. Synopsys*, Mentor Graphics*, and Intel software products support the Tcl language.

By combining Tcl commands and Quartus Prime API functions, you can create your own procedures and automate your design flow. Run Quartus Prime software in batch mode, or execute individual Tcl commands interactively in the Quartus Prime Tcl shell.

Quartus Prime software supports Tcl/Tk version 8.5, supplied by the Tcl DeveloperXchange.

2.2. The Quartus Prime Tcl Console Window

A Tcl Console Window is available in the Quartus Prime software GUI by clicking **View ➤ Tcl Console**. You can run Quartus Prime Tcl commands directly in the **Tcl Console** window. The Quartus Prime Tcl shell interprets all Tcl commands that you type in the **Tcl Console**.

In addition, when you run commands in the Quartus Prime GUI, the Tcl Console displays the equivalent Tcl command. You can right-click in the Tcl Console and click **Save to File** to save a log file of the Tcl commands in the Tcl Console.

Note: Some shell commands such as `cd`, `ls`, and others can be run in the Tcl Console window, with the Tcl `exec` command. However, for best results, run shell commands and Quartus Prime executables from a system command prompt outside of the Quartus Prime software GUI.

Tcl messages appear in the **System** tab (**Messages** window). Errors and messages written to `stdout` and `stderr` also are shown in the Quartus Prime **Tcl Console** window.

2.3. Quartus Prime Tcl Packages

The Quartus Prime software groups Tcl commands into packages by function.

Note: Refer to TCL Commands and Packages on page 52 for a comprehensive reference of all Quartus Prime Tcl packages and commands.

Table 6. Quartus Prime Tcl Packages

Package Name	Package Description
`backannotate`	Back-annotate the Compiler's assignments.
`bpps`	Floorplan IP interfaces and other device resources in Interface Planner.
`chip_planner`	Identify and modify resource usage and routing with the Chip Planner.
`design`	Manipulate project databases, including the assignments database, to enable the creation of instance assignments without modifying the `.qsf` file.
`device`	Get device and family information from the device database.
`dni_sdc`	Set false path, input delay, or output delay SDC constraints.
`drc`	Interact with Design Assistant design rule checks.
`eco`	Specify engineering change orders after design compilation.
`external_memif_toolkit`	Interact with external memory interfaces and debug components.
`fif`	Contains the set of Tcl functions for using the Fault Injection File (FIF) Driver
`flng`	Query properties of generic objects.
`flow`	Compile a project, run command-line executables, and other compilation flows.
`help`	Tcl command help.

continued...

Package Name	Package Description
insystem_memory_edit	Read and edit memory contents in Intel FPGA devices.
insystem_source_probe	Interact with the In-System Sources and Probes tool in an Intel device.
interactive_synthesis	Interactive synthesis controls.
ipgen	IP generation controls.
iptclgen	Memory IP generation controls.
jtag	Control the JTAG chain.
logic_analyzer_interface	Query and modify the Logic Analyzer Interface output pin state.
misc	Perform miscellaneous tasks such as enabling natural bus naming, package loading, and message posting.
names	Gets or sets assignment names.
periph	Interact with the interface pins.
pfg	Controls the Programming File Generator.
project	Create and manage projects and revisions, make any project assignments including timing assignments.
project_ui	Query the GUI.
qshm	Client and server controls.
report	Get information from report tables, create custom reports.
rtl	Traverse and query the RTL netlist of your design.
sdc	Specify constraints and exceptions to the Timing Analyzer.
sdc_ext	Intel FPGA-specific SDC commands.
sta	Contains the set of Tcl functions for obtaining advanced information from the Timing Analyzer.
stp	Run the Signal Tap Logic Analyzer.
tdc	Obtain information from the Timing Analyzer.

To keep memory requirements as low as possible, only the minimum number of packages load automatically with each Quartus Prime executable. To run commands from other packages, load those packages beforehand.

Run your scripts with executables that include the packages you use in the scripts. For example, to use commands in the `sdc_ext` package, you must use the `quartus_sta` executable because `quartus_sta` is the only executable with support for the `sdc_ext` package.

The following command prints lists of the packages loaded or available to load for an executable, to the console:

`<executable name> --tcl_eval help`

For example, type the following command to list the packages loaded or available to load by the `quartus_fit` executable:

```
quartus_fit --tcl_eval help
```

Related Information

- Tcl Design Examples
- TCL Commands and Packages on page 52

2.3.1. Loading Tcl Packages

To load an Quartus Prime Tcl package, use the `load_package` command as follows:

```
load_package [-version <version number>] <package name>
```

This command is similar to `package require`, but it allows to alternate between different versions of an Quartus Prime Tcl package.

2.3.2. Quartus Prime Tcl API Help

Quartus Prime Tcl help allows easy access to information about the Quartus Prime Tcl commands.

- This command opens the Quartus Prime Command-Line and Tcl API help browser, which documents all commands and options in the Quartus Prime Tcl API. At a system command prompt, access the Quartus Prime Tcl API Help by typing:

  ```
  quartus_sh --qhelp
  ```

- The Tcl API Help can be accessed from the Tcl console as well. At a Tcl prompt, type

  ```
  help
  ```

  to access the help information. The output is:

The Tcl console provides help options that display specific information:

Table 7. Help Options Available in the Quartus Prime Tcl Environment

Help Command	Description
`help`	Displays complete list of available Quartus Prime Tcl packages.
`help -tcl`	Explains how to load Tcl packages and access command-line help.
`help -pkg <package_name -[-version <version number>]`	Displays help commands of the Quartus Prime package that you specify, including the list of available Tcl commands. • If you do not specify `-version`, the Quartus Prime software loads the latest version of the package. • If the package is not loaded, the Quartus Prime software displays the help for the latest version of the package. Examples: `help -pkg ::quartus::project` `help -pkg project` `help -pkg project -version 1.0`

continued...

Help Command	Description
`<command_name> -h` or `<command_name> -help`	Displays the short help of a Quartus Prime Tcl command in a loaded package. Examples: `project_open -h` `project_open -help`
`package require ::quartus::<package name>[<version>]`	Loads a specific version of an Quartus Prime Tcl package. If you do not specify `-version`, the Quartus Prime software loads the latest version of the package. Example: `package require ::quartus::project 1.0` This command is similar to the `load_package` command
`load_package <package name> [-version <version number>]`	Allows you to alternate between different versions of the same package. Example: `load_package ::quartus::project -version 1.0`
`help -cmd <command_name>` `-[-version <version>]` or `<command_name> -long_help`	Displays the complete help text for an Quartus Prime Tcl command. If you do not specify `-version`, the Quartus Prime software loads the latest version of the package. Examples: `project_open -long_help` `help -cmd project_open` `help -cmd project_open -version 1.0`
`help -examples`	Displays examples of Quartus Prime Tcl usage.
`help -quartus`	To view help on the predefined global Tcl array that contains project information and information about the Quartus Prime executable that is currently running.
`quartus_sh --qhelp`	Launches the Tk viewer for Quartus Prime command-line help and display help for the command-line executables and Tcl API packages.
`help -timequestinfo`	To view help on the predefined global `"TimeQuestInfo"` Tcl array that contains delay model information and speed grade information of a Timing Analyzer design that is currently running.

The Tcl API help is also available in Quartus Prime online help. Search for the command or package name to find details about that command or package.

2.3.2.1. Command-Line Options

You can use any of the following command line options with executables that support Tcl:

Table 8. Command-Line Options Supporting Tcl Scripting

Command-Line Option	Description
`--script=<script file> [<script args>]`	Run the specified Tcl script with optional arguments.
`-t <script file> [<script args>]`	Run the specified Tcl script with optional arguments. The `-t` option is the short form of the `--script` option.
`--shell`	Open the executable in the interactive Tcl shell mode.
`-s`	Open the executable in the interactive Tcl shell mode. The `-s` option is the short form of the `--shell` option.
`--tcl_eval <tcl command>`	Evaluate the remaining command-line arguments as Tcl commands. For example, the following command displays help for the project package: `quartus_sh --tcl_eval help -pkg project`

2.3.2.1.1. Run a Tcl Script

Running an executable with the `-t` option runs the specified Tcl script. You can also specify arguments to the script. Access the arguments through the `argv` variable, or use a package such as `cmdline`, which supports arguments of the following form:

```
-<argument name> <argument value>
```

The `cmdline` package is included in the *<Quartus Prime directory>*`/common/tcl/packages` directory.

For example, to run a script called `myscript.tcl` with one argument, `Stratix`, type the following command at a system command prompt:

```
quartus_sh -t myscript.tcl Stratix
```

2.3.2.1.2. Interactive Shell Mode

Running an executable with the `-s` option starts an interactive Tcl shell. For example, to open the Quartus Prime Timing Analyzer executable in interactive shell mode, type:

```
quartus_sta -s
```

Commands you type in the Tcl shell are interpreted when you press Enter. To run a Tcl script in the interactive shell type:

```
source <script name>
```

If a command is not recognized by the shell, it is assumed to be external and executed with the `exec` command.

2.3.2.1.3. Evaluate as Tcl

Running an executable with the `--tcl_eval` option causes the executable to immediately evaluate the remaining command-line arguments as Tcl commands. This can be useful if you want to run simple Tcl commands from other scripting languages.

For example, the following command runs the Tcl command that prints out the commands available in the project package.

```
quartus_sh --tcl_eval help -pkg project
```

2.4. Tcl Design Flow Controls

You can use Tcl scripts to control all aspects of the design flow, including controlling other software, when the other software also includes a scripting interface.

Typically, EDA tools include their own script interpreters that extend core language functionality with tool-specific commands. For example, the Quartus Prime Tcl interpreter supports all core Tcl commands, and adds numerous commands specific to the Quartus Prime software. You can include commands in one Tcl script to run another script, which allows you to combine or chain together scripts to control different tools. Because scripts for different tools must be executed with different Tcl interpreters, it is difficult to pass information between the scripts unless one script writes information into a file and another script reads it.

Within the Quartus Prime software, you can perform many different operations in a design flow (such as synthesis, fitting, and timing analysis) from a single script, making it easy to maintain global state information and pass data between the operations. However, there are some limitations on the operations you can perform in a single script due to the various packages supported by each executable.

There are no limitations on running flows from any executable. Flows include operations found in

Processing ➤ Start in the Quartus Prime GUI, and are also documented as options for the `execute_flow` Tcl command. If you can make settings in the Quartus Prime software and run a flow to get your desired result, you can make the same settings and run the same flow in a Tcl script.

2.4.1. Creating Projects and Making Assignments

You can create a script that makes all the assignments for an existing project, and then use the script at any time to restore your project settings to a known state.

Click **Project ➤ Generate Tcl File for Project** to automatically generate a `.tcl` file containing your assignments. You can source this file to recreate your project, and you can add other commands to this file, such as commands for compiling the design. This file is a good starting point to learn about project management and assignment commands.
To commit the assignments you create or modify to the `.qsf` file, you use the `export_assignments` or `project_close` commands. However, when you run the `execute_flow` command, Quartus Prime software automatically commits the assignment changes to the `.qsf` file. To prevent this behavior, specify the `-dont_export_assignments` logic option.

Related Information

- Quartus Prime Pro Edition Settings File Reference Manual
- Interactive Shell Mode on page 24
- Constraining Designs

2.4.2. Compiling Designs

You can run the Quartus Prime command-line executables from Tcl scripts. Use the included `flow` package to run various Quartus Prime compilation flows, or run each executable directly.

2.4.2.1. The flow Package

The `flow` package includes two commands for running Quartus Prime command-line executables, either individually or together in standard compilation sequence.

- The `execute_module` command allows you to run an individual Quartus Prime command-line executable.
- The `execute_flow` command allows you to run some or all the executables in commonly-used combinations.

Use the `flow` package instead of system calls to run Quartus Prime executables from scripts or from the Quartus Prime Tcl Console.

2.4.2.2. Compile All Revisions

You can use a simple Tcl script to compile all revisions in your project. Save the following script in a file called `compile_revisions.tcl` and type the following to run it:

```
quartus_sh -t compile_revisions.tcl <project name>
```

Compile All Revisions

```
load_package flow project_open [lindex $quartus(args) 0] set original_revision
[get_current_revision] foreach revision [get_project_revisions]
{ set_current_revision $revision execute flow -compile } set_current_revision
$original_revision project_close
```

2.4.3. Reporting

You can extract information from the Compilation Report to evaluate results. The Quartus Prime Tcl API provides easy access to report data so you do not have to write scripts to parse the text report files.

If you know the exact report cell or cells you want to access, use the `get_report_panel_data` command and specify the row and column names (or `x` and `y` coordinates) and the name of the appropriate report panel. You can often search for data in a report panel. To do this, use a loop that reads the report one row at a time with the `get_report_panel_row` command.

Column headings in report panels are in row 0. If you use a loop that reads the report one row at a time, start with row 1 to skip column headings. The `get_number_of_rows` command returns the number of rows in the report panel, including the column heading row. Since the number of rows includes the column heading row, continue your loop if the loop index is less than the number of rows.

Report panels are hierarchically arranged and each level of hierarchy is denoted by the string "||" in the panel name. For example, the name of the Fitter Settings report panel is `Fitter||Fitter Settings` because it is in the `Fitter` folder. Panels at the highest hierarchy level do not use the "||" string. For example, the Flow Settings report panel is named `Flow Settings`.

The following Tcl code prints a list of all report panel names in your project. You can run this code with any executable that includes support for the report package.

Print All Report Panel Names

```
load_package report
project_open myproject
load_report
set panel_names [get_report_panel_names]
foreach panel_name $panel_names {
post_message "$panel_name"
}
```

2.4.3.1. Saving Report Data in csv Format

You can create a Comma Separated Value (`.csv`) file from any Quartus Prime report to open with a spreadsheet editor.

The following Tcl code shows a simple way to create a `.csv` file with data from the Fitter panel in a report.

Create .csv Files from Reports

```
load_package report
project_open my-project
load_report
# This is the name of the report panel to save as a CSV file
set panel_name "Fitter||Fitter Settings"
set csv_file "output.csv"
set fh [open $csv_file w]
set num_rows [get_number_of_rows -name $panel_name]
# Go through all the rows in the report file, including the
# row with headings, and write out the comma-separated data
for { set i 0 } { $i < $num_rows } { incr i } {
    set row_data [get_report_panel_row -name $panel_name \
        -row $i]
    puts $fh [join $row_data ","]
}
close $fh
unload_report
```

You can modify the script to use command-line arguments to pass in the name of the project, report panel, and output file to use. You can run this script example with any executable that supports the report package.

2.4.4. Timing Analysis

The Quartus Prime Timing Analyzer includes support for industry-standard SDC commands in the `sdc` package.

The Quartus Prime software includes comprehensive Tcl APIs and SDC extensions for the Timing Analyzer in the `sta`, and `sdc_ext` packages. The Quartus Prime software also includes a `tdc` package that obtains information from the Timing Analyzer.

Related Information

Quartus Prime Pro Edition Settings File Reference Manual

2.5. Automating Script Execution

You can configure scripts to run automatically at various points during compilation. Use this capability to automatically run scripts that perform custom reporting, make specific assignments, and perform many other tasks.

The following three global assignments control when a script is run automatically:

- `PRE_FLOW_SCRIPT_FILE` —before a flow starts
- `POST_MODULE_SCRIPT_FILE` —after a module finishes
- `POST_FLOW_SCRIPT_FILE` —after a flow finishes

A module is another term for an Quartus Prime executable that performs one step in a flow. For example, two modules are Analysis and Synthesis (`quartus_syn`), and timing analysis (`quartus_sta`).

A flow is a series of modules that the Quartus Prime software runs with predefined options. For example, compiling a design is a flow that typically consists of the following steps (performed by the indicated module):

1. Analysis and Synthesis (`quartus_syn`)
2. Fitter (`quartus_fit`)
3. Assembler (`quartus_asm`)
4. Timing Analyzer (`quartus_sta`)

Other flows are described in the help for the `execute_flow` Tcl command. In addition, many commands in the **Processing** menu of the Quartus Prime GUI correspond to this design flow.

To make an assignment automatically run a script, add an assignment with the following form to the `.qsf` for your project:

```
set_global_assignment -name <assignment name> <executable>:<script name>
```

The Quartus Prime software runs the scripts.

```
<executable> -t <script name> <flow or module name> <project name> <revision
name>
```

The first argument passed in the `argv` variable (or `quartus(args)` variable) is the name of the flow or module being executed, depending on the assignment you use. The second argument is the name of the project and the third argument is the name of the revision.

The last process, current project, and current revision are passed to the script by the Quartus Prime software and can be accessed by the following commands:

```
set process  [lindex $quartus(args) 0]
set project  [lindex $quartus(args) 1]
set revision [lindex $quartus(args) 2]

project_open $project -revision $revision
```

When you use the POST_MODULE_SCRIPT_FILE assignment, the specified script is automatically run after every executable in a flow. You can use a string comparison with the module name (the first argument passed in to the script) to isolate script processing to certain modules.

2.5.1. Execution Example

To illustrate how automatic script execution works in a complete flow, assume you have a project called **top** with a current revision called **rev_1**, and you have the following assignments in the `.qsf` for your project.

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE quartus_sh:first.tcl
set_global_assignment -name POST_MODULE_SCRIPT_FILE quartus_sh:next.tcl
set_global_assignment -name POST_FLOW_SCRIPT_FILE quartus_sh:last.tcl
```

When you compile your project, the PRE_FLOW_SCRIPT_FILE assignment causes the following command to be run before compilation begins:

```
quartus_sh -t first.tcl compile top rev_1
```

Next, the Quartus Prime software starts compilation with analysis and synthesis, performed by the `quartus_syn` executable. After the Analysis and Synthesis finishes, the POST_MODULE_SCRIPT_FILE assignment causes the following command to run:

```
quartus_sh -t next.tcl quartus_syn top rev_1
```

Then, the Quartus Prime software continues compilation with the Fitter, performed by the `quartus_fit` executable. After the Fitter finishes, the POST_MODULE_SCRIPT_FILE assignment runs the following command:

```
quartus_sh -t next.tcl quartus_fit top rev_1
```

Corresponding commands are run after the other stages of the compilation. When the compilation is over, the POST_FLOW_SCRIPT_FILE assignment runs the following command:

```
quartus_sh -t last.tcl compile top rev_1
```

2.5.2. Controlling Processing

The POST_MODULE_SCRIPT_FILE assignment causes a script to run after every module. Because the same script is run after every module, you might have to include some conditional statements that restrict processing in your script to certain modules.

For example, if you want a script to run only after timing analysis, use a conditional test like the following example. It checks the flow or module name passed as the first argument to the script and executes code when the module is `quartus_sta`.

Restrict Processing to a Single Module

```
set module [lindex $quartus(args) 0]
if [string match "quartus_sta" $module] {
    # Include commands here that are run
    # after timing analysis
    # Use the post-message command to display
```

```
    # messages
    post_message "Running after timing analysis"
}
```

2.5.3. Displaying Messages

Because of the way the Quartus Prime software runs the scripts automatically, you must use the `post_message` command to display messages, instead of the `puts` command. This requirement applies only to scripts that are run by the three assignments listed in "Automating Script Execution".

Related Information

- The post_message Command on page 41
- Automating Script Execution on page 28

2.6. Other Scripting Features

The Quartus Prime Tcl API includes other general-purpose commands and features described in this section.

2.6.1. Natural Bus Naming

The Quartus Prime software supports natural bus naming. Natural bus naming allows you to use square brackets to specify bus indexes in HDL, without including escape characters to prevent Tcl from interpreting the square brackets as containing commands. For example, one signal in a bus named `address` can be identified as `address[0]` instead of `address\[0\]`. You can take advantage of natural bus naming when making assignments.

```
set_location_assignment -to address[10] Pin_M20
```

The Quartus Prime software defaults to natural bus naming. You can turn off natural bus naming with the `disable_natural_bus_naming` command. For more information about natural bus naming, type the following at an Quartus Prime Tcl prompt:

```
enable_natural_bus_naming -h
```

2.6.2. Short Option Names

You can use short versions of command options, if they are unambiguous. For example, the `project_open` command supports two options: `-current_revision` and `-revision`.

You can use any of the following abbreviations of the `-revision` option:

- `-r`
- `-re`
- `-rev`

- `-revi`
- `-revis`
- `-revisio`

You can use an extremely short option such as `-r` because in the case of the `project_open` command no other option starts with the letter r. However, the `report_timing` command includes the options `-recovery` and `-removal`. You cannot use `-r` or `-re` to shorten either of those options, because the abbreviation is not unique.

2.6.3. Collection Commands

Some Quartus Prime Tcl functions return very large sets of data that are inefficient as Tcl lists. These data structures are referred to as collections. The Quartus Prime Tcl API uses a collection ID to access the collection.

There are two Quartus Prime Tcl commands for working with collections, `foreach_in_collection` and `get_collection_size`. Use the `set` command to assign a collection ID to a variable.

2.6.3.1. The foreach_in_collection Command

The `foreach_in_collection` command is similar to the `foreach` Tcl command. Use it to iterate through all elements in a collection. The following example prints all instance assignments in an open project.

foreach_in_collection Example

```
set all_instance_assignments [get_all_instance_assignments -name *]
foreach_in_collection asgn $all_instance_assignments {
    # Information about each assignment is
    # returned in a list. For information
    # about the list elements, refer to Help
    # for the get-all-instance-assignments command.
    set to [lindex $asgn 2]
    set name [lindex $asgn 3]
    set value [lindex $asgn 4]
    puts "Assignment to $to: $name = $value"
}
```

Related Information

foreach_in_collection (::quartus::misc)
In Quartus Prime Help

2.6.3.2. The get_collection_size Command

Use the `get_collection_size` command to get the number of elements in a collection. The following example prints the number of global assignments in an open project.

get_collection_size Example

```
set all_global_assignments [get_all_global_assignments -name *]
set num_global_assignments [get_collection_size $all_global_assignments]
puts "There are $num_global_assignments global assignments in your project"
```

2.6.4. Node Finder Commands

The Node Finder allows you to find any node name in your project's compilation database. You can then perform various actions on found nodes, such as specifying constraints or assignments to those nodes. You can filter the search on various criteria, and also use wildcard characters in the search string.

A complete set of Node Finder Tcl commands that support the equivalent Node Finder filtering options is available for use in the scripted design flow environment.

The filtering options include the following default filters that appear in the filter combo box in the Node Finder:

2.6.4.1. Design Entry (all names) Filter

This Node Finder filter finds all user-entered names in your design.

The following Tcl command demonstrates the use of the `Design Entry (all names)` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.2. Pins: assigned Filter

This Node Finder filter finds all pin names assigned locations or other pin-related assignments.

The following Tcl command demonstrates the use of the `Pins: assigned` filtering option:

```
set name_ids_col [get_names -filter * -node_type assigned \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.3. Pins: unassigned Filter

This Node Finder filter finds all pin names unassigned locations or other pin related assignments.

The following Tcl command demonstrates the use of the `Pins: unassigned` filtering option:

```
set name_ids_col [get_names -filter * -node_type unassigned \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.4. Pins: input Filter

This Node Finder filter finds all input pin names in your design files.

The following Tcl command demonstrates the use of the `Pins: input` filtering option:

```
set name_ids_col [get_names -filter * -node_type input \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.5. Pins: output Filter

This Node Finder filter finds all output pin names in your design files.

The following Tcl command demonstrates the use of the `Pins: output` filtering option:

```
set name_ids_col [get_names -filter * -node_type output \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.6. Pins: bidirectional Filter

This Node Finder filter finds all bidirectional pin names in your design files.

The following Tcl command demonstrates the use of the `Pins: bidirectional` filtering option:

```
set name_ids_col [get_names -filter * -node_type bidir \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.7. Pins: virtual Filter

This Node Finder filter finds names of all I/O elements mapped to logic elements with a Virtual Pin logic option assignment.

The following Tcl command demonstrates the use of the `Pins: virtual` filtering option:

```
set name_ids_col [get_names -filter * -node_type virtual \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.8. Pins: all Filter

This Node Finder filter finds all pin names in your design files.

The following Tcl command demonstrates the use of the `Pins: all` filtering option:

```
set name_ids_col [get_names -filter * -node_type \
pin -observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.9. Pins: all & Registers: post-fitting Filter

This Node Finder filter finds all pin names in your design along with all register names from your design files that persist after physical synthesis and fitting. The `Pins: all & Registers: post-fitting` filter is a combination of the `Pins: all` and `Registers: post-fitting` filters.

The following Tcl command demonstrates the use of the `Pins: all & Registers: post-fitting` filtering option:

```
set name_ids_col [get_names -filter * -node_type \
all_reg -observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type post_fitter \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.10. Ports: partition

This Node Finder filter must be used after running the Fitter, to find nodes for post-fit partition.

Note: When you run this filter before running the Fitter, a "No nodes available. Run Fitter." message displays.

The following Tcl command demonstrates the use of the `Ports: partition` filtering option:

```
set name_ids_col [get_names -filter * -node_type partition \
-observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type post_fitter \
$name_id]
     append name ","
```

```
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.11. Entity instance: pre-synthesis Filter

This Node Finder filter finds a list of instances in the logical hierarchy for pre-synthesis netlist.

The following Tcl command demonstrates the use of the `Entity instance: pre-synthesis` filtering option:

```
set name_ids_col [get_names -filter * -node_type hierarchy \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.12. Registers: pre-synthesis Filter

This Node Finder filter finds all register names you entered in the design after Analysis and Elaboration, but before physical synthesis performs any synthesis optimizations.

The following Tcl command demonstrates the use of the `Registers: pre-synthesis` filtering option:

```
set name_ids_col [get_names -filter * -node_type reg \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type pre_synthesis \
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.13. Registers: post-fitting Filter

This Node Finder filter finds all user-entered register names in your design files that remain after physical synthesis and fitting.

The following Tcl command demonstrates the use of the `Registers: post-fitting` filtering option:

```
set name_ids_col [get_names -filter * -node_type reg \
-observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
      set name [get_name_info -info full_path -observable_type post_fitter \
```

```
$name_id]
      append name ","
      append name [get_name_info -info node_type $name_id]
      puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.14. Post-synthesis Filter

This Node Finder filter finds all user-entered and synthesis-generated names that remain in the design after design elaboration and physical synthesis.

The following Tcl command demonstrates the use of the `Post-Synthesis` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type post_synthesis]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type post_synthesis \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.15. Post-Compilation Filter

This Node Finder filter finds all user-centered and Compiler-generated names that remain after fitting and do not have location assignments.

The following Tcl command demonstrates the use of the `Post-Compilation` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
     set name [get_name_info -info full_path -observable_type post_fitter \
$name_id]
     append name ","
     append name [get_name_info -info node_type $name_id]
     puts $name
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.16. Signal Tap: pre-synthesis Filter

This Node Finder filter finds all internal device nodes in the pre-synthesis netlist that can be analyzed by the Signal Tap Logic Analyzer.

The following Tcl command demonstrates the use of the `Signal Tap: pre-synthesis` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type pre_synthesis]
foreach_in_collection name_id $name_ids_col {
     set is_signaltap [get_name_info -info signaltapii -observable_type \
pre_synthesis $name_id]
     if {$is_signaltap == 1} {
          set name [get_name_info -use_cached_database -info full_path \
-observable_type pre_synthesis $name_id]
          append name ","
          append name [get_name_info -info node_type -observable_type \
pre_synthesis $name_id]
          puts $name
     }
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.4.17. Signal Tap: post-fitting Filter

This Node Finder filter finds all internal device nodes in the post fit netlist that can be analyzed by the Signal Tap Logic Analyzer.

The following Tcl command demonstrates the use of the `Signal Tap: post-fitting` filtering option:

```
set name_ids_col [get_names -filter * -node_type all \
-observable_type post_fitter]
foreach_in_collection name_id $name_ids_col {
    set is_signaltap [get_name_info -info signaltapii -observable_type \
post_fitter $name_id]
    if {$is_signaltap == 1} {
        set name [get_name_info -use_cached_database -info full_path \
-observable_type post_fitter $name_id]
        append name ","
        append name [get_name_info -info node_type -observable_type \
pre_synthesis $name_id]
        puts $name
    }
}
```

For more information about the `get_names` command, refer to The get_names Command on page 38.

2.6.5. The get_names Command

To query a filtered output collection of all matching node name IDs found in a compiled Quartus Prime project, use the `get_names` command.

To access each element of the output collection, use the Tcl command `foreach_in_collection`. For `get_names` or `foreach_in_collection` command example, type `get_names -long_help` or `foreach_in_collection -long_help`.

- If the -node_type option is not specified, the default value is all.
- If the -observable_type option is not specified, the default value is all.
- The node type pin includes input, output, bidir, assigned, unassigned, virtual, and pin.
- The node type qsf include names from the .qsf settings file.
- The node type all includes all node types.
- The node type all_reg includes all node types and registers post-fitting.

The value for -observable_type option can be one of the following:

Table 9. Values for observable_type Option

Observable Type	Description
all	Use post-Fitter information. If it is not available, post-synthesis information is used. Else, pre-synthesis information is used if it exists.
pre_synthesis	Use pre-synthesis information.
post_synthesis	Use post-synthesis information.
post_fitter	Use post-Fitter information.
post_asm	Use post-Assembler information. The post-Assembler information is supported only for designs using the HardCopy II device family.
stp_pre_synthesis	Use Signal Tap pre-synthesis information.

Arguments

Following table lists the get_names command arguments:

Table 10. The get_names Command Arguments

Argument	Description
-h | -help	Displays a short help.
-long help	Displays a long help with examples and possible return values.
-entity<wildcard>	Specifies the entity to get names from hierarchies instantiated by the entity.
-filter<wildcard>	Specifies the node's full path name and wildcard characters.
-node_type <all|comb|reg|pin|input|output|bidir|hierarchy|mem|bus|qsf|state_machine|assigned|unassigned|all_reg|partition|virtual>	Filters based on the specified node type.
-observable_type <all|pre_synthesis|post_synthesis|post_fitter|stp_pre_synthesis>]	Filters based on the specified observable type.

Return Values

Following table lists values returned by the get_names command:

Table 11. The get_names Command Return Values

Code Name	Code	String Returned
TCL_OK	0	INFO: operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Get names cannot return <string> because the name was found in a partition that's not the root partition. Refine your get_names search pattern to exclude child partitions.
TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <string>. At the minimum, run Analysis & Synthesis with the specified revision name before using this Tcl command.
TCL_ERROR	1	ERROR: Illegal node type: <string>. Specify "all", "comb", "reg", "pin", "hierarchy", or "bus".
TCL_ERROR	1	ERROR: Illegal observable type: <string>. Specify "all", "pre_synthesis", "post_synthesis", or "post_fitter".
TCL_ERROR	1	ERROR: You must open a project before you can use this command.

Example Use

```
# Search for a single post-Fitter pin with the name accel and make assignments
set accel_name_id [get_names -filter accel -node_type pin -observable_type
post_fitter]

foreach_in_collection name_id $accel_name_id {
   # Get the full path name of the node
   set target [get_name_info -info full_path $name_id]
   # Set multicycle assignment
   set_multicycle_assignment -to $target 2
   # Set location assignment
   set_location_assignment -to $target Pin_E22
  }
# Search for nodes of any post-Fitter node type with name length <= 5. The
default node type is "all"
set name_ids [get_names -filter ????? -observable_type post_fitter]
foreach_in_collection name_id $name_ids {
   # Print the name id
   puts $name_id
   # Print the node type
   puts [get_name_info -info node_type $name_id]
   # Print the full path (which excludes the current focus entity from the path)
   puts [get_name_info -info full_path $name_id]
  }
# Search for nodes of any post-Fitter node type that end in "eed".
# The default node type is "all"
  set name_ids [get_names -filter *eed -observable_type post_fitter]
  foreach_in_collection name_id $name_ids {
   # Print the name id
   puts $name_id
   # Print the node type
   puts [get_name_info -info node_type $name_id]
   # Print the full path (which excludes the current
   # focus entity from the path)
   puts [get_name_info -info full_path $name_id]
  }
```

2.6.6. The post_message Command

To print messages that are formatted like Quartus Prime software messages, use the `post_message` command. Messages printed by the `post_message` command appear in the **System** tab of the **Messages** window in the Quartus Prime GUI, and are written to standard output when scripts are run. Arguments for the `post_message` command include an optional message type and a required message string.

The message type can be one of the following:

- `info` (default)
- `extra_info`
- `warning`
- `critical_warning`
- `error`

If you do not specify a type, Quartus Prime software defaults to `info`.

With the Quartus Prime software in Windows, you can color code messages displayed at the system command prompt with the `post_message` command. Add the following line to your `quartus2.ini` file:

```
DISPLAY_COMMAND_LINE_MESSAGES_IN_COLOR = on
```

The following example shows how to use the `post_message` command.

```
post_message -type warning "Design has gated clocks"
```

2.6.7. Accessing Command-Line Arguments

The global variable `quartus(args)` is a list of the arguments typed on the command-line following the name of the Tcl script.

Example 1. Simple Command-Line Argument Access

The following Tcl example prints all the arguments in the `quartus(args)` variable:

```
set i 0
foreach arg $quartus(args) {
    puts "The value at index $i is $arg"
    incr i
}
```

Example 2. Passing Command-Line Arguments to Scripts

If you copy the script in the previous example to a file named `print_args.tcl`, it displays the following output when you type the following at a command prompt.

```
quartus_sh -t print_args.tcl my_project 100MHz
The value at index 0 is my_project
The value at index 1 is 100MHz
```

2.6.7.1. The cmdline Package

You can use the `cmdline` package included with the Quartus Prime software for more robust and self-documenting command-line argument passing. The `cmdline` package supports command-line arguments with the form `-<option><value>`.

cmdline Package

```
package require cmdline
variable ::argv0 $::quartus(args)
set options {
   { "project.arg" "" "Project name" }
   { "frequency.arg" "" "Frequency" }
}
set usage "You need to specify options and values"
array set optshash [::cmdline::getoptions ::argv $options $usage]
puts "The project name is $optshash(project)"
puts "The frequency is $optshash(frequency)"
```

If you save those commands in a Tcl script called `print_cmd_args.tcl` you see the following output when you type the following command at a command prompt.

Passing Command-Line Arguments for Scripts

```
quartus_sh -t print_cmd_args.tcl -project my_project -frequency 100MHz
The project name is my_project
The frequency is 100MHz
```

Virtually all Quartus Prime Tcl scripts must open a project. You can open a project, and you can optionally specify a revision name with code like the following example. The example checks whether the specified project exists. If it does, the example opens the current revision, or the revision you specify.

Full-Featured Method to Open Projects

```
package require cmdline
variable ::argv0 $::quartus(args)
set options { \
{ "project.arg" "" "Project Name" } \
{ "revision.arg" "" "Revision Name" } \
}
array set optshash [::cmdline::getoptions ::argv0 $options]
# Ensure the project exists before trying to open it
if {[project_exists $optshash(project)]} {
    if {[string equal "" $optshash(revision)]} {
        # There is no revision name specified, so default
        # to the current revision
        project_open $optshash(project) -current_revision
    } else {
        # There is a revision name specified, so open the
        # project with that revision
        project_open $optshash(project) -revision \
            $optshash(revision)
    }
} else {
    puts "Project $optshash(project) does not exist"
    exit 1
}
# The rest of your script goes here
```

If you do not require this flexibility or error checking, you can use just the `project_open` command.

Simple Method to Open Projects

```
set proj_name [lindex $argv 0]
project_open $proj_name
```

2.6.8. The quartus() Array

The global `quartus()` Tcl array includes other information about your project and the current Quartus Prime executable that might be useful to your scripts. The scripts in the preceding examples parsed command line arguments found in `quartus(args)`. For information on the other elements of the `quartus()` array, type the following command at a Tcl prompt:

```
help -quartus
```

2.7. The Quartus Prime Tcl Shell in Interactive Mode Example

This section presents how to make project assignments and then compile the finite impulse response (FIR) filter tutorial project with the `quartus_sh` interactive shell.

This example assumes you already have the `fir_filter` tutorial design files in a project directory.

1. To run the interactive Tcl shell, type the following at the system command prompt:

   ```
   quartus_sh -s
   ```

2. Create a new project called `fir_filter`, with a revision called `filtref` by typing:

   ```
   project_new -revision filtref fir_filter
   ```

   Note:
   - If the project file and project name are the same, the Quartus Prime software gives the revision the same name as the project.
   - If a `.qpf` file for this project already exists, the Quartus Prime software displays an error stating that the project already exists.

   Because the revision named `filtref` matches the top-level file, all design files are automatically picked up from the hierarchy tree.

3. Set a global assignment for the device:

   ```
   set_global_assignment -name family <device family name>
   ```

   To learn more about assignment names that you can use with the `-name` option, refer to Quartus Prime Help.

   Note: For assignment values that contain spaces, enclose the value in quotation marks.

4. To compile a design, use the `::quartus::flow` package, which properly exports the new project assignments and compiles the design with the proper sequence of the command-line executables. First, load the package:

   ```
   load_package flow
   ```

   It returns:

   ```
   1.1
   ```

5. To perform a full compilation of the FIR filter design, use the `execute_flow` command with the `-compile` option:

```
execute_flow -compile
```

   This command compiles the FIR filter tutorial project, exporting the project assignments and running `quartus_syn`, `quartus_fit`, `quartus_asm`, and `quartus_sta`. This sequence of events is the same as selecting **Processing ➤ Start Compilation** in the Quartus Prime GUI.
6. When you are finished with a project, close it with the `project_close` command.
7. To exit the interactive Tcl shell, type `exit` at a Tcl prompt.

2.8. The tclsh Shell

On the UNIX and Linux operating systems, the tclsh shell included with the Quartus Prime software is initialized with a minimal PATH environment variable. As a result, system commands might not be available within the tclsh shell because certain directories are not in the PATH environment variable.

To include other directories in the path searched by the tclsh shell, set the `QUARTUS_INIT_PATH` environment variable before running the tclsh shell. Directories in the `QUARTUS_INIT_PATH` environment variable are searched by the tclsh shell when you execute a system command.

2.9. Tcl Scripting Basic Examples

The core Tcl commands support variables, control structures, and procedures. Additionally, there are commands for accessing the file system and network sockets, and running other programs. You can create platform-independent graphical interfaces with the Tk widget set.

Tcl commands are executed immediately as they are typed in an interactive Tcl shell. You can also create scripts (including the examples in this chapter) in files and run them with the Quartus Prime executables or with the tclsh shell.

2.9.1. Hello World Example

The following shows the basic "Hello world" example in Tcl:

```
puts "Hello world"
```

Use double quotation marks to group the words `hello` and `world` as one argument. Double quotation marks allow substitutions to occur in the group. Substitutions can be simple variable substitutions, or the result of running a nested command. Use curly braces {} for grouping when you want to prevent substitutions.

2.9.2. Variables

Assign a value to a variable with the `set` command. You do not have to declare a variable before using it. Tcl variable names are case-sensitive.

```
set a 1
```

To access the contents of a variable, use a dollar sign (“$”) before the variable name. The following example prints `"Hello world"` in a different way.

```
set a Hello
set b world
puts "$a $b"
```

2.9.3. Substitutions

Tcl performs three types of substitution:

- Variable value substitution
- Nested command substitution
- Backslash substitution

2.9.3.1. Variable Value Substitution

Variable value substitution, refers to accessing the value stored in a variable with a dollar sign (“$”) before the variable name.

2.9.3.2. Nested Command Substitution

Nested command substitution refers to how the Tcl interpreter evaluates Tcl code in square brackets. The Tcl interpreter evaluates nested commands, starting with the innermost nested command, and commands nested at the same level from left to right. Each nested command result is substituted in the outer command.

```
set a [string length foo]
```

2.9.3.3. Backslash Substitution

Backslash substitution allows you to quote reserved characters in Tcl, such as dollar signs (“$”) and braces (“[ ]”). You can also specify other special ASCII characters like tabs and new lines with backslash substitutions. A backslash before a character tells the TCL interpreter to treat the next character as a literal if the character is not the last character on the line.

```
puts "This is a \$ special character"

puts "This is a\
$ special character and line continuation"

puts "This is backslash \is ignored"

puts "This is backslash\
    continued on next line"
```

2.9.4. Arithmetic

Use the `expr` command to perform arithmetic calculations. Use curly braces ("`{ }`") to group the arguments of this command for greater efficiency and numeric precision.

```
set a 5
set b [expr { $a + sqrt(2) }]
```

The Quartus Prime software supports all standard Tcl boolean and arithmetic operators, such as && (AND), || (OR), ! (NOT), and comparison operators such as < (less than), > (greater than), and == (equal to).

2.9.5. Lists

A Tcl list is a series of values. Supported list operations include creating lists, appending lists, extracting list elements, computing the length of a list, sorting a list, and more.

```
set a { 1 2 3 }
```

You can use the `lindex` command to extract information at a specific index in a list. Indexes are zero-based. You can use the index `end` to specify the last element in the list, or the index `end-`<*n*> to count from the end of the list. For example, to print the second element (at index 1) in the list stored in `a` use the following code.

```
puts [lindex $a 1]
```

The `llength` command returns the length of a list.

```
puts [llength $a]
```

The `lappend` command appends elements to a list. If a list does not already exist, the list you specify is created. The list variable name is not specified with a dollar sign ("$").

```
lappend a 4 5 6
```

2.9.6. Arrays

Arrays are similar to lists except that they use a string-based index. Tcl arrays are implemented as hash tables. You can create arrays by setting each element individually or with the `array set` command.

To set an element with an index of `Mon` to a value of `Monday` in an array called `days`, use the following command:

```
set days(Mon) Monday
```

The `array set` command requires a list of index/value pairs. This example sets the array called days:

```
array set days { Sun Sunday Mon Monday Tue Tuesday\
                 Wed Wednesday Thu Thursday Fri Friday Sat Saturday }
```

```
set day_abbreviation Mon
puts $days($day_abbreviation)
```

Use the `array names` command to get a list of all the indexes in a particular array. The index values are not returned in any specified order. The following example is one way to iterate over all the values in an array.

```
foreach day [array names days] {
    puts "The abbreviation $day corresponds to the day name $days($day)"
}
```

Arrays are a very flexible way of storing information in a Tcl script and are a good way to build complex data structures.

2.9.7. Control Structures

Tcl supports common control structures, including if-then-else conditions and `for`, `foreach`, and `while` loops. The position of the curly braces as shown in the following examples ensures the control structure commands are executed efficiently and correctly. The following example prints whether the value of variable a positive, negative, or zero.

If-Then-Else Structure

```
if { $a > 0 } { puts "The value is positive"
} elseif { $a < 0 } {
    puts "The value is negative"
} else {
    puts "The value is zero"
}
```

The following example uses a `for` loop to print each element in a list.

For Loop

```
set a { 1 2 3 }
for { set i 0 } { $i < [llength $a] } { incr i } {
    puts "The list element at index $i is [lindex $a $i]"
}
```

The following example uses a `foreach` loop to print each element in a list.

foreach Loop

```
set a { 1 2 3 }
foreach element $a {
    puts "The list element is $element"
}
```

The following example uses a `while` loop to print each element in a list.

while Loop

```
set a { 1 2 3 }
set i 0
while { $i < [llength $a] } { puts "The list element at index $i is [lindex $a
$i]"
    incr i
}
```

You do not have to use the `expr` command in boolean expressions in control structure commands because they invoke the `expr` command automatically.

2.9.8. Procedures

Use the `proc` command to define a Tcl procedure (known as a subroutine or function in other scripting and programming languages). The scope of variables in a procedure is local to the procedure. If the procedure returns a value, use the `return` command to return the value from the procedure. The following example defines a procedure that multiplies two numbers and returns the result.

Simple Procedure

```
proc multiply { x y } {
    set product [expr { $x * $y }]
    return $product
}
```

The following example shows how to use the `multiply` procedure in your code. You must define a procedure before your script calls it.

Using a Procedure

```
proc multiply { x y } {
    set product [expr { $x * $y }]
    return $product
}
set a 1
set b 2
puts [multiply $a $b]
```

Define procedures near the beginning of a script. If you want to access global variables in a procedure, use the `global` command in each procedure that uses a global variable.

Accessing Global Variables

```
proc print_global_list_element { i } {
    global my_data
    puts "The list element at index $i is [lindex $my_data $i]"
}
set my_data { 1 2 3}
print_global_list_element 0
```

2.9.9. File I/O

Tcl includes commands to read from and write to files. You must open a file before you can read from or write to it, and close it when the read and write operations are done.

To open a file, use the `open` command; to close a file, use the `close` command. When you open a file, specify its name and the mode in which to open it. If you do not specify a mode, Tcl defaults to read mode. To write to a file, specify `w` for write mode.

Open a File for Writing

```
set output [open myfile.txt w]
```

Tcl supports other modes, including appending to existing files and reading from and writing to the same file.

The `open` command returns a file handle to use for read or write access. You can use the `puts` command to write to a file by specifying a file handle.

Write to a File

```
set output [open myfile.txt w]
puts $output "This text is written to the file."
close $output
```

You can read a file one line at a time with the `gets` command. The following example uses the `gets` command to read each line of the file and then prints it out with its line number.

Read from a File

```
set input [open myfile.txt]
set line_num 1
while { [gets $input line] >= 0 } {
    # Process the line of text here
    puts "$line_num: $line"
    incr line_num
}
close $input
```

2.9.10. Syntax and Comments

Arguments to Tcl commands are separated by white space, and Tcl commands are terminated by a newline character or a semicolon. You must use backslashes when a Tcl command extends more than one line. The backslash (\) must be the last character in the line to designate line extension. If the backslash is followed by any other character including a space, that character is treated as a literal.

Tcl uses the hash or pound character (#) to begin comments. The # character must begin a comment. If you prefer to include comments on the same line as a command, be sure to terminate the command with a semicolon before the # character. The following example is a valid line of code that includes a `set` command and a comment.

```
set a 1;# Initializes a
```

Without the semicolon, the command is invalid because the `set` command does not terminate until the new line after the comment.

The Tcl interpreter counts curly braces inside comments, which can lead to errors that are difficult to track down. The following example causes an error because of unbalanced curly braces.

```
# if { $x > 0 } {
if { $y > 0 } {
    # code here
}
```

2.9.11. External References

For more information about Tcl, refer to the following sources:

- Brent B. Welch and Ken Jones, and Jeffery Hobbs, *Practical Programming in Tcl and Tk* (Upper Saddle River: Prentice Hall, 2003)
- John Ousterhout and Ken Jones, *Tcl and the Tk Toolkit* (Boston: Addison-Wesley Professional, 2009)
- Mark Harrison and Michael McLennan, *Effective Tcl/Tk Programming: Writing Better Programs in Tcl and Tk* (Boston: Addison-Wesley Professional, 1997)

Related Information

www.tcl.tk
Tcl Developer Xchange

2.10. Tcl Scripting Revision History

The following revision history applies to this chapter:

Table 12. Document Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied phase I Altera rebranding throughout.
2023.10.02	23.3	• Updated *The Tcl Console Window* topic for Tcl command echo and log file save.
2022.04.03	23.1	• Updated name of Intel Agilex 7 device family.
2020.12.14	20.4	• Revised "Tcl Scripting" topic to include link to new "Tcl Commands and Packages" reference. • Revised "Tcl Packages" topic for latest supported packages.
2019.06.28	19.1	Minor correction in *The get_names Command*
2019.04.01	19.1	• Added Node Finder Tcl commands. • Added the `get_names` command. • Rectified code snippet formatting in *Compile All Revisions*, *Arrays*, and *Control Structures* topics.
2018.05.07	18.0.0	• Removed deprecated options. • External reference links updated. • Corrected typos and made minor content fixes.
2016.10.31	16.1.0	• Implemented Intel rebranding.

continued...

Document Version	Quartus Prime Version	Changes
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Updated the list of Tcl packages in the *Quartus Prime Tcl Packages* section. • Updated the *Quartus Prime Tcl API Help* section: — Updated the Tcl Help Output
June 2014	14.0.0	Updated the format.
June 2012	12.0.0	• Removed survey link.
November 2011	11.0.1	• Template update • Updated supported version of Tcl in the section "Tool Command Language." • minor editorial changes
May 2011	11.0.0	Minor updates throughout document.
December 2010	10.1.0	Template update Updated to remove tcl packages used by the Classic Timing Analyzer
July 2010	10.0.0	Minor updates throughout document.
November 2009	9.1.0	• Removed Logic Lock example. • Added the incremental_compilation, insystem_source_probe, and rtl packages to Table 3-1 and Table 3-2. • Added quartus_map to table 3-2.
March 2009	9.0.0	• Removed the "EDA Tool Assignments" section • Added the section "Compile All Revisions" on page 3–9 • Added the section "Using the tclsh Shell" on page 3–20
November 2008	8.1.0	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0.0	Updated references.

3. TCL Commands and Packages

3.1. TCL Commands and Packages Summary

Tcl Command	Tcl Package	Package Version
get_back_annotation_assignments	backannotate	1.1
logiclock_back_annotate	backannotate	1.1
activate_link	board	1.0
check_online_design_validity	board	1.0
deploy_par_file	board	1.0
download_par_file	board	1.0
get_board_design_path	board	1.0
get_board_devkits	board	1.0
get_board_families	board	1.0
get_board_info	board	1.0
get_board_vendors	board	1.0
get_design_description	board	1.0
get_design_development_kits	board	1.0
get_design_documents_info	board	1.0
get_design_download_link	board	1.0
get_design_families	board	1.0
get_design_info	board	1.0
get_design_quartus_versions	board	1.0
get_design_rich_description	board	1.0
get_ui_file	board	1.0
launch_qsys	board	1.0
load_design_info	board	1.0
reset_board_info	board	1.0
apply_assignments	bpps	1.0
check_plan	bpps	1.0
export_constraints_to_qsf	bpps	1.0

continued...

Tcl Command	Tcl Package	Package Version
get_cell_info	bpps	1.0
get_device	bpps	1.0
get_hdbpath_from_id	bpps	1.0
get_id_from_hdbpath	bpps	1.0
get_location_info	bpps	1.0
get_placement	bpps	1.0
get_placement_info	bpps	1.0
get_placements	bpps	1.0
get_placements_of_group	bpps	1.0
harden_cell	bpps	1.0
harden_cells	bpps	1.0
initialize	bpps	1.0
load_floorplan	bpps	1.0
place_cells	bpps	1.0
read_tpl_placement	bpps	1.0
remove_invalid_reports	bpps	1.0
report_all	bpps	1.0
report_cell_connectivity	bpps	1.0
report_cell_placement_reasons	bpps	1.0
report_cells	bpps	1.0
report_clocks	bpps	1.0
report_legal_cell_locations	bpps	1.0
report_location_types	bpps	1.0
report_locations	bpps	1.0
report_regions	bpps	1.0
report_summary	bpps	1.0
reset_plan	bpps	1.0
save_floorplan	bpps	1.0
save_pin_assignments	bpps	1.0
select_dr_ips	bpps	1.0
set_mode	bpps	1.0
shutdown	bpps	1.0
soften_cell	bpps	1.0
soften_cells	bpps	1.0
undo_last_placement	bpps	1.0
unplace_cells	bpps	1.0

continued...

Tcl Command	Tcl Package	Package Version
update_pdpw	bpps	1.0
validate_placement	bpps	1.0
write_plan	bpps	1.0
write_tpl_placement	bpps	1.0
check_node	chip_planner	2.0
close_chip_planner	chip_planner	2.0
design_has_ace_support	chip_planner	2.0
design_has_encrypted_ip	chip_planner	2.0
get_info_parameters	chip_planner	2.0
get_iports	chip_planner	2.0
get_node_by_name	chip_planner	2.0
get_node_info	chip_planner	2.0
get_nodes	chip_planner	2.0
get_oports	chip_planner	2.0
get_port_by_type	chip_planner	2.0
get_port_info	chip_planner	2.0
get_sp_pin_list	chip_planner	2.0
get_tile_power_setting	chip_planner	2.0
read_netlist	chip_planner	2.0
set_batch_mode	chip_planner	2.0
add_to_collection	dcmd_dni	1.0
all_clocks	dcmd_dni	1.0
all_fanin	dcmd_dni	1.0
all_fanout	dcmd_dni	1.0
all_inputs	dcmd_dni	1.0
all_outputs	dcmd_dni	1.0
all_registers	dcmd_dni	1.0
append_to_collection	dcmd_dni	1.0
color	dcmd_dni	1.0
copy_collection	dcmd_dni	1.0
create_clock	dcmd_dni	1.0
current_design	dcmd_dni	1.0
current_instance	dcmd_dni	1.0
delete_stale_sandboxes	dcmd_dni	1.0
filter_collection	dcmd_dni	1.0
get_cells	dcmd_dni	1.0

continued...

Tcl Command	Tcl Package	Package Version
get_clocks	dcmd_dni	1.0
get_designs	dcmd_dni	1.0
get_generated_clocks	dcmd_dni	1.0
get_nets	dcmd_dni	1.0
get_pins	dcmd_dni	1.0
get_ports	dcmd_dni	1.0
get_property	dcmd_dni	1.0
highlight	dcmd_dni	1.0
index_collection	dcmd_dni	1.0
is_dni_mode	dcmd_dni	1.0
is_dni_mode_for_developer_testing	dcmd_dni	1.0
list_properties	dcmd_dni	1.0
load_design	dcmd_dni	1.0
read_sdc	dcmd_dni	1.0
remove_from_collection	dcmd_dni	1.0
selection	dcmd_dni	1.0
set_property	dcmd_dni	1.0
set_time_format	dcmd_dni	1.0
set_time_unit	dcmd_dni	1.0
sizeof_collection	dcmd_dni	1.0
sort_collection	dcmd_dni	1.0
unload_design	dcmd_dni	1.0
write_sdc	dcmd_dni	1.0
commit_design	design	1.0
convert_partition	design	1.0
create_assignment	design	1.0
delete_assignments	design	1.0
disable_assignments	design	1.0
enable_assignments	design	1.0
export_design	design	1.0
export_partition	design	1.0
extract_metadata	design	1.0
get_assignment_info	design	1.0
get_assignment_names	design	1.0
get_assignments	design	1.0
get_entity_names	design	1.0

continued...

Tcl Command	Tcl Package	Package Version
get_instances	design	1.0
import_design	design	1.0
import_partition	design	1.0
list_valid_snapshot_names	design	1.0
load_design	design	1.0
report_assignments	design	1.0
set_assignment_info	design	1.0
get_family_list	device	1.0
get_part_info	device	1.0
get_part_list	device	1.0
report_device_info	device	1.0
report_family_info	device	1.0
report_part_info	device	1.0
create_generated_clock	dni_sdc	1.5
remove_clock_groups	dni_sdc	1.5
remove_clock_latency	dni_sdc	1.5
remove_clock_uncertainty	dni_sdc	1.5
remove_disable_timing	dni_sdc	1.5
remove_input_delay	dni_sdc	1.5
remove_output_delay	dni_sdc	1.5
set_clock_groups	dni_sdc	1.5
set_clock_latency	dni_sdc	1.5
set_clock_uncertainty	dni_sdc	1.5
set_data_delay	dni_sdc	1.5
set_disable_timing	dni_sdc	1.5
set_false_path	dni_sdc	1.5
set_input_delay	dni_sdc	1.5
set_input_transition	dni_sdc	1.5
set_max_delay	dni_sdc	1.5
set_max_skew	dni_sdc	1.5
set_max_time_borrow	dni_sdc	1.5
set_min_delay	dni_sdc	1.5
set_multicycle_path	dni_sdc	1.5
set_net_delay	dni_sdc	1.5
set_operating_conditions	dni_sdc	1.5
set_output_delay	dni_sdc	1.5

continued...

Tcl Command	Tcl Package	Package Version
set_sense	dni_sdc	1.5
set_timing_derate	dni_sdc	1.5
add_check_op	drc	1.0
add_check_parameter	drc	1.0
add_object	drc	1.0
add_object_with_properties	drc	1.0
add_property	drc	1.0
add_rule	drc	1.0
add_rule_violation	drc	1.0
add_violation_record	drc	1.0
add_waiver	drc	1.0
check_design	drc	1.0
delete_waivers	drc	1.0
get_objects	drc	1.0
get_option	drc	1.0
get_property	drc	1.0
get_stage_info	drc	1.0
get_waivers	drc	1.0
list_properties	drc	1.0
report_waivers	drc	1.0
set_option	drc	1.0
set_property	drc	1.0
should_run_drc	drc	1.0
update_check_op	drc	1.0
update_rule	drc	1.0
adjust_pll_refclk	eco	1.0
create_new_node	eco	1.0
create_wirelut	eco	1.0
report_partitions	eco	1.0
eco_reroute	eco	1.0
eco_unload_design	eco	1.0
fitter_report_timing	eco	1.0
fitter_timing_summary	eco	1.0
get_available_snapshots	eco	1.0
get_eco_checkpoint	eco	1.0
get_loaded_snapshot	eco	1.0

continued...

Tcl Command	Tcl Package	Package Version
get_lutmask_equation	eco	1.0
get_node_location	eco	1.0
make_connection	eco	1.0
modify_io_current_strength	eco	1.0
modify_io_delay_chain	eco	1.0
modify_io_slew_rate	eco	1.0
modify_lutmask	eco	1.0
place_node	eco	1.0
remove_connection	eco	1.0
remove_node	eco	1.0
report_connections	eco	1.0
report_legal_locations	eco	1.0
report_nodes_at_location	eco	1.0
report_ports	eco	1.0
report_routing	eco	1.0
report_unplaced_nodes	eco	1.0
restore_eco_checkpoint	eco	1.0
unplace_node	eco	1.0
update_mif_files	eco	1.0
apply_setting	external_memif_toolkit	1.0
calibrate_termination	external_memif_toolkit	1.0
configure_driver	external_memif_toolkit	1.0
create_connection_report	external_memif_toolkit	1.0
create_toolkit_report	external_memif_toolkit	1.0
driver_margining	external_memif_toolkit	1.0
establish_connection	external_memif_toolkit	1.0
generate_eye_diagram	external_memif_toolkit	1.0
get_connection_commands	external_memif_toolkit	1.0
get_connection_info	external_memif_toolkit	1.0
get_connection_interfaces	external_memif_toolkit	1.0
get_connection_report_info	external_memif_toolkit	1.0
get_connection_report_types	external_memif_toolkit	1.0
get_connection_types	external_memif_toolkit	1.0
get_connections	external_memif_toolkit	1.0
get_device_names	external_memif_toolkit	1.0
get_hardware_names	external_memif_toolkit	1.0

continued...

Tcl Command	Tcl Package	Package Version
get_setting_types	external_memif_toolkit	1.0
get_toolkit_report_types	external_memif_toolkit	1.0
initialize_connections	external_memif_toolkit	1.0
link_project_to_device	external_memif_toolkit	1.0
read_setting	external_memif_toolkit	1.0
reindex_connections	external_memif_toolkit	1.0
reset_tg2	external_memif_toolkit	1.0
run_connection_command	external_memif_toolkit	1.0
set_active_interface	external_memif_toolkit	1.0
set_stress_pattern	external_memif_toolkit	1.0
terminate_connection	external_memif_toolkit	1.0
terminate_connections	external_memif_toolkit	1.0
unlink_project_from_device	external_memif_toolkit	1.0
write_connection_target_report	external_memif_toolkit	1.0
check	fif	1.0
dump	fif	1.0
dump_cram_frame	fif	1.0
dump_mem	fif	1.0
dump_pr_bitstream	fif	1.0
generate	fif	1.0
get_frame_count	fif	1.0
get_frame_size	fif	1.0
get_sector_information_sdm_based_fpga	fif	1.0
get_sensitive_location	fif	1.0
get_sensitive_location_sdm_based_fpga	fif	1.0
setup	fif	1.0
setup_sdm_based_fpga	fif	1.0
terminate	fif	1.0
add_object	flng	1.0
add_property	flng	1.0
bind_flow	flng	1.0
delete_object	flng	1.0
get_default_flow_run_name	flng	1.0
get_flow_list	flng	1.0
get_next_available_id	flng	1.0

continued...

Tcl Command	Tcl Package	Package Version
get_object	flng	1.0
get_objects	flng	1.0
get_option	flng	1.0
get_property	flng	1.0
get_task_command	flng	1.0
get_task_status_property	flng	1.0
init_repository	flng	1.0
list_properties	flng	1.0
monitor_flow	flng	1.0
run_flow	flng	1.0
run_flow_command	flng	1.0
set_option	flng	1.0
set_property	flng	1.0
write_task_assignment_digest	flng	1.0
write_task_checkpoint_written	flng	1.0
write_task_finished	flng	1.0
write_task_started	flng	1.0
execute_flow	flow	1.1
execute_module	flow	1.1
get_flow_templates	flow	1.1
get_status_db_property	flow	1.1
write_flow_assignment_digest	flow	1.1
write_flow_finished	flow	1.1
write_flow_started	flow	1.1
write_flow_template	flow	1.1
begin_memory_edit	insystem_memory_edit	1.0
end_memory_edit	insystem_memory_edit	1.0
get_editable_mem_instances	insystem_memory_edit	1.0
read_content_from_memory	insystem_memory_edit	1.0
save_content_from_memory_to_file	insystem_memory_edit	1.0
update_content_to_memory_from_file	insystem_memory_edit	1.0
write_content_to_memory	insystem_memory_edit	1.0
end_insystem_source_probe	insystem_source_probe	1.0
get_insystem_source_probe_instance_info	insystem_source_probe	1.0
read_probe_data	insystem_source_probe	1.0

continued...

Tcl Command	Tcl Package	Package Version
read_source_data	insystem_source_probe	1.0
start_insystem_source_probe	insystem_source_probe	1.0
write_source_data	insystem_source_probe	1.0
analyze_files	interactive_synthesis	1.0
check_rtl_connections	interactive_synthesis	1.0
dissolve_rtl_partition	interactive_synthesis	1.0
dynamic_report	interactive_synthesis	1.0
elaborate	interactive_synthesis	1.0
get_entities	interactive_synthesis	1.0
get_rtl_partition_name	interactive_synthesis	1.0
get_rtl_partitions	interactive_synthesis	1.0
init_synthesis_constraints_propagation_reporter	interactive_synthesis	1.0
link_rtl_design	interactive_synthesis	1.0
print_ipxact	interactive_synthesis	1.0
report_rtl_assignments	interactive_synthesis	1.0
report_rtl_parameters	interactive_synthesis	1.0
report_rtl_stats	interactive_synthesis	1.0
reset_rtl_design	interactive_synthesis	1.0
sasic	interactive_synthesis	1.0
save_rtl_design	interactive_synthesis	1.0
set_sasic_handoff_flow	interactive_synthesis	1.0
synthesize	interactive_synthesis	1.0
uniquify	interactive_synthesis	1.0
write_rtl_report	interactive_synthesis	1.0
get_device_speed	ipdrc	1.0
get_ip_hpaths	ipdrc	1.0
get_ip_name	ipdrc	1.0
get_ip_pma_modulation	ipdrc	1.0
get_ip_speed	ipdrc	1.0
get_ip_type	ipdrc	1.0
get_ip_xcvr_type	ipdrc	1.0
set_ip_info	ipdrc	1.0
clear_ip_generation_dirs	ipgen	1.0
generate_ip_file	ipgen	1.0
generate_project_ip_files	ipgen	1.0

continued...

Tcl Command	Tcl Package	Package Version
get_project_ip_files	ipgen	1.0
compute_pll	iptclgen	1.0
generate_vhdl_simgen_model	iptclgen	1.0
parse_hdl	iptclgen	1.0
parse_tcl	iptclgen	1.0
close_device	jtag	1.0
device_dr_shift	jtag	1.0
device_ir_shift	jtag	1.0
device_lock	jtag	1.0
device_run_test_idle	jtag	1.0
device_unlock	jtag	1.0
device_virtual_dr_shift	jtag	1.0
device_virtual_ir_shift	jtag	1.0
get_device_names	jtag	1.0
get_hardware_names	jtag	1.0
open_device	jtag	1.0
begin_logic_analyzer_interface_control	logic_analyzer_interface	1.0
change_bank_to_output_pin	logic_analyzer_interface	1.0
end_logic_analyzer_interface_control	logic_analyzer_interface	1.0
get_current_state_of_output_pin	logic_analyzer_interface	1.0
tristate_output_pin	logic_analyzer_interface	1.0
checksum	misc	1.0
disable_natural_bus_naming	misc	1.0
enable_natural_bus_naming	misc	1.0
escape_brackets	misc	1.0
foreach_in_collection	misc	1.0
get_collection_size	misc	1.0
get_environment_info	misc	1.0
get_message_count	misc	1.0
init_tk	misc	1.0
load	misc	1.0
load_package	misc	1.0
post_message	misc	1.0
qerror	misc	1.0
qexec	misc	1.0
qexit	misc	1.0

continued...

Tcl Command	Tcl Package	Package Version
record_tcl_cmd	misc	1.0
stopwatch	misc	1.0
get_assignment	names	1.0
set_assignment	names	1.0
initialize	periph	1.0
shutdown	periph	1.0
check_plan	periph	1.0
get_cell_info	periph	1.0
get_cells	periph	1.0
get_location_info	periph	1.0
get_placement_info	periph	1.0
get_placements	periph	1.0
load_floorplan	periph	1.0
place_cells	periph	1.0
remove_invalid_reports	periph	1.0
report_all	periph	1.0
report_cell_connectivity	periph	1.0
report_cell_placement_reasons	periph	1.0
report_cells	periph	1.0
report_clocks	periph	1.0
report_legal_cell_locations	periph	1.0
report_location_types	periph	1.0
report_locations	periph	1.0
report_noc_performance	periph	1.0
report_regions	periph	1.0
report_summary	periph	1.0
reset_plan	periph	1.0
save_floorplan	periph	1.0
set_clock_type	periph	1.0
undo_last_placement	periph	1.0
unplace_cells	periph	1.0
update_pdpw	periph	1.0
update_plan	periph	1.0
write_plan	periph	1.0
test	pfg	1.0
create_revision	proj_asgn	1.0

continued...

Tcl Command	Tcl Package	Package Version
generate_project_tcl	proj_asgn	1.0
get_name_info	proj_asgn	1.0
get_names	proj_asgn	1.0
get_top_level_entity	proj_asgn	1.0
is_fitter_in_qhd_mode	proj_asgn	1.0
close_side_revision	project	7.0
create_revision	project	7.0
delete_revision	project	7.0
execute_assignment_batch	project	7.0
export_assignments	project	7.0
generate_project_tcl	project	7.0
get_all_assignment_names	project	7.0
get_all_assignments	project	7.0
get_all_global_assignments	project	7.0
get_all_instance_assignments	project	7.0
get_all_parameters	project	7.0
get_all_quartus_defaults	project	7.0
get_all_user_option_names	project	7.0
get_assignment_info	project	7.0
get_assignment_name_info	project	7.0
get_current_project	project	7.0
get_current_revision	project	7.0
get_database_version	project	7.0
get_global_assignment	project	7.0
get_instance_assignment	project	7.0
get_location_assignment	project	7.0
get_name_info	project	7.0
get_names	project	7.0
get_parameter	project	7.0
get_project_directory	project	7.0
get_project_revisions	project	7.0
get_revision_description	project	7.0
get_top_level_entity	project	7.0
get_user_option	project	7.0
is_database_version_compatible	project	7.0
is_fitter_in_qhd_mode	project	7.0

continued...

Tcl Command	Tcl Package	Package Version
is_project_open	project	7.0
open_side_revision	project	7.0
project_archive	project	7.0
project_clean	project	7.0
project_close	project	7.0
project_exists	project	7.0
project_new	project	7.0
project_open	project	7.0
project_restore	project	7.0
remove_all_global_assignments	project	7.0
remove_all_instance_assignments	project	7.0
remove_all_parameters	project	7.0
resolve_file_path	project	7.0
revision_exists	project	7.0
set_current_revision	project	7.0
set_global_assignment	project	7.0
set_high_effort_fmax_optimization_assignments	project	7.0
set_instance_assignment	project	7.0
set_io_assignment	project	7.0
set_location_assignment	project	7.0
set_parameter	project	7.0
set_power_file_assignment	project	7.0
set_revision_description	project	7.0
set_user_option	project	7.0
test_assignment_trait	project	7.0
close_project	project2	1.0
open_project	project2	1.0
assignment_group	project_ui	2.0
delete_revision	project_ui	2.0
execute_assignment_batch	project_ui	2.0
export_assignments	project_ui	2.0
get_all_assignment_names	project_ui	2.0
get_all_assignments	project_ui	2.0
get_all_global_assignments	project_ui	2.0
get_all_instance_assignments	project_ui	2.0

continued...

Tcl Command	Tcl Package	Package Version
get_all_parameters	project_ui	2.0
get_all_quartus_defaults	project_ui	2.0
get_all_user_option_names	project_ui	2.0
get_assignment_info	project_ui	2.0
get_assignment_name_info	project_ui	2.0
get_current_project	project_ui	2.0
get_current_revision	project_ui	2.0
get_global_assignment	project_ui	2.0
get_instance_assignment	project_ui	2.0
get_location_assignment	project_ui	2.0
get_parameter	project_ui	2.0
get_project_directory	project_ui	2.0
get_project_revisions	project_ui	2.0
get_user_option	project_ui	2.0
is_project_open	project_ui	2.0
project_archive	project_ui	2.0
project_close	project_ui	2.0
project_exists	project_ui	2.0
project_new	project_ui	2.0
project_open	project_ui	2.0
project_restore	project_ui	2.0
remove_all_global_assignments	project_ui	2.0
remove_all_instance_assignments	project_ui	2.0
remove_all_parameters	project_ui	2.0
resolve_file_path	project_ui	2.0
revision_exists	project_ui	2.0
set_current_revision	project_ui	2.0
set_global_assignment	project_ui	2.0
set_instance_assignment	project_ui	2.0
set_io_assignment	project_ui	2.0
set_location_assignment	project_ui	2.0
set_parameter	project_ui	2.0
set_power_file_assignment	project_ui	2.0
set_user_option	project_ui	2.0
test_assignment_trait	project_ui	2.0
add_projects_from_archive	qed	1.0

continued...

Tcl Command	Tcl Package	Package Version
check_properties	qed	1.0
check_properties_of_projects	qed	1.0
compile	qed	1.0
configure_local_compute_spec	qed	1.0
configure_lsf_compute_spec	qed	1.0
configure_pbspro_compute_spec	qed	1.0
configure_slurm_compute_spec	qed	1.0
configure_ssh_compute_spec	qed	1.0
create_object	qed	1.0
delete_object	qed	1.0
delete_object_report_panel	qed	1.0
disconnect	qed	1.0
find_projects_under_directory	qed	1.0
fork_new_revision	qed	1.0
fork_new_seeds	qed	1.0
generate_report	qed	1.0
get_all_properties_dict	qed	1.0
get_default_group_id	qed	1.0
get_object_report_panel_contents	qed	1.0
get_object_report_panel_names	qed	1.0
get_objects	qed	1.0
get_project_report_panel_names	qed	1.0
get_property	qed	1.0
get_property_of_projects	qed	1.0
get_return_value	qed	1.0
get_user_data	qed	1.0
has_property	qed	1.0
import_report_panel	qed	1.0
import_report_panel_names	qed	1.0
is_connected	qed	1.0
is_workspace_open	qed	1.0
launch_connection	qed	1.0
list_properties	qed	1.0
load_db_state	qed	1.0
open_project	qed	1.0
pop_from_property	qed	1.0

continued...

Tcl Command	Tcl Package	Package Version
push_to_property	qed	1.0
refresh_reports	qed	1.0
run_analysis	qed	1.0
run_command	qed	1.0
sanitize_workspace	qed	1.0
set_properties	qed	1.0
set_property	qed	1.0
set_user_data	qed	1.0
wait_for_ready	qed	1.0
workspace_close	qed	1.0
workspace_new	qed	1.0
workspace_open	qed	1.0
write_object_reports_to_file	qed	1.0
test	qmtf	1.0
qshm_connect_to_quartus	qshm	1.0
qshm_disconnect_from_quartus	qshm	1.0
qshm_dispose_client	qshm	1.0
qshm_get_hub_key_prefix	qshm	1.0
qshm_get_parent_hub_key	qshm	1.0
qshm_obtain_client	qshm	1.0
qshm_send_request	qshm	1.0
qshm_send_server_state_query	qshm	1.0
qshm_set_context	qshm	1.0
add_row_to_table	report	2.1
create_report_panel	report	2.1
delete_report_panel	report	2.1
get_fitter_resource_usage	report	2.1
get_number_of_columns	report	2.1
get_number_of_rows	report	2.1
get_report_panel_column_index	report	2.1
get_report_panel_data	report	2.1
get_report_panel_id	report	2.1
get_report_panel_names	report	2.1
get_report_panel_row	report	2.1
get_report_panel_row_index	report	2.1
load_report	report	2.1

continued...

Tcl Command	Tcl Package	Package Version
read_xml_report	report	2.1
refresh_report_window	report	2.1
save_report_database	report	2.1
unload_report	report	2.1
write_report_panel	report	2.1
write_xml_report	report	2.1
all_clocks	sdc	1.5
all_inputs	sdc	1.5
all_outputs	sdc	1.5
all_registers	sdc	1.5
create_clock	sdc	1.5
create_generated_clock	sdc	1.5
derive_clocks	sdc	1.5
get_cells	sdc	1.5
get_clocks	sdc	1.5
get_nets	sdc	1.5
get_pins	sdc	1.5
get_ports	sdc	1.5
remove_clock_groups	sdc	1.5
remove_clock_latency	sdc	1.5
remove_clock_uncertainty	sdc	1.5
remove_disable_timing	sdc	1.5
remove_input_delay	sdc	1.5
remove_output_delay	sdc	1.5
reset_design	sdc	1.5
set_clock_groups	sdc	1.5
set_clock_latency	sdc	1.5
set_clock_uncertainty	sdc	1.5
set_disable_timing	sdc	1.5
set_false_path	sdc	1.5
set_input_delay	sdc	1.5
set_input_transition	sdc	1.5
set_max_delay	sdc	1.5
set_max_time_borrow	sdc	1.5
set_min_delay	sdc	1.5
set_multicycle_path	sdc	1.5

continued...

Tcl Command	Tcl Package	Package Version
set_output_delay	sdc	1.5
derive_clock_uncertainty	sdc_ext	2.0
derive_pll_clocks	sdc_ext	2.0
disable_min_pulse_width	sdc_ext	2.0
get_active_clocks	sdc_ext	2.0
get_fanins	sdc_ext	2.0
get_fanouts	sdc_ext	2.0
get_keepers	sdc_ext	2.0
get_nodes	sdc_ext	2.0
get_partitions	sdc_ext	2.0
get_registers	sdc_ext	2.0
remove_annotated_delay	sdc_ext	2.0
remove_clock	sdc_ext	2.0
reset_timing_derate	sdc_ext	2.0
set_active_clocks	sdc_ext	2.0
set_annotated_delay	sdc_ext	2.0
set_data_delay	sdc_ext	2.0
set_max_skew	sdc_ext	2.0
set_net_delay	sdc_ext	2.0
set_scc_mode	sdc_ext	2.0
set_time_format	sdc_ext	2.0
set_timing_derate	sdc_ext	2.0
add_to_collection	sta	1.0
check_timing	sta	1.0
create_report_histogram	sta	1.0
create_slack_histogram	sta	1.0
create_timing_netlist	sta	1.0
create_timing_summary	sta	1.0
delete_sta_collection	sta	1.0
delete_timing_netlist	sta	1.0
enable_ccpp_removal	sta	1.0
enable_sdc_extension_collections	sta	1.0
get_available_operating_conditions	sta	1.0
get_cell_info	sta	1.0
get_clock_domain_info	sta	1.0
get_clock_fmax_info	sta	1.0

continued...

Tcl Command	Tcl Package	Package Version
get_clock_info	sta	1.0
get_clock_pair_info	sta	1.0
get_datasheet	sta	1.0
get_default_sdc_file_names	sta	1.0
get_edge_info	sta	1.0
get_entity_instances	sta	1.0
get_min_pulse_width	sta	1.0
get_net_info	sta	1.0
get_node_info	sta	1.0
get_object_info	sta	1.0
get_operating_conditions	sta	1.0
get_operating_conditions_info	sta	1.0
get_partition_info	sta	1.0
get_path	sta	1.0
get_path_info	sta	1.0
get_pin_info	sta	1.0
get_point_info	sta	1.0
get_port_info	sta	1.0
get_register_info	sta	1.0
get_timing_paths	sta	1.0
import_sdc	sta	1.0
is_post_syn_sta	sta	1.0
locate	sta	1.0
print_total_sdc_processing_time	sta	1.0
query_collection	sta	1.0
read_sdc	sta	1.0
register_delete_timing_netlist_callback	sta	1.0
remove_from_collection	sta	1.0
report_advanced_io_timing	sta	1.0
report_asynch_cdc	sta	1.0
report_bottleneck	sta	1.0
report_cdc_viewer	sta	1.0
report_clock_fmax_summary	sta	1.0
report_clock_network	sta	1.0
report_clock_transfers	sta	1.0
report_clocks	sta	1.0

continued...

Tcl Command	Tcl Package	Package Version
report_datasheet	sta	1.0
report_ddr	sta	1.0
report_exceptions	sta	1.0
report_ini_usage	sta	1.0
report_logic_depth	sta	1.0
report_max_clock_skew	sta	1.0
report_max_skew	sta	1.0
report_metastability	sta	1.0
report_min_pulse_width	sta	1.0
report_neighbor_paths	sta	1.0
report_net_delay	sta	1.0
report_net_timing	sta	1.0
report_partitions	sta	1.0
report_path	sta	1.0
report_pipelining_info	sta	1.0
report_register_spread	sta	1.0
report_register_statistics	sta	1.0
report_retiming_restrictions	sta	1.0
report_route_net_of_interest	sta	1.0
report_rskm	sta	1.0
report_sdc	sta	1.0
report_skew	sta	1.0
report_tccs	sta	1.0
report_timing	sta	1.0
report_timing_by_source_files	sta	1.0
report_timing_tree	sta	1.0
report_ucp	sta	1.0
set_operating_conditions	sta	1.0
timing_netlist_exist	sta	1.0
update_timing_netlist	sta	1.0
use_timing_analyzer_style_escaping	sta	1.0
write_sdc	sta	1.0
close_session	stp	1.0
export_data_log	stp	1.0
open_session	stp	1.0
run	stp	1.0

continued...

Tcl Command	Tcl Package	Package Version
run_multiple_end	stp	1.0
run_multiple_start	stp	1.0
stop	stp	1.0
is_place	tdc	1.0
is_plan	tdc	1.0
is_post_route	tdc	1.0

3.1.1. ::quartus::backannotate

The following table displays information for the **::quartus::backannotate** Tcl package:

Tcl Package and Version	::quartus::backannotate 1.1
Description	This package contains the set of Tcl functions for back-annotating assignments for a project.
Availability	This package is available for loading in the following executables: qpro quartus quartus_cdb
Tcl Commands	get_back_annotation_assignments (::quartus::backannotate) on page 73 logiclock_back_annotate (::quartus::backannotate) on page 74

3.1.1.1. get_back_annotation_assignments (::quartus::backannotate)

The following table displays information for the `get_back_annotation_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::backannotate on page 73	
Syntax	`get_back_annotation_assignments [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Returns an output collection of back-annotation assignments. Each element of the collection is a list with the following format: { {<Source>} {<Destination>} {<Assignment name>} {<Assignment value>} {<Entity name>} }	
Example Usage	## Print out all the back-annotation assignments set asgn_col [get_back_annotation_assignments] foreach_in_collection asgn $asgn_col { ## Each element in the collection has the following ## format: ## { {<Source>} {<Destination>} {<Assignment name>} {<Assignment value>} {<Entity name>} } set from [lindex $asgn 0] set to [lindex $asgn 1] set name [lindex $asgn 2] set value [lindex $asgn 3]	

continued...

```
    set entity [lindex $asgn 4]
    puts "$entity : $name ($from -> $to) = $value"
}
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Back annotation failed -- design did not compile properly. Run a successful compilation before performing back-annotation.
	TCL_ERROR	1	ERROR: Project has no active revision. Make sure there is an open, active revision.
	TCL_ERROR	1	ERROR: No project is currently open. Open an exising project or create a new project.
	TCL_ERROR	1	ERROR: Device or device family does not support node location back annotation.
	TCL_ERROR	1	ERROR: Device or device family does not support LogicLock back annotation.
	TCL_ERROR	1	ERROR: Wrong number of arguments. For correct syntax, refer to help for the logiclock_back_annotate command.

3.1.1.2. logiclock_back_annotate (::quartus::backannotate)

The following table displays information for the `logiclock_back_annotate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::backannotate on page 73	
Syntax	`logiclock_back_annotate [-h | -help] [-long_help] [-exclude_from] [-exclude_to] [-from` *<source name>* `] [-lock] [-no_contents] [-no_delay_chain] [-no_demote_lab] [-no_demote_mac] [-no_demote_pin] [-no_demote_ram] [-no_dont_touch] [-path_exclude` *<path_exclude name>* `] [-region` *<region name>* `] [-remove_assignments] [-resource_filter` *<resource_filter value>* `] [-routing] [-to` *<destination name>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exclude_from`	Option to exclude the source node
	`-exclude_to`	Option to exclude the destination node
	`-from` *<source name>*	Name (or wildcard expression) of the source node to be back-annotated
	`-lock`	Option to lock back-annotated regions
	`-no_contents`	Option not to back-annotate contents
	`-no_delay_chain`	Option not to back-annotate delay chain settings
	`-no_demote_lab`	Option not to demote LAB or LE assignments
	`-no_demote_mac`	Option not to demote DSP block assignments
	`-no_demote_pin`	Option not to demote pin assignments
	`-no_demote_ram`	Option not to demote RAM assignments

continued...

	`-no_dont_touch`	Option not to set the don't_touch flag for each back-annotated node
	`-path_exclude` *<path_exclude name>*	Option to exclude the specified node from the path filter
	`-region` *<region name>*	Name (or wildcard expression) of region to be back-annotated
	`-remove_assignments`	Option to remove matching assignments instead of creating them
	`-resource_filter` *<resource_filter value>*	Option to use the resource filter
	`-routing`	Option to back-annotate the LogicLock region's routing
	`-to` *<destination name>*	Name (or wildcard expression) of the destination node to be back-annotated

Description

```
Back-annotates a LogicLock region and its contents.

When you use the "-routing" option, you must use the "-lock"
and "-no_demote_lab" options, without the "-no_contents" option,
or use the"-remove_assignments" option.

The "-remove_assignments" option removes all matching region
contents. When you use the "-remove_assignments" option, the
demotion options, "-no_contents" and "-lock", are not applicable.

The "-resource_filter" option allows you to back-annotate
only specific resource types on the device. For example:

    logiclock_back_annotate -resource_filter "COMBINATORIAL"

This command back-annotates all combinatorial nodes in the
design. The complete set of options is:

    COMBINATORIAL    combinatorial nodes
    REGISTER         registered nodes
    MEGA             M-RAMs
    MEDIUM           M4K memory blocks
    SMALL            M512 memory blocks
    IO               I/O elements
    MAC              DSP blocks

Intel recommends that you use a Verilog Quartus(R) Mapping
File (.vqm) as the source. When any of the advanced netlist
optimizations are enabled, it is possible for the Fitter
to create and rename nodes in the design during a place and
route operation. Back annotation requires that on subsequent
compilations the node names in the netlist match those in
the constraint file. Write out a VQM netlist and create a
new project using that netlist as its source. Copy all of
the existing constraint files into the new project directory
and remove all the design files except the new .vqm by using
the Add/Remove Files in a Project command (Project menu)
in the Quartus Prime GUI.

The Quartus Prime software will create a root region if you
back-annotate nodes that are not members of a LogicLock
region. The root region is device-size and locked. You can
make assignments to the root region but you cannot delete
it or modify its size or location.
```

Example Usage

```
# Open the project "example_project"
project_open example_project

# Compile the design
package require ::quartus::flow
execute_flow -compile

package require ::quartus::backannotate

# Back annotate all nodes and routing in the region "one_region"
logiclock_back_annotate -routing -lock -no_demote_lab -region one_region

# Back annotate the location of the nodes on all paths that
# start with a node that matches the "Data_in*" wildcard
```

continued...

```
# expression, and end with a node that matches the "Data_out*"
# wildcard expression
logiclock_back_annotate -from Data_in* -to Data_out*

# Back annotate the placement of all the registers in the design
logiclock_back_annotate -resource_filter "REGISTER"

# Close the project
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Back annotation failed -- design did not compile properly. Run a successful compilation before performing back-annotation.
	TCL_ERROR	1	ERROR: Project has no active revision. Make sure there is an open, active revision.
	TCL_ERROR	1	ERROR: No project is currently open. Open an exising project or create a new project.
	TCL_ERROR	1	ERROR: Device or device family does not support node location back annotation.
	TCL_ERROR	1	ERROR: The -routing option is used with incompatible options. To use the -routing option, you must use the -lock and -no_demote_lab options without the -no_contents option, or use the -remove_assignments option.
	TCL_ERROR	1	ERROR: Device or device family does not support LogicLock back annotation.
	TCL_ERROR	1	ERROR: Wrong number of arguments. For correct syntax, refer to help for the logiclock_back_annotate command.

3.1.2. ::quartus::board

The following table displays information for the **::quartus::board** Tcl package:

Tcl Package and Version	::quartus::board 1.0
Description	This package contains no general description.
Availability	This package is loaded by default in the following executables: quartus quartus_sh
Tcl Commands	activate_link (::quartus::board) on page 77 check_online_design_validity (::quartus::board) on page 77 deploy_par_file (::quartus::board) on page 77 download_par_file (::quartus::board) on page 78 get_board_design_path (::quartus::board) on page 78 get_board_devkits (::quartus::board) on page 79 get_board_families (::quartus::board) on page 79 get_board_info (::quartus::board) on page 79 get_board_vendors (::quartus::board) on page 80 get_design_description (::quartus::board) on page 80 get_design_development_kits (::quartus::board) on page 81 get_design_documents_info (::quartus::board) on page 81 get_design_download_link (::quartus::board) on page 82 get_design_families (::quartus::board) on page 82 get_design_info (::quartus::board) on page 83 get_design_quartus_versions (::quartus::board) on page 83 get_design_rich_description (::quartus::board) on page 84 get_ui_file (::quartus::board) on page 84 launch_qsys (::quartus::board) on page 85 load_design_info (::quartus::board) on page 85 reset_board_info (::quartus::board) on page 86

3.1.2.1. activate_link (::quartus::board)

The following table displays information for the `activate_link` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`activate_link [-h | -help] [-long_help] -link <link>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-link <link>`		The URL
Description	`This command will open the URL link from the default Browser.`		
Example Usage	`::board::activate_link -link https://my_link.com`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.2. check_online_design_validity (::quartus::board)

The following table displays information for the `check_online_design_validity` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`check_online_design_validity [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`The command will return '1' if the Board Package is allowed to load the online design examples.`		
Example Usage	`set valid [::board::check_online_design_validity]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.3. deploy_par_file (::quartus::board)

The following table displays information for the `deploy_par_file` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`deploy_par_file [-h | -help] [-long_help] -design_path <design_path> -destination_dir <destination_dir>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-design_path <design_path>`		The full path to the PAR file

continued...

	`-destination_dir` *<destination_dir>*		The directory to extract the PAR file to
Description	`This command will deploy the PAR file from the given design_path argument, and extract it to the given destination_dir.`		
Example Usage	`::board::deploy_par_file -design_path ./test_dir/top.par -destination_dir ./test_dir`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.4. download_par_file (::quartus::board)

The following table displays information for the `download_par_file` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`download_par_file [-h | -help] [-long_help] -destination_dir` *<destination_dir>* `-link` *<link>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-destination_dir` *<destination_dir>*		The directory to save the PAR file
	`-link` *<link>*		The URL to download the PAR file
Description	`This commands will download the PAR file from the given link argument and save the downloaded PAR file to the given destination_dir argument.`		
Example Usage	`::board::download_par_file -link https://design_link.com -destination_dir ./test_dir`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.5. get_board_design_path (::quartus::board)

The following table displays information for the `get_board_design_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`get_board_design_path [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`This command returns the path that contains the pre-installed designs.`		
Example Usage	`set path [::board::get_board_design_path]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.6. get_board_devkits (::quartus::board)

The following table displays information for the `get_board_devkits` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`get_board_devkits [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`This command returns the unique development kits of all the boards.`		
Example Usage	`set devkits [::board::get_board_devkits]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.7. get_board_families (::quartus::board)

The following table displays information for the `get_board_families` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`get_board_families [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`This command returns the unique device families of all the loaded boards.`		
Example Usage	`set families [::board::get_board_families]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.8. get_board_info (::quartus::board)

The following table displays information for the `get_board_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_board_info [-h | -help] [-long_help] [-family <family> ] [-name <name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-family <family>`	The device family that the board targeted to
	`-name <name>`	The name of the board

continued...

Description	This command returns the information of boards that contain the given board name and device family (case insensitive). If no argument is given, the command will return the information of all the available boards.		
Example Usage	`::board::get_board_info`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.9. get_board_vendors (::quartus::board)

The following table displays information for the `get_board_vendors` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`get_board_vendors [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	This command returns the unique vendors of all the boards.		
Example Usage	`set vendors [::board::get_board_vendors]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.10. get_design_description (::quartus::board)

The following table displays information for the `get_design_description` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_design_description [-h | -help] [-long_help] [-name` *<name>* `] [-source` *<online|quartus|my_download|all>* `] [-version` *<version>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name` *<name>*	The name of the design
	`-source` *<online|quartus|my_download|all>*	The Load From sources of the designs (Online, Pre-installed, Downloaded or All)
	`-version` *<version>*	The version of the design
Description	This commands returns the text description information of designs that contain the given design name. If no argument is supplied, the command will return the design description of all the available designs.	

continued...

Example Usage	`set description [::board::get_design_description -name pll]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.11. get_design_development_kits (::quartus::board)

The following table displays information for the `get_design_development_kits` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_design_development_kits [-h | -help] [-long_help] [-source <online|quartus|my_download|all> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-source <online|quartus|my_download|all>`	The Load From sources of the designs (Online, Pre-installed, Downloaded or All)
Description	`This command returns the unique development kits of the designs that are loaded from the source argument. If source argument isn't specified, the command will return unique development kits of all the loaded designs.`	
Example Usage	`set devkits [::board::get_design_development_kits]`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.12. get_design_documents_info (::quartus::board)

The following table displays information for the `get_design_documents_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_design_documents_info [-h | -help] [-long_help] [-name <name> ] [-source <online|quartus|my_download|all> ] [-version <version> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	The name of the design
	`-source <online|quartus|my_download|all>`	The Load From sources of the designs (Online, Pre-installed, Downloaded or All)
	`-version <version>`	The version of the design
Description	`This commands returns the document title(s) and URL(s) of documentation(s) of designs that contain the given design name. If no argument is supplied, the command will return the design documentation of all the available designs.`	

continued...

Example Usage	`set docs [::board::get_design_documents_info -name memory]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.13. get_design_download_link (::quartus::board)

The following table displays information for the `get_design_download_link` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_design_download_link [-h | -help] [-long_help] [-link_only] [-name <name> ] [-source <online|quartus|my_download|all> ] [-version <version> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-link_only`	If this option is specified, only a list of download link will be returned (instead of list of <design_name, download_link> pair)
	`-name <name>`	The name of the design
	`-source <online|quartus|my_download|all>`	The Load From sources of the designs (Online, Pre-installed, Downloaded or All)
	`-version <version>`	The version of the design
Description	`This command returns the URL to download the designs that contain the given design name. If no argument is supplied, the command will return the design download link of all the available designs.`	
Example Usage	`set link [::board::get_design_download_link -link_only]`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.14. get_design_families (::quartus::board)

The following table displays information for the `get_design_families` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_design_families [-h | -help] [-long_help] [-source <online|quartus|my_download|all> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-source <online|quartus|my_download|all>`	The Load From sources of the designs (Online, Pre-installed, Downloaded or All)

continued...

Description	This command returns the unique device families of the designs that are loaded from the source argument. If source argument isn't specified, the command will return unique device families of all the loaded designs.		
Example Usage	`set families [::board::get_design_families]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.15. get_design_info (::quartus::board)

The following table displays information for the `get_design_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`get_design_info [-h | -help] [-long_help] [-family` *<family>* `] [-name` *<name>* `] [-source` *<online|quartus|my_download|all>* `] [-version` *<version>* `]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-family` *<family>*	The targeted device family of the design	
	`-name` *<name>*	The name of the design	
	`-source` *<online|quartus|my_download|all>*	The Load From sources of the designs (Online, Pre-installed, Downloaded or All)	
	`-version` *<version>*	The version of the design	
Description	This command returns the information of designs that contain the given design name and device family (case insensitive). If no argument is given, the command will return the information of all the available designs.		
Example Usage	`set designs [::board::get_design_info -name pll -family agilex -source online]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.16. get_design_quartus_versions (::quartus::board)

The following table displays information for the `get_design_quartus_versions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_design_quartus_versions [-h | -help] [-long_help] [-max_count` *<max_count>* `] [-source` *<online|quartus|my_download|all>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-max_count` *<max_count>*	The maximum number of Quartus versions returned (the last max_count from the latest Quartus version)

continued...

	`-source <online|quartus|my_download|all>`	The Load From sources of the designs (Online, Pre-installed, Downloaded or All)
Description	This command returns the unique Quartus versions of the designs that are loaded from the source argument. If source argument isn't specified, the command will return unique Quartus versions of all the loaded designs.	
Example Usage	`set quartus_versions [::board::get_design_quartus_versions -max_count 5]`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.17. get_design_rich_description (::quartus::board)

The following table displays information for the `get_design_rich_description` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_design_rich_description [-h | -help] [-long_help] [-name <name> ] [-source <online|quartus|my_download|all> ] [-version <version> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	The name of the design
	`-source <online|quartus|my_download|all>`	The Load From sources of the designs (Online, Pre-installed, Downloaded or All)
	`-version <version>`	The version of the design
Description	This commands returns the rich-text description information of designs that contain the given design name. If no argument is supplied, the command will return the rich-text description of all the available designs.	
Example Usage	`set rich_description [::board::get_design_rich_description]`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.18. get_ui_file (::quartus::board)

The following table displays information for the `get_ui_file` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`get_ui_file [-h | -help] [-long_help] [-destination_dir <destination_dir> ] [-download_url <download_url> ] [-name <name> ] [-source <online|quartus|my_download|all> ] [-version <version> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-destination_dir <destination_dir>`	The directory to save the UI file

continued...

	`-download_url` <*download_url*>		The download URL of the design
	`-name` <*name*>		The UI filename of the design
	`-source` <*online*|*quartus*|*my_download*|*all*>		The Load From sources of the designs (Online, Pre-installed, Downloaded or All)
	`-version` <*version*>		The version of the design
Description	`This commands returns the UI filename of designs that contain the given design name. If no argument is supplied, the command will return the UI filename of all the available designs.`		
Example Usage	`set ui_file [::board::get_ui_file -name pll]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.19. launch_qsys (::quartus::board)

The following table displays information for the `launch_qsys` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`launch_qsys [-h | -help] [-long_help] [-board` <*board*> `] [-qpf_file` <*qpf_file*> `] [-qsf_file` <*qsf_file*> `] [-qsys_file` <*qsys_file*> `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-board` <*board*>		The name of the board (e.g. Intel Agilex F-Series Transceiver-SoC Development Kit DK-SI-AGF014EA)
	`-qpf_file` <*qpf_file*>		The full path to the Quartus Project file (QPF, *.qpf file)
	`-qsf_file` <*qsf_file*>		The full path to the Quartus Setting file (QSF, *.qsf file)
	`-qsys_file` <*qsys_file*>		The full path to the Platform Designer System (QSYS, *.qsys file)
Description	`This command will launch Platform Designer application in GUI mode. With qsys_file, qpf_file and board argument supplied, the Platform Designer will open the project and skip the Open System dialog.`		
Example Usage	`::board::launch_qsys`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.20. load_design_info (::quartus::board)

The following table displays information for the `load_design_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76	
Syntax	`load_design_info [-h | -help] [-long_help] [-append] [-download_path` <*download_path*> `] [-source` <*online*|*quartus*|*my_download*> `]`	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`		Long help with examples and possible return values
	`-append`		If this option is specified, the designs information will be appended to the existing loaded designs
	`-download_path` *<download_path>*		A list of paths that contains the PAR files. This argument will be ignored for source = online | quartus option
	`-source` *<online|quartus|my_download>*		Load From sources of the designs (Online, Pre-installed or Downloaded)
Description	`The command loads the designs based on the source argument specified.`		
Example Usage	`::board::load_design_info -source quartus` `::board::load_design_info -source my_download -download_path ../project_dir -append`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.2.21. reset_board_info (::quartus::board)

The following table displays information for the `reset_board_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::board on page 76		
Syntax	`reset_board_info [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Clear the loaded board information.`		
Example Usage	`::board::reset_board_info`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3. ::quartus::bpps

The following table displays information for the **::quartus::bpps** Tcl package:

Tcl Package and Version	::quartus::bpps 1.0
Description	`This package provides non-backend support for pin-planner mode in Interface Planner.`
Availability	`This package is loaded by default in the following executables:` `qacv` `qppl` `qpro` `quartus` `quartus_bpps` `quartus_drc` `quartus_pdp` `quartus_pow` `quartus_sta` `quartus_staw`

continued...

Tcl Commands	bpps::apply_assignments (::quartus::bpps) on page 87 bpps::check_plan (::quartus::bpps) on page 88 bpps::export_constraints_to_qsf (::quartus::bpps) on page 88 bpps::get_cell_info (::quartus::bpps) on page 89 bpps::get_device (::quartus::bpps) on page 90 bpps::get_hdbpath_from_id (::quartus::bpps) on page 90 bpps::get_id_from_hdbpath (::quartus::bpps) on page 91 bpps::get_location_info (::quartus::bpps) on page 91 bpps::get_placement (::quartus::bpps) on page 92 bpps::get_placement_info (::quartus::bpps) on page 92 bpps::get_placements (::quartus::bpps) on page 93 bpps::get_placements_of_group (::quartus::bpps) on page 93 bpps::harden_cell (::quartus::bpps) on page 94 bpps::harden_cells (::quartus::bpps) on page 95 bpps::initialize (::quartus::bpps) on page 95 bpps::load_floorplan (::quartus::bpps) on page 96 bpps::place_cells (::quartus::bpps) on page 96 bpps::read_tpl_placement (::quartus::bpps) on page 97 bpps::remove_invalid_reports (::quartus::bpps) on page 98 bpps::report_all (::quartus::bpps) on page 98 bpps::report_cell_connectivity (::quartus::bpps) on page 98 bpps::report_cell_placement_reasons (::quartus::bpps) on page 99 bpps::report_cells (::quartus::bpps) on page 99 bpps::report_clocks (::quartus::bpps) on page 100 bpps::report_legal_cell_locations (::quartus::bpps) on page 101 bpps::report_location_types (::quartus::bpps) on page 101 bpps::report_locations (::quartus::bpps) on page 102 bpps::report_regions (::quartus::bpps) on page 102 bpps::report_summary (::quartus::bpps) on page 103 bpps::reset_plan (::quartus::bpps) on page 103 bpps::save_floorplan (::quartus::bpps) on page 104 bpps::save_pin_assignments (::quartus::bpps) on page 104 bpps::select_dr_ips (::quartus::bpps) on page 105 bpps::set_mode (::quartus::bpps) on page 105 bpps::shutdown (::quartus::bpps) on page 105 bpps::soften_cell (::quartus::bpps) on page 106 bpps::soften_cells (::quartus::bpps) on page 106 bpps::undo_last_placement (::quartus::bpps) on page 107 bpps::unplace_cells (::quartus::bpps) on page 107 bpps::update_pdpw (::quartus::bpps) on page 108 bpps::validate_placement (::quartus::bpps) on page 108 bpps::write_plan (::quartus::bpps) on page 109 bpps::write_tpl_placement (::quartus::bpps) on page 109

3.1.3.1. bpps::apply_assignments (::quartus::bpps)

The following table displays information for the `bpps::apply_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::apply_assignments [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`In classic mode, applies all changes to constraints and reloads them into Interface Planner. After the platform has been updated with the constraints, placement operations can be performed.` `In pin planner mode, loads the QSF constraints related to pin assignments.`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `blueprint::shutdown` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.2. bpps::check_plan (::quartus::bpps)

The following table displays information for the `bpps::check_plan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::check_plan [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`In classic mode, checks the legality of the current plan` `In pin planner mode, this will be a stub. Assignments are checked real time, no backend engine to check legality of anything.`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `bpps::check_plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.3. bpps::export_constraints_to_qsf (::quartus::bpps)

The following table displays information for the `bpps::export_constraints_to_qsf` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::export_constraints_to_qsf [-h | -help] [-long_help] [-bb_locations] [-close_pdp] [-disabled] [-pin_locations] [-tile_locations]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-bb_locations`	Write out building block location assignments
	`-close_pdp`	Send call back to PDPW to close after exporting is done
	`-disabled`	Write out disabled assignments
	`-pin_locations`	Write out pin location assignments
	`-tile_locations`	Write out tile location assignments
Description	`In Tile Planner mode, export constraints to qsf file`	
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `set io_cells [bpps::get_cells -unplaced -type IO_CLUSTER]` `bpps::place_cells -cells $io_cells`	

continued...

	bpps::validate_placement bpps::export_constraints -disabled -tile_locations -bb_locations project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.4. bpps::get_cell_info (::quartus::bpps)

The following table displays information for the `bpps::get_cell_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::get_cell_info [-h | -help] [-long_help] [-children] [-guide_cell_id] [-ip_type] [-links] [-location] [-name] [-parent] [-type] <cell_id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-children`		Return the the cell id of the children cells
	`-guide_cell_id`		Returns the guide cell's elem_id
	`-ip_type`		Returns the IP type if the cell is an IP cell or an empty string otherwise
	`-links`		Return the given design element's connections to other cells
	`-location`		Returns the location ID if the cell is placed or an empty string otherwise
	`-name`		Return the cell name of the cell id
	`-parent`		Return the the cell id of the parent cells
	`-type`		Return the the type of the cell
	<cell_id>		Single cell id
Description	`Gets information about the specified cell (referenced by cell ID).` `You can obtain cell using the periph::get_cells Tcl command.`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `foreach cell [periph::get_cells -type IO_CLUSTER] {` `  puts "Found cell ID $cell named [periph::get_cell_info -name $cell]"` `}` `blueprint::shutdown` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.5. bpps::get_device (::quartus::bpps)

The following table displays information for the `bpps::get_device` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::get_device [-h | -help] [-long_help] [-compress]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-compress`		Compress requested data
Description	`Internal function to get the device tree in json, this gets the complete device model.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.6. bpps::get_hdbpath_from_id (::quartus::bpps)

The following table displays information for the `bpps::get_hdbpath_from_id` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::get_hdbpath_from_id [-h | -help] [-long_help] [-design_cell_id` *<design_cell_id>* `] [-device_loc_id` *<device_loc_id>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-design_cell_id` *<design_cell_id>*		design cell ID
	`-device_loc_id` *<device_loc_id>*		device location ID
Description	`In classic mode, load the floorplan from a Interface Planner floorplan file` `In pin planner mode, this is a stub.`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `bpps::place_cells -unplaced_cells` `bpps::save_floorplan -filename onewire_blueprint_floorplan.plan` `bpps::load_floorplan -filename onewire_blueprint_floorplan.plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.7. bpps::get_id_from_hdbpath (::quartus::bpps)

The following table displays information for the `bpps::get_id_from_hdbpath` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::get_id_from_hdbpath [-h | -help] [-long_help] [-design_cell` *<design_cell>* `] [-device_loc` *<device_loc>* `]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-design_cell` *<design_cell>*	HDB Path of design cell	
	`-device_loc` *<device_loc>*	HDB Path of device location	
Description	`In classic mode, load the floorplan from a Interface Planner floorplan file` `In pin planner mode, this is a stub.`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `bpps::place_cells -unplaced_cells` `bpps::save_floorplan -filename onewire_blueprint_floorplan.plan` `bpps::load_floorplan -filename onewire_blueprint_floorplan.plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.8. bpps::get_location_info (::quartus::bpps)

The following table displays information for the `bpps::get_location_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::get_location_info [-h | -help] [-long_help] [-children] [-gid] [-name] [-parents] [-placed_cells] [-properties] [-type]` *<location_id>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-children`	Query the children location IDs
	`-gid`	Query the gid of the location IDs
	`-name`	Return the location name of the location id
	`-parents`	Query the parent location IDs
	`-placed_cells`	Return the placed cells at the location id
	`-properties`	Return the device location properties in json
	`-type`	Return the location type of the location id

continued...

	<location_id>		location id
Description	Gets information about the specified location (referenced by location ID). You can obtain location using the periph::get_locations Tcl command or using the bpps::get_location_info -properties <loc_id> Tcl command		
Example Usage	project_open onewire_nf blueprint::initialize periph::update_plan foreach cell [periph::get_cells -placed] { puts "Found cell ID $cell named [periph::get_cell_info -name $cell] placed in location [periph::get_cell_info -location $cell] named [periph::get_location_info -name [periph::get_cell_info -location $cell]]" }		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.9. bpps::get_placement (::quartus::bpps)

The following table displays information for the `bpps::get_placement` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::get_placement [-h | -help] [-long_help] [-compress]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-compress`	Compress requested data
Description	Return information about design placement	
Example Usage	This command currently contains no example usage.	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.10. bpps::get_placement_info (::quartus::bpps)

The following table displays information for the `bpps::get_placement_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::get_placement_info [-h | -help] [-long_help] [-placement]` *<placement_id>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-placement`	Return the placement as a list of cell/id pairs
	<placement_id>	Single placement id

continued...

Description	In classic mode, return information about a given placement In pin planner mode, looking up placement objects is not supported, no such thing.		
Example Usage	This command currently contains no example usage.		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.11. bpps::get_placements (::quartus::bpps)

The following table displays information for the `bpps::get_placements` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::get_placements [-h | -help] [-long_help] [-ips] <cell_id>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-ips`	Get placements for IPs only	
	<cell_id>	Single cell id	
Description	In classic mode, returns a vector of placements for the supplied cell In pin planner mode, returns a vector of compatible locations for the supplied pin		
Example Usage	This command currently contains no example usage.		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied design pin ID *<string>* is invalid.
	TCL_ERROR	1	ERROR: At least one pin ID must be supplied, but no IDs were supplied
	TCL_ERROR	1	ERROR: The supplied pin id *<string>* is not a placeable pin.
	TCL_ERROR	1	ERROR: *<string>* IDs expected, but *<string>* were supplied

3.1.3.12. bpps::get_placements_of_group (::quartus::bpps)

The following table displays information for the `bpps::get_placements_of_group` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::get_placements_of_group [-h | -help] [-long_help] -cells <cells> [-ips]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-cells <cells>`		One or more cell ids
	`-ips`		Get placements for IPs only
Description	`Given a list of design cell IDs, returns a vector of possible placement IDs for all the cells.`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `bpps::place_cells -unplaced_cells` `bpps::check_plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied ID <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one ID must be supplied, but no IDs were supplied
	TCL_ERROR	1	ERROR: <*string*> IDs expected, but <*string*> were supplied

3.1.3.13. bpps::harden_cell (::quartus::bpps)

The following table displays information for the `bpps::harden_cell` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::harden_cell [-h | -help] [-long_help] -cell <cell>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-cell <cell>`		Harden the existing placement of the specificed cell
Description	`In modes that support soft / hard placements (ie. Tile Planner mode), hardens the existing placement of the specificed cell.` `If the cell is subsequently unplaced and placed again, the placement soft / hard attribute will be based` `on the new placement action.` `For modes that do not support soft / hard placements, nothing is performed.`		
Example Usage	`bpps::harden_cell -cell <design_cell_id>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.14. bpps::harden_cells (::quartus::bpps)

The following table displays information for the `bpps::harden_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::harden_cells [-h | -help] [-long_help] -cells <cells>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-cells <cells>`		Harden the existing placement of the specificed cells
Description	In modes that support soft / hard placements (ie. Tile Planner mode), hardens the existing placement of the specificed cell. If the cell is subsequently unplaced and placed again, the placement soft / hard attribute will be based on the new placement action. For modes that do not support soft / hard placements, nothing is performed.		
Example Usage	`bpps::harden_cells -cells [<design_cell_id>]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.15. bpps::initialize (::quartus::bpps)

The following table displays information for the `bpps::initialize` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::initialize [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	Replaces blueprint::initialize command. It will create the design and device models without a backend separate-exe engine.		
Example Usage	`project_open onewire_nf` `bpps::initialize` `bpps::update_plan` `bpps::shutdown` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.16. bpps::load_floorplan (::quartus::bpps)

The following table displays information for the `bpps::load_floorplan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::load_floorplan [-h | -help] [-long_help] -filename <filename>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-filename <filename>`	Filename to load	
Description	In classic mode, load the floorplan from a Interface Planner floorplan file In pin planner mode, this is a stub.		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `bpps::place_cells -unplaced_cells` `bpps::save_floorplan -filename onewire_blueprint_floorplan.plan` `bpps::load_floorplan -filename onewire_blueprint_floorplan.plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.17. bpps::place_cells (::quartus::bpps)

The following table displays information for the `bpps::place_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::place_cells [-h | -help] [-long_help] [-cell_location <cell_location> ] [-cells <cells> ] [-dont_revert_on_fail] [-fixed_cells] [-placement <placement> ] [-unplaced_cells]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cell_location <cell_location>`	Cell location id pair to place cells into
	`-cells <cells>`	One or more cell ids
	`-dont_revert_on_fail`	Option to specify that the best partial placement should be saved on the undo stack upon a placement failure
	`-fixed_cells`	Place all unplaced cells
	`-placement <placement>`	Place cells according to a placement. A placement is a special object that comes from the bpps::get_placements Tcl command
	`-unplaced_cells`	Place all unplaced cells

continued...

Description	In classic mode, performs a placement on the supplied cells In pin planner mode, auto assigns all the pins in the design if possible. Only -cell_ids <cell_id_list> and -cell_location <(cell_id, loc_id)> are actually used. If, exclusively, one of these are not specified, the command does NOTHING.		
Example Usage	project_open onewire_nf blueprint::initialize bpps::update_plan bpps::place_cells -unplaced_cells bpps::check_plan project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied ID <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one ID must be supplied, but no IDs were supplied
	TCL_ERROR	1	ERROR: <*string*> IDs expected, but <*string*> were supplied

3.1.3.18. bpps::read_tpl_placement (::quartus::bpps)

The following table displays information for the `bpps::read_tpl_placement` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::read_tpl_placement [-h | -help] [-long_help] -filename` <*filename*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-filename` <*filename*>		Filename to write to
Description	In TilePlanner mode, read placement from a JSON file. Nothing happens (should not be available in GUI) in other modes.		
Example Usage	project_open onewire_nf blueprint::initialize bpps::update_plan bpps::place_cells -unplaced_cells bpps::check_plan bpps::write_tpl_placement -filename onewire_blueprint_assignments.tcl project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Filename provided incorrectly

3.1.3.19. bpps::remove_invalid_reports (::quartus::bpps)

The following table displays information for the `bpps::remove_invalid_reports` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::remove_invalid_reports [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	In classic mode, remove all invalid report In pin planner mode, this is a stub call.		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.20. bpps::report_all (::quartus::bpps)

The following table displays information for the `bpps::report_all` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_all [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	In classic mode, create all default summary reports. In pin planner mode, this is a stub call.		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.21. bpps::report_cell_connectivity (::quartus::bpps)

The following table displays information for the `bpps::report_cell_connectivity` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_cell_connectivity [-h | -help] [-long_help] [-fanins] [-fanouts] [-panel_name` *<name>* `] <cell_id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values

continued...

	`-fanins`		Report only the fanins of the cell
	`-fanouts`		Report only the fanouts of the cell
	`-panel_name` *<name>*		Sends the results to the panel and specifies the name of the new panel
	<cell_id>		Single cell id
Description	`In classic mode, creates a report of the connectivity for a cell.` `In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.22. bpps::report_cell_placement_reasons (::quartus::bpps)

The following table displays information for the `bpps::report_cell_placement_reasons` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_cell_placement_reasons [-h | -help] [-long_help] [-panel_name` *<name>* `] <cell_id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name` *<name>*		Sends the results to the panel and specifies the name of the new panel
	<cell_id>		Single cell id
Description	`In classic mode, creates a report of all the locations a particular cell can be placed and the reasons` `it cannot be placed there` `In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.23. bpps::report_cells (::quartus::bpps)

The following table displays information for the `bpps::report_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_cells [-h | -help] [-long_help] [-name` *<name>* `] [-panel_name` *<name>* `] [-placed] [-type` *<type>* `] [-unplaced]`		
Arguments	`-h | -help`		Short help

continued...

	`-long_help`		Long help with examples and possible return values
	`-name <name>`		Filter the list of placed cells specifying a name. Wildcards are supported.
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-placed`		Report the list of placed cells
	`-type <type>`		Filter the list of placed cells specifying a list of types
	`-unplaced`		Report the list of unplaced cells
Description			
	`In classic mode, returns a list of periphery cells based on the specified criteria.` `In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.24. bpps::report_clocks (::quartus::bpps)

The following table displays information for the `bpps::report_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_clocks [-h | -help] [-long_help] [-panel_name <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
Description			
	`In classic mode, show the signals that are using low-skew routing networks (clock networks) in the device.` `If applicable, also show any signals that were considered for automatic clock network promotion, but were not promoted.` `In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.25. bpps::report_legal_cell_locations (::quartus::bpps)

The following table displays information for the `bpps::report_legal_cell_locations` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_legal_cell_locations [-h | -help] [-long_help] [-panel_name <name> ] <cell_id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	<cell_id>		Single cell id
Description	`In classic mode, creates a report of the legal periphery cell locations of a cell` `In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.26. bpps::report_location_types (::quartus::bpps)

The following table displays information for the `bpps::report_location_types` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_location_types [-h | -help] [-long_help] [-panel_name <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
Description	`In classic mode, creates a report of the location types in the periphery` `In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.27. bpps::report_locations (::quartus::bpps)

The following table displays information for the `bpps::report_locations` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_locations [-h | -help] [-long_help] [-panel_name <name> ] <type>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	<type>		location type to query
Description	`In classic mode, Creates a report of the locations for the requested type in the periphery` `In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.28. bpps::report_regions (::quartus::bpps)

The following table displays information for the `bpps::report_regions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_regions [-h | -help] [-long_help] [-panel_name <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
Description	`In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.29. bpps::report_summary (::quartus::bpps)

The following table displays information for the `bpps::report_summary` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::report_summary [-h | -help] [-long_help] [-panel_name <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
Description	`In pin planner mode, this is a stub call.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.30. bpps::reset_plan (::quartus::bpps)

The following table displays information for the `bpps::reset_plan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::reset_plan [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`In classic mode, reverts the current design to be unplaced and without assignments applied` `In pin planner mode, removes all the user created pin assignments. Keeps the original assignments. (currently just a stub still)`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `bpps::reset_plan` `blueprint::shutdown` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.31. bpps::save_floorplan (::quartus::bpps)

The following table displays information for the `bpps::save_floorplan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::save_floorplan [-h | -help] [-long_help] -filename <filename>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-filename <filename>`	Filename to write to	
Description	In classic mode, write the Interface Planner floorplan that can be reloaded in Interface Planner In pin planner mode, write the user pin assignments as constraints to QSF file. It is preferred to use the new save_pin_assignments call instead in pin planner mode.		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `set io_cells [bpps::get_cells -unplaced -type IO_CLUSTER]` `bpps::place_cells -cells $io_cells` `bpps::save_floorplan -filename onewire_blueprint_floorplan.plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.32. bpps::save_pin_assignments (::quartus::bpps)

The following table displays information for the `bpps::save_pin_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::save_pin_assignments [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	Write the Interface Planner floorplan that can be reloaded in Interface Planner		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `bpps::save_pin_assignments` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.33. bpps::select_dr_ips (::quartus::bpps)

The following table displays information for the `bpps::select_dr_ips` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::select_dr_ips [-h | -help] [-long_help] [-deselect] [-ips <ips> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-deselect`		Deselects any selections previously made
	`-ips <ips>`		One or more ip ids that exist in DR groups
Description	`Selects the DR ips. Any placement sync requests afterwards will only return placements for those selected DR IPs. All other non-DR placements are still returned.` `Any IP IDs passed over that are inside DR groups will be ignored.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.34. bpps::set_mode (::quartus::bpps)

The following table displays information for the `bpps::set_mode` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::set_mode [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Internal use only for PDPW to set the mode of the middleware. Also defines what plugins will be loaded`		
Example Usage	`DO NOT call this explicitly`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.35. bpps::shutdown (::quartus::bpps)

The following table displays information for the `bpps::shutdown` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::shutdown [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values

continued...

Description	Shutdown Interface Planner.		
Example Usage	project_open onewire_nf blueprint::initialize bpps::update_plan blueprint::shutdown project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.36. bpps::soften_cell (::quartus::bpps)

The following table displays information for the `bpps::soften_cell` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::soften_cell [-h | -help] [-long_help] -cell <cell>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cell <cell>`	Soften the existing placement of the specificed cell
Description	In modes that support soft / hard placements (ie. Tile Planner mode), softens the existing placement of the specificed cell. If the cell is subsequently unplaced and placed again, the placement soft / hard attribute will be based on the new placement action. For modes that do not support soft / hard placements, nothing is performed.	
Example Usage	bpps::soften_cell -cell <design_cell_id>	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.37. bpps::soften_cells (::quartus::bpps)

The following table displays information for the `bpps::soften_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::soften_cells [-h | -help] [-long_help] [-cells <cells> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cells <cells>`	One or more cell ids to soften
Description	In modes that support soft / hard placements (ie. Tile Planner mode), softens the existing placement of the specificed cell. If the cell is subsequently unplaced and placed again, the placement soft / hard attribute will be based	

continued...

	on the new placement action. For modes that do not support soft / hard placements, nothing is performed.		
Example Usage	bpps::soften_cells -cells [<design_cell_ids>]		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.38. bpps::undo_last_placement (::quartus::bpps)

The following table displays information for the `bpps::undo_last_placement` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::undo_last_placement [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	In classic mode, undo the last placement or unplacement operation. In classic mode, undo the last pin assignment or assignment removal operation.	
Example Usage	This command currently contains no example usage.	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.39. bpps::unplace_cells (::quartus::bpps)

The following table displays information for the `bpps::unplace_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::unplace_cells [-h | -help] [-long_help] [-cells <cells> ] [-placed_cells]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cells <cells>`	One or more cell ids
	`-placed_cells`	Unplace all placed cells
Description	In classic mode, removes the placement from the specified cells. Any constraints for the cells remain, but the cell no longer has a placement. In pin planner mode, similar to classic mode, unassigns pin locations made within this session.	
Example Usage	project_open onewire_nf blueprint::initialize bpps::update_plan	

continued...

	`` bpps::unplace_cells -placed_cells bpps::check_plan project_close ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.40. bpps::update_pdpw (::quartus::bpps)

The following table displays information for the `bpps::update_pdpw` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::update_pdpw [-h | -help] [-long_help] [-assignments] [-placement]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-assignments`		Indicates assignment model needs updating
	`-placement`		Indicates placement needs updating
Description	``` In classic mode, this command update everything that needs updating in pdpw. This essentially sends a single TCL command to pdpw to update everything as needed. Used in the TCL proc source wrapper only. In pin planner mode, this command is just a stub. ```		
Example Usage	``` project_open onewire_nf blueprint::initialize bpps::update_pdpw -pdp_state [blueprint_internal::get_pdp_state] ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.41. bpps::validate_placement (::quartus::bpps)

The following table displays information for the `bpps::validate_placement` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::validate_placement [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	``` Removes the exception to ignore the given project assignments. The result is the project assignments will take affect on the active design. ```		
Example Usage	``` This command currently contains no example usage. ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.42. bpps::write_plan (::quartus::bpps)

The following table displays information for the `bpps::write_plan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86		
Syntax	`bpps::write_plan [-h | -help] [-long_help] [-clocks] [-disabled] -filename <filename> [-force] [-other_locations] [-pin_locations]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-clocks`	Write out clock assignments	
	`-disabled`	Write out disabled assignments	
	`-filename <filename>`	Filename to write to	
	`-force`	Force the creation of the plan	
	`-other_locations`	Write out other location assignments	
	`-pin_locations`	Write out pin location assignments	
Description	`In classic mode, export the floorplan constraints Tcl script` `In pin planner mode, does nothing (we're not exporting to any TCL file, instead we write to QSF directly in save_floorplan, or save_pin_assignments (recommended), calls.`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `bpps::update_plan` `bpps::place_cells -unplaced_cells` `bpps::check_plan` `bpps::write_plan -filename onewire_blueprint_assignments.tcl` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.3.43. bpps::write_tpl_placement (::quartus::bpps)

The following table displays information for the `bpps::write_tpl_placement` Tcl command:

Tcl Package and Version	Belongs to ::quartus::bpps on page 86	
Syntax	`bpps::write_tpl_placement [-h | -help] [-long_help] -filename <filename>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-filename <filename>`	Filename to write to
Description	`In TilePlanner mode, write out the placement JSON file. Nothing happens (should not be available in GUI) in other modes.`	

continued...

Example Usage	<pre>project_open onewire_nf blueprint::initialize bpps::update_plan bpps::place_cells -unplaced_cells bpps::check_plan bpps::write_tpl_placement -filename onewire_blueprint_assignments.tcl project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Filename provided incorrectly

3.1.4. ::quartus::chip_planner

The following table displays information for the **::quartus::chip_planner** Tcl package:

Tcl Package and Version	::quartus::chip_planner 2.0
Description	<pre>This package contains the set of Tcl functions for identifying and modifying resource usage and routing with the Chip Planner.</pre>
Availability	<pre>This package is available for loading in the following executables: qacv qpro quartus quartus_cdb</pre>
Tcl Commands	check_node (::quartus::chip_planner) on page 110 close_chip_planner (::quartus::chip_planner) on page 111 design_has_ace_support (::quartus::chip_planner) on page 111 design_has_encrypted_ip (::quartus::chip_planner) on page 112 get_info_parameters (::quartus::chip_planner) on page 112 get_iports (::quartus::chip_planner) on page 113 get_node_by_name (::quartus::chip_planner) on page 114 get_node_info (::quartus::chip_planner) on page 114 get_nodes (::quartus::chip_planner) on page 115 get_oports (::quartus::chip_planner) on page 115 get_port_by_type (::quartus::chip_planner) on page 116 get_port_info (::quartus::chip_planner) on page 117 get_sp_pin_list (::quartus::chip_planner) on page 118 get_tile_power_setting (::quartus::chip_planner) on page 118 read_netlist (::quartus::chip_planner) on page 119 set_batch_mode (::quartus::chip_planner) on page 119

3.1.4.1. check_node (::quartus::chip_planner)

The following table displays information for the check_node Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110	
Syntax	`check_node [-h | -help] [-long_help] [-gen_id <gen id> ] [-node <node id> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-gen_id <gen id>`	Node generic ID

continued...

	-node <*node id*>		Node ID
Description	Checks whether the specified node is legal. Returns 1, if the node is legal. Returns 0, otherwise. Even if a node is legal, you still must run the check_netlist_and_save command to verify the node legality within the netlist.		
Example Usage	check_node -node 3		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Conflicting arguments. Consult help for the Tcl command for details.
	TCL_ERROR	1	ERROR: Illegal node generic ID: %u. Specify a legal node generic ID.
	TCL_ERROR	1	ERROR: Illegal node ID: %u. Specify a legal node ID.
	TCL_ERROR	1	ERROR: The node you specified is a legalization node. Modification of legalization nodes is not supported.
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.2. close_chip_planner (::quartus::chip_planner)

The following table displays information for the `close_chip_planner` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	close_chip_planner [-h | -help] [-long_help]		
Arguments	-h | -help		Short help
	-long_help		Long help with examples and possible return values
Description	Releases the chip planner netlist from use.		
Example Usage	close_chip_planner		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.4.3. design_has_ace_support (::quartus::chip_planner)

The following table displays information for the `design_has_ace_support` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110	
Syntax	design_has_ace_support [-h | -help] [-long_help]	
Arguments	-h | -help	Short help

continued...

	`-long_help`		Long help with examples and possible return values
Description	`Determines whether Chip Planner operations can be performed on the current design.`		
Example Usage	`design_has_ace_support`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.4. design_has_encrypted_ip (::quartus::chip_planner)

The following table displays information for the `design_has_encrypted_ip` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`design_has_encrypted_ip [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Determines whether the current design contains encrypted IP.` `Returns 1, if the design contains encrypted IP. You may be able to view or edit individual nodes of the design if they are not part of an encrypted IP. To check individual nodes, use the command "get_node_info -node <node id> -info encrypted". Returns 0, otherwise.`		
Example Usage	`design_has_encrypted_ip`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.5. get_info_parameters (::quartus::chip_planner)

The following table displays information for the `get_info_parameters` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110	
Syntax	`get_info_parameters [-h | -help] [-long_help] [-file` *<file name>* `] [-for_chip]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-file` *<file name>*	Name of output file
	`-for_chip`	Option to display all of the chip info parameters

continued...

Description	```Returns a Tcl list of information parameters. When you use the -file option, the list is redirected to the specified output file. If the output file already exists, it is overwritten without warning.```		
Example Usage	```get_info_parameters get_info_parameters -file the_list.txt```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.4.6. get_iports (::quartus::chip_planner)

The following table displays information for the `get_iports` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_iports [-h | -help] [-long_help] [-as_gen_id] [-gen_id` *<gen id>* `] [-node` *<node id>* `] [-src_gen_id` *<gen id>* `]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-as_gen_id`	Option to return results as generic ID	
	`-gen_id` *<gen id>*	Node generic ID	
	`-node` *<node id>*	Node id	
	`-src_gen_id` *<gen id>*	Source port generic ID	
Description	```Returns a collection of input ports for the specified node. You can use the collection with the "foreach_in_collection" command.```		
Example Usage	```get_iports -node 3 get_iports -src_gen_id 5```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Conflicting arguments. Consult help for the Tcl command for details.
	TCL_ERROR	1	ERROR: Illegal node generic ID: %u. Specify a legal node generic ID.
	TCL_ERROR	1	ERROR: Illegal node ID: %u. Specify a legal node ID.
	TCL_ERROR	1	ERROR: Illegal oport generic ID: %u. Specify a legal oport generic ID.
	TCL_ERROR	1	ERROR: The node you specified is a legalization node. Modification of legalization nodes is not supported.
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.7. get_node_by_name (::quartus::chip_planner)

The following table displays information for the `get_node_by_name` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_node_by_name [-h | -help] [-long_help] [-as_gen_id] -name` *<node name>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-as_gen_id`		Option to return result as generic id
	`-name` *<node name>*		Node name
Description	`Returns the node id of the specified node.` `Returns -1 if the node cannot be found.`		
Example Usage	`get_node_by_name -name 3`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.8. get_node_info (::quartus::chip_planner)

The following table displays information for the `get_node_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_node_info [-h | -help] [-long_help] [-gen_id` *<gen id>* `] -info` *<information type>* `[-node` *<node id>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-gen_id` *<gen id>*		Node generic id
	`-info` *<information type>*		Type of information
	`-node` *<node id>*		Node id
Description	`Returns the requested type of information for the specified node.` `To get available information types, use the "get_info_parameters" command.` `If the information type is legal for the specified node, the result is the requested information. Otherwise, the result is an empty string.`		
Example Usage	`get_node_info -node 3 -info name`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: Conflicting arguments. Consult help for the Tcl command for details.
	TCL_ERROR	1	ERROR: <*string*> value <*string*> is not valid for the specified node. Specify a legal value.
	TCL_ERROR	1	ERROR: Illegal node generic ID: %u. Specify a legal node generic ID.
	TCL_ERROR	1	ERROR: Illegal node ID: %u. Specify a legal node ID.
	TCL_ERROR	1	ERROR: The node you specified is a legalization node. Modification of legalization nodes is not supported.
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.9. get_nodes (::quartus::chip_planner)

The following table displays information for the `get_nodes` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_nodes [-h | -help] [-long_help] -type` <*all|lcell|io|pll|dsp|ram*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-type` <*all|lcell|io|pll|dsp|ram*>		Type of nodes to return
Description	`Returns a collection of nodes of the specified type.` `You can use the collection with the foreach_in_collection Tcl command.`		
Example Usage	`get_nodes -type all`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.10. get_oports (::quartus::chip_planner)

The following table displays information for the `get_oports` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_oports [-h | -help] [-long_help] [-as_gen_id] [-gen_id` <*gen id*> `] [-node` <*node id*> `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-as_gen_id`		Option to return results as generic id
	`-gen_id` <*gen id*>		Node generic id
	`-node` <*node id*>		Node id

continued...

Description	Returns a collection of output ports for the specified node. You can use the collection with the foreach_in_collection command.		
Example Usage	`get_oports -node 3`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Conflicting arguments. Consult help for the Tcl command for details.
	TCL_ERROR	1	ERROR: Illegal node generic ID: %u. Specify a legal node generic ID.
	TCL_ERROR	1	ERROR: Illegal node ID: %u. Specify a legal node ID.
	TCL_ERROR	1	ERROR: The node you specified is a legalization node. Modification of legalization nodes is not supported.
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.11. get_port_by_type (::quartus::chip_planner)

The following table displays information for the `get_port_by_type` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_port_by_type [-h | -help] [-long_help] [-as_gen_id] [-gen_id` *<gen id>* `] [-literal_index` *<literal index>* `] [-node` *<node id>* `] -port_type` *<port type>* `-type` *<iport|oport>*		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-as_gen_id`	Option to return result as generic ID	
	`-gen_id` *<gen id>*	Node generic id	
	`-literal_index` *<literal index>*	Literal index	
	`-node` *<node id>*	Node id	
	`-port_type` *<port type>*	Port type	
	`-type` *<iport|oport>*	Option to specify the port as an input or output port	
Description	Returns the port index for the specified port type on the specified node. Returns -1 if the port is not in use or is invalid for the specified node.		
Example Usage	`get_port_by_type -node 0 -port_type SLOAD -type iport` `get_port_by_type -node 0 -port_type EXTCLK -literal_index 2 -type oport`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: Unable to find port type: <*string*>. Specify a different port type.
	TCL_ERROR	1	ERROR: Conflicting arguments. Consult help for the Tcl command for details.
	TCL_ERROR	1	ERROR: Illegal node generic ID: %u. Specify a legal node generic ID.
	TCL_ERROR	1	ERROR: Illegal node ID: %u. Specify a legal node ID.
	TCL_ERROR	1	ERROR: Illegal port type: <*string*>. Specify a legal port type.
	TCL_ERROR	1	ERROR: The node you specified is a legalization node. Modification of legalization nodes is not supported.
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.12. get_port_info (::quartus::chip_planner)

The following table displays information for the `get_port_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_port_info [-h | -help] [-long_help] [-gen_id` <*gen id*> `] -info` <*information type*> `[-node` <*node id*> `] [-port_id` <*port id*> `] [-type` <*iport*|*oport*> `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-gen_id` <*gen id*>		Port generic ID
	`-info` <*information type*>		Type of information
	`-node` <*node id*>		Node ID
	`-port_id` <*port id*>		Port ID
	`-type` <*iport*|*oport*>		Option to specify the port as an input or output port
Description	`Returns the requested type of information for the specified port.` `To get available information types, use the get_info_parameters command.` `If the information type is legal for the specified port, the result is the requested information. Otherwise, the result is an empty string.`		
Example Usage	`get_port_info -node 3 -port_id 2 -type iport -info port_name`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unable to find port ID: %u. Specify a different port ID.
	TCL_ERROR	1	ERROR: Conflicting arguments. Consult help for the Tcl command for details.
	TCL_ERROR	1	ERROR: <*string*> value <*string*> is not valid for the specified port. Specify a legal value.

continued...

	TCL_ERROR	1	ERROR: Illegal node ID: %u. Specify a legal node ID.
	TCL_ERROR	1	ERROR: Illegal port generic ID: %u. Specify a legal port generic ID.
	TCL_ERROR	1	ERROR: The node you specified is a legalization node. Modification of legalization nodes is not supported.
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.13. get_sp_pin_list (::quartus::chip_planner)

The following table displays information for the `get_sp_pin_list` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_sp_pin_list [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns a list of the pins availible for use as signal probe output pins.;`		
Example Usage	`get_sp_pin_list`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.4.14. get_tile_power_setting (::quartus::chip_planner)

The following table displays information for the `get_tile_power_setting` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`get_tile_power_setting [-h | -help] [-long_help] [-X <X location> ] [-Y <Y location> ] [-gen_id <gen id> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-X <X location>`		X location
	`-Y <Y location>`		Y location
	`-gen_id <gen id>`		Generic id
Description	`Returns the High-Speed/Low Power setting of the tile at the specified location.`		
Example Usage	`get_tile_power_setting -gen_id 12345` `get_tile_power_setting -X 12 -Y 5`		
Return Value	**Code Name**	**Code**	**String Return**

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Conflicting arguments. Consult help for the Tcl command for details.
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.4.15. read_netlist (::quartus::chip_planner)

The following table displays information for the `read_netlist` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110		
Syntax	`read_netlist [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Reads the Chip Planner netlist from the last compilation.` `You must open a project before using this command.`		
Example Usage	`read_netlist`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unable to create Chip Planner netlist. Current device family does not support the Chip Planner. Specify another device family and recompile the design.
	TCL_ERROR	1	ERROR: Chip Planner (::quartus::chip_planner) is not available from the Quartus Prime Tcl Console. Run the quartus_cdb executable with commands from the ::quartus::chip_planner package from a system command prompt.
	TCL_ERROR	1	ERROR: Chip Planner is unavailable with the current license. Refer to the Licensing section of the Intel website to obtain a valid Quartus Prime license file.
	TCL_ERROR	1	ERROR: Unable to find an active revision. Make sure there is an open, active revision.
	TCL_ERROR	1	ERROR: No open project. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: Before running Chip Planner, run Analysis & Synthesis (quartus_map) for read-only use and quartus_fit to enable writable ECO changes.

3.1.4.16. set_batch_mode (::quartus::chip_planner)

The following table displays information for the `set_batch_mode` Tcl command:

Tcl Package and Version	Belongs to ::quartus::chip_planner on page 110	
Syntax	`set_batch_mode [-h | -help] [-long_help] <`*on*`|`*off*`>`	
Arguments	`-h | -help`	Short help

continued...

	-long_help		Long help with examples and possible return values
	<on|off>		Option to turn batch mode on or off
Description	Sets the batch mode to On or Off.		
Example Usage	set_batch_mode on		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unable to find Chip Planner netlist. Read the netlist by using the read_netlist command.

3.1.5. ::quartus::dcmd_dni

The following table displays information for the **::quartus::dcmd_dni** Tcl package:

Tcl Package and Version	::quartus::dcmd_dni 1.0
Description	This package contains no general description.
Availability	This package is available for loading in the following executables: quartus quartus_sh quartus_sta quartus_syn
Tcl Commands	dni::add_to_collection (::quartus::dcmd_dni) on page 121 dni::all_clocks (::quartus::dcmd_dni) on page 121 dni::all_fanin (::quartus::dcmd_dni) on page 121 dni::all_fanout (::quartus::dcmd_dni) on page 122 dni::all_inputs (::quartus::dcmd_dni) on page 122 dni::all_outputs (::quartus::dcmd_dni) on page 123 dni::all_registers (::quartus::dcmd_dni) on page 123 dni::append_to_collection (::quartus::dcmd_dni) on page 124 dni::color (::quartus::dcmd_dni) on page 124 dni::copy_collection (::quartus::dcmd_dni) on page 125 dni::create_clock (::quartus::dcmd_dni) on page 125 dni::current_design (::quartus::dcmd_dni) on page 126 dni::current_instance (::quartus::dcmd_dni) on page 126 dni::delete_stale_sandboxes (::quartus::dcmd_dni) on page 127 dni::filter_collection (::quartus::dcmd_dni) on page 127 dni::get_cells (::quartus::dcmd_dni) on page 128 dni::get_clocks (::quartus::dcmd_dni) on page 129 dni::get_designs (::quartus::dcmd_dni) on page 129 dni::get_generated_clocks (::quartus::dcmd_dni) on page 130 dni::get_nets (::quartus::dcmd_dni) on page 131 dni::get_pins (::quartus::dcmd_dni) on page 131 dni::get_ports (::quartus::dcmd_dni) on page 132 dni::get_property (::quartus::dcmd_dni) on page 133 dni::highlight (::quartus::dcmd_dni) on page 133 dni::index_collection (::quartus::dcmd_dni) on page 134 dni::is_dni_mode (::quartus::dcmd_dni) on page 134 dni::is_dni_mode_for_developer_testing (::quartus::dcmd_dni) on page 135 dni::list_properties (::quartus::dcmd_dni) on page 135 dni::load_design (::quartus::dcmd_dni) on page 135 dni::read_sdc (::quartus::dcmd_dni) on page 136 dni::remove_from_collection (::quartus::dcmd_dni) on page 136 dni::selection (::quartus::dcmd_dni) on page 137 dni::set_property (::quartus::dcmd_dni) on page 137 dni::set_time_format (::quartus::dcmd_dni) on page 138 dni::set_time_unit (::quartus::dcmd_dni) on page 138 dni::sizeof_collection (::quartus::dcmd_dni) on page 139 dni::sort_collection (::quartus::dcmd_dni) on page 139 dni::unload_design (::quartus::dcmd_dni) on page 140 dni::write_sdc (::quartus::dcmd_dni) on page 140

3.1.5.1. dni::add_to_collection (::quartus::dcmd_dni)

The following table displays information for the `dni::add_to_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::add_to_collection [-h | -help] [-long_help] [-unique] <collection> <object_spec>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-unique`		Only add non duplicated objects to the collection
	<collection>		The base collection
	<object_spec>		A list of objects to be added to collection
Description	`Add objects to a collection, resulting in a new collection. The base collection remains unchanged.`		
Example Usage	`::dni::add_to_collection $collection [list $o1 $o2]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.2. dni::all_clocks (::quartus::dcmd_dni)

The following table displays information for the `dni::all_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::all_clocks [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`This command creates a collection of all clocks in the design.`		
Example Usage	`dni::all_clocks`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.3. dni::all_fanin (::quartus::dcmd_dni)

The following table displays information for the `dni::all_fanin` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::all_fanin [-h | -help] [-long_help] [-flat] [-insts_only] [-startpoints_only] -to <to>`	
Arguments	`-h | -help`	Short help

continued...

	-long_help		Long help with examples and possible return values
	-flat		trace across the hierarchies
	-insts_only		returns a set of all instances in the timing fanin of the sink
	-startpoints_only		returns only the timing startpoints
	-to <*to*>		sink port or inst_port in the design
Description	This command reports the timing fanin of specified sink ports or inst_ports in the design		
Example Usage	dni::all_fanin -to [dni::get_ports {a[0]}]		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.4. dni::all_fanout (::quartus::dcmd_dni)

The following table displays information for the dni::all_fanout Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	dni::all_fanout [-h | -help] [-long_help] [-endpoints_only] [-flat] -from <*from*> [-insts_only]		
Arguments	-h | -help		Short help
	-long_help		Long help with examples and possible return values
	-endpoints_only		returns only the timing endpoints
	-flat		trace across the hierarchies
	-from <*from*>		source port or inst_port in the design
	-insts_only		returns a set of all instances in the timing fanout of the source
Description	This command reports the timing fanout of specified source ports or inst_ports in the design		
Example Usage	dni::all_fanout -from [dni::get_ports {a[0]}]		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.5. dni::all_inputs (::quartus::dcmd_dni)

The following table displays information for the dni::all_inputs Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	dni::all_inputs [-h | -help] [-long_help]		
Arguments	-h | -help		Short help
	-long_help		Long help with examples and possible return values

continued...

Description	This command creates a collection of all input ports in the current design.		
Example Usage	`# List all input ports from top module in current design.` `dni::all_inputs`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.6. dni::all_outputs (::quartus::dcmd_dni)

The following table displays information for the `dni::all_outputs` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::all_outputs [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	This command creates a collection of all output ports in the current design.		
Example Usage	`# List all output ports from top module in current design.` `dni::all_outputs`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.7. dni::all_registers (::quartus::dcmd_dni)

The following table displays information for the `dni::all_registers` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::all_registers [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	This command creates a collection of all register cells		
Example Usage	`dni::all_registers`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.8. dni::append_to_collection (::quartus::dcmd_dni)

The following table displays information for the `dni::append_to_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::append_to_collection [-h | -help] [-long_help] [-unique] <collection> <object_spec>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-unique`	Only append non duplicated objects to the collection	
	<collection>	TCL variable name pointing to a collection	
	<object_spec>	A list of objects to be appended to collection	
Description	`Append objects to a collection and modifies a variable.`		
Example Usage	`::dni::append_to_collection collection [list $o1 $o2]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.9. dni::color (::quartus::dcmd_dni)

The following table displays information for the `dni::color` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::color [-h | -help] [-long_help] -action <action> [-append] [-checkpoint <checkpoint name> ] [-file <file> ] [-name <name> ] [-object <object> ] [-set <set> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-action <action>`	color action	
	`-append`	to append to existing file or overwrite	
	`-checkpoint <checkpoint name>`	checkpoint name	
	`-file <file>`	file name to save or restore color sets	
	`-name <name>`	name for the colored set	
	`-object <object>`	object to be colored	
	`-set <set>`	a set containing colored objects	
Description	`Work with object color set.`		
Example Usage	`dni::color -action $action`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.10. dni::copy_collection (::quartus::dcmd_dni)

The following table displays information for the `dni::copy_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::copy_collection [-h | -help] [-long_help] <collection>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<collection>	The collection to be copied	
Description	`Duplicates the contents of a collection, resulting in a new collection. The base collection remains unchanged.`		
Example Usage	`set copied_collection [::dni::copy_collection $collection]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.11. dni::create_clock (::quartus::dcmd_dni)

The following table displays information for the `dni::create_clock` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::create_clock [-h | -help] [-long_help] [-add] [-comment <comment> ] [-name <name> ] -period <period> [-waveform <waveform> ] [ <targets> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add`	whether to add
	`-comment <comment>`	comment for creating a clock
	`-name <name>`	name for creating a clock
	`-period <period>`	period for creating a clock
	`-waveform <waveform>`	waveform for creating a clock
	<targets>	target objects for creating a clock
Description	`Defines a clock. If the -name option is not used, the clock name is the same as the first target in the list or collection. The clock name is used to refer to the clock in other commands.` `The -period option specifies the clock period. It is also possible to use this option to specify a frequency to define the clock period. This can be done by using -period option followed by either <frequency>MHz or "<frequency> MHz". Please note this is a convenience extension and using it is non-standard SDC syntax.`  `The -waveform option specifies the rising and falling edges (duty cycle) of the clock, and is specified as a list of two time values: the first rising edge and the next falling edge. The rising edge must be within the range [0, period]. The falling edge must be within one clock period of the rising edge. The waveform defaults to (0, period/2).`	

continued...

	If a clock with the same name is already assigned to a given target, the create_clock command will return an error. If a clock with a different name exists on the given target, the create_clock command will be ignored unless the -add option is used. The -add option can be used to assign multiple clocks to a pin or port. If the target of the clock is internal (i.e. not an input port), the source latency is zero by default. If a clock is on a path after another clock, then it blocks or overwrites the previous clock from that point forward. The value of the targets is either a collection or a Tcl list of wildcards used to create a collection of the appropriate type. The values used must follow standard Tcl substitution rules.		
Example Usage	`dni::create_clock $period`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.12. dni::current_design (::quartus::dcmd_dni)

The following table displays information for the `dni::current_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::current_design [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	This command returns the current top module of the design. In Quartus, top module cannot be changed.		
Example Usage	`dni::current_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.13. dni::current_instance (::quartus::dcmd_dni)

The following table displays information for the `dni::current_instance` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::current_instance [-h | -help] [-long_help] [ <instance> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<instance>	An instance name relative to the current instance	
Description	This command sets the instance as current point of reference for object names (e.g. relative path names) used in object query commands. This command returns the new current instance hierarchy or empty if current is design top.		

continued...

	If no instance name specified, the current instance is set to the current design top. If instance name specified as ".", the current instance is not changed. If instance name specified as "..", move current instance one level up in the hierarchy. The instance name can include any number of ".." (separated by hierarchy separators '|') as well as hierarchy instance names. The new current instance cannot be a leaf cell.
Example Usage	`move current instance up two levels` `dni::current_instance ..|..` `# move current instance to foo under parent of current instance` `dni::current_instance ..|foo` `# Save the current instance` `set saved_ci [dni::current_instance .]` `# make the design top the new current instance` `dni::current_instance` `# query top-level objects` `set top_level_foo [dni::get_cells foo]` `# restore saved current instance` `current_instance $saved_ci`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.14. dni::delete_stale_sandboxes (::quartus::dcmd_dni)

The following table displays information for the `dni::delete_stale_sandboxes` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::delete_stale_sandboxes [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Removes stale sandbox directories on disk.	
Example Usage	`project_open <revision name>` `dni::delete_stale_sandboxes` `project_close`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.15. dni::filter_collection (::quartus::dcmd_dni)

The following table displays information for the `dni::filter_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120
Syntax	`dni::filter_collection [-h | -help] [-long_help] [-nocase] [-regexp]` *<collection> <filter>*

continued...

Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-nocase`		Whether to perform case insensitive filter
	`-regexp`		Whether to apply regular expression when to filter
	<collection>		The base collection
	<filter>		Filter expression
Description			
	`Filters an existing collection, resulting in a new collection. The base collection remains unchanged.`		
Example Usage	`set filtered_collection [::dni::filter_collection $collection "name==\"speed\""]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.16. dni::get_cells (::quartus::dcmd_dni)

The following table displays information for the `dni::get_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::get_cells [-h | -help] [-long_help] [-exact] [-filter` *<expression>* `] [-hierarchical] [-nocase] [-of_objects` *<objects>* `] [-quiet] [` *<patterns>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exact`	exact patterns to search for object. Exact patterns cannot include wildcard characters "*" or "?" and escaped characters "\".
	`-filter` *<expression>*	filter search by expression
	`-hierarchical`	search level-by-level down from current instance
	`-nocase`	perform case insensitive search
	`-of_objects` *<objects>*	search based on relationship to objects
	`-quiet`	Suppress error, warning, or information messages
	<patterns>	patterns to search for object. Patterns can include wildcard characters "*" or "?". Wildcard characters do not match with hierarchy separator
Description		
	`This command creates a collection of instances from the current design, relative to the current instance.`	
Example Usage	`# Suppose a design from top contains two hierchical instances.` `#  instance name (module name)` `# |  (top)` `# |-h1_i1  (mod_A)` `# | |-h2_i1  (mod_B)` `# |` `# |-h1_i2  (mod_A)` `#  |-h2_i1  (mod_B)` `# Get instances from top module: {h1_1 h1_2}` `dni::get_cells`	

continued...

	`` # Get hierarchical instances with prefix h from top module: { h1_i1 h1_i1|h2_i1 h1_i2 h1_i2| h2_i1 } dni::get_cells h* -hierarchical # Get hierarchical instances with prefix h from |h1_i2 instance: { h1_i2|h2_i1 } dni::current_instance h1_i2 dni::get_cells h* -hierarchical # Get instance of instance port dni::get_cells -of_objects [dni::get_pins h1_i2|in1] ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.17. dni::get_clocks (::quartus::dcmd_dni)

The following table displays information for the `dni::get_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::get_clocks [-h | -help] [-long_help] [-filter` *<filter>* `] [-nocase] [-quiet] [-regexp] [` *<patterns>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-filter` *<filter>*	filter for getting clocks
	`-nocase`	whether to apply case insensitive on filter for getting clocks
	`-quiet`	quiet mode for getting clocks
	`-regexp`	whether to use regexp on filter for getting clocks
	<patterns>	patterns for getting clocks
Description	``` This command creates a collection of clocks from clocks that are currently defined. ```	
Example Usage	``` dni::get_clocks ```	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.18. dni::get_designs (::quartus::dcmd_dni)

The following table displays information for the `dni::get_designs` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::get_designs [-h | -help] [-long_help] [-exact` *<exact>* `] [-filter` *<expression>* `] [-nocase] [-quiet] [` *<patterns>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exact` *<exact>*	exact patterns to search for object. Exact patterns cannot include wildcard characters "*" or "?" and escaped characters "\".

continued...

	`-filter` *<expression>*		filter search by expression
	`-nocase`		perform case insensitive search
	`-quiet`		Suppress error, warning, or information messages
	<patterns>		patterns to search for object. Patterns can include wildcard characters "*" or "?". Wildcard characters do not match with hierarchy separator
Description	`Creates a collection of one or more modules in the design.`		
Example Usage	`# Create a collection of all modules` `dni::get_designs` `# Lookup by prefix name` `dni::get_designs auto*`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.19. dni::get_generated_clocks (::quartus::dcmd_dni)

The following table displays information for the `dni::get_generated_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::get_generated_clocks [-h | -help] [-long_help] [-exact] [-filter` *<filter>* `] [-nocase] [-quiet] [-regexp] [` *<patterns>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-exact`		whether to match filter exactly for getting generated clocks
	`-filter` *<filter>*		filter for getting generated clocks
	`-nocase`		whether to apply case insensitive on filter for getting generated clocks
	`-quiet`		quiet mode for getting generated clocks
	`-regexp`		whether to use regexp on filter for getting generated clocks
	<patterns>		patterns for getting generated clocks
Description	`This command creates a collection of all generated clocks in the design.`		
Example Usage	`dni::get_generated_clocks`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.20. dni::get_nets (::quartus::dcmd_dni)

The following table displays information for the `dni::get_nets` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::get_nets [-h | -help] [-long_help] [-exact] [-filter <expression> ] [-hierarchical] [-nocase] [-of_objects <objects> ] [-quiet] [ <patterns> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-exact`	exact patterns to search for object. Exact patterns cannot include wildcard characters "*" or "?" and escaped characters "\".	
	`-filter <expression>`	filter search by expression	
	`-hierarchical`	search level-by-level down from current instance	
	`-nocase`	perform case insensitive search	
	`-of_objects <objects>`	search based on relationship to objects	
	`-quiet`	Suppress error, warning, or information messages	
	<patterns>	patterns to search for object. Patterns can include wildcard characters "*" or "?". Wildcard characters do not match with hierarchy separator	
Description	`This command creates a collection of nets from the current design, relative to the current instance.`		
Example Usage	`# Creates a collection of nets at hierarchy h1* from top module.` `dni::get_nets h1*|*` `# Creates a collection of nets connected to instance port h1_i1|in?.` `dni::get_nets -of_objects [dni::get_pins h1_i1|in?]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.21. dni::get_pins (::quartus::dcmd_dni)

The following table displays information for the `dni::get_pins` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::get_pins [-h | -help] [-long_help] [-exact] [-filter <expression> ] [-hierarchical] [-nocase] [-of_objects <objects> ] [-quiet] [ <patterns> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exact`	exact patterns to search for object. Exact patterns cannot include wildcard characters "*" or "?" and escaped characters "\".
	`-filter <expression>`	filter search by expression
	`-hierarchical`	search level-by-level down from current instance

continued...

	-nocase		perform case insensitive search
	-of_objects <*objects*>		search based on relationship to objects
	-quiet		Suppress error, warning, or information messages
	<*patterns*>		patterns to search for object. Patterns can include wildcard characters "*" or "?". Wildcard characters do not match with hierarchy separator
Description	This command creates a collection of instance ports from given search specifications.		
Example Usage	# Creates a collection of instance ports with prefix out from top module. dni::get_pins out* # Creates a collection of instance ports connected to net h1_i1|n1. dni::get_pins -of_objects [dni::get_nets h1_i1|n1] # Creates a collection of instance ports from hierarchy h1* with direction as input and name prefix. dni::get_pins h1*|* -filter {direction==input && name=~in*}]		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.22. dni::get_ports (::quartus::dcmd_dni)

The following table displays information for the `dni::get_ports` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	dni::get_ports [-h | -help] [-long_help] [-exact] [-filter <*expression*>] [-hierarchical] [-nocase] [-of_objects <*objects*>] [-quiet] [<*patterns*>]	
Arguments	-h | -help	Short help
	-long_help	Long help with examples and possible return values
	-exact	exact patterns to search for object. Exact patterns cannot include wildcard characters "*" or "?" and escaped characters "\".
	-filter <*expression*>	filter search by expression
	-hierarchical	search level-by-level down from current instance
	-nocase	perform case insensitive search
	-of_objects <*objects*>	search based on relationship to objects
	-quiet	Suppress error, warning, or information messages
	<*patterns*>	patterns to search for object. Patterns can include wildcard characters "*" or "?". Wildcard characters do not match with hierarchy separator
Description	This command creates a collection of ports from the current design, relative to the current instance.	
Example Usage	# Creates a collection of ports from top module. dni::get_ports # Creates a collection of ports filter by output direction. dni::get_ports -filter {direction==output}	

continued...

	`# Creates a collection of ports connected to net n1.` `dni::get_ports -of_objects [dni::get_nets n1]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.23. dni::get_property (::quartus::dcmd_dni)

The following table displays information for the `dni::get_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::get_property [-h | -help] [-long_help] -name <name> -object <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	property name
	`-object <object>`	object from which to get property
Description	`Command to get property.`	
Example Usage	`set mod_port_name [dni::get_property -object $mod_port -name name]`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.24. dni::highlight (::quartus::dcmd_dni)

The following table displays information for the `dni::highlight` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::highlight [-h | -help] [-long_help] -action <action> [-append] [-file <file> ] [-name <name> ] [-object <object> ] [-set <set> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-action <action>`	highlight action
	`-append`	to append to existing file or overwrite
	`-file <file>`	file name to save or restore highlight sets
	`-name <name>`	name for the highlight set
	`-object <object>`	object to be highlighted
	`-set <set>`	a set containing highlight objects
Description	`Command for object highlight.`	

continued...

Example Usage	`dni::highlight -action find`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.25. dni::index_collection (::quartus::dcmd_dni)

The following table displays information for the `dni::index_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::index_collection [-h | -help] [-long_help]` *<collection> <index> <index2>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<collection>	The base collection
	<index>	The index to locate an object in the collection
	<index2>	The second index to locate an item in the collection. If present, a list of objects between index and index2 in the base collection will be used to create a new collection
Description	`Given a collection and an index, if the index is in range, create a new collection containing only the single object. Optionally a second index can be passed to create a new collection with the objects between the two indices in the base collection.`	
Example Usage	`set new_collection [::dni::index_collection $collection 0 10]`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.26. dni::is_dni_mode (::quartus::dcmd_dni)

The following table displays information for the `dni::is_dni_mode` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::is_dni_mode [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Command to check if it is in DNI mode.`	
Example Usage	`dni::is_dni_mode`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.27. dni::is_dni_mode_for_developer_testing (::quartus::dcmd_dni)

The following table displays information for the `dni::is_dni_mode_for_developer_testing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::is_dni_mode_for_developer_testing [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Command to check if it is in DNI mode for developer testing.`		
Example Usage	`dni::is_dni_mode_for_developer_testing`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.28. dni::list_properties (::quartus::dcmd_dni)

The following table displays information for the `dni::list_properties` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::list_properties [-h | -help] [-long_help] -object <object> -type <type>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-object <object>`		object from which to list properties
	`-type <type>`		type from which to list properties
Description	`This command gets the list of property names of an object or a specific object type.`		
Example Usage	`set prop_list [dni::list_properties -type $type -object $object]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.29. dni::load_design (::quartus::dcmd_dni)

The following table displays information for the `dni::load_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::load_design [-h | -help] [-long_help] -checkpoint <checkpoint name> [-mode <mode> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values

continued...

	`-checkpoint` *<checkpoint name>*		checkpoint name
	`-mode` *<mode>*		Mode for open design
Description	`Open the specified checkpoint into a sandbox and load the design from the sandbox.` `The opened design becomes the default design used for Tcl commands query.`		
Example Usage	`dni::load_design -checkpoint "elaborated"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.30. dni::read_sdc (::quartus::dcmd_dni)

The following table displays information for the `dni::read_sdc` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::read_sdc [-h | -help] [-long_help] [-replace]` *<sdc_file>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-replace`	Option to override existing constraints
	<sdc_file>	Sdc filename
Description	`Command to source constraints from sdc file.`	
Example Usage	`dni::read_sdc top.sdc`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.31. dni::remove_from_collection (::quartus::dcmd_dni)

The following table displays information for the `dni::remove_from_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::remove_from_collection [-h | -help] [-long_help] [-intersect]` *<collection>* *<object_spec>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-intersect`	Removes objects from collection1 not found in object_spec. Without this option, removes objects from collection1 that are found in object_spec.
	<collection>	The base collection
	<object_spec>	A list of objects to be removed from the base collection

continued...

Description	Remove objects from a collection, resulting in a new collection. The base collection remains unchanged.		
Example Usage	`set new_collection [::dni::remove_from_collection $collection [list $o1 $o2]]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.32. dni::selection (::quartus::dcmd_dni)

The following table displays information for the `dni::selection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::selection [-h | -help] [-long_help] -action <action> [-object <object> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-action <action>`		selection action
	`-object <object>`		object to be selected
Description	Command for object selection.		
Example Usage	`dni::selection -action clear`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.33. dni::set_property (::quartus::dcmd_dni)

The following table displays information for the `dni::set_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::set_property [-h | -help] [-long_help] -name <name> -object <object> -value <value>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		property name to set property
	`-object <object>`		object from which to set property
	`-value <value>`		property value to set property
Description	Command to set property.		
Example Usage	`dni::set_property -object $mod_port -name name -value {new_name}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.34. dni::set_time_format (::quartus::dcmd_dni)

The following table displays information for the `dni::set_time_format` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::set_time_format [-h | -help] [-long_help] [-decimal_places` *<decimal_places>* `] [-unit` *<unit>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-decimal_places` *<decimal_places>*	Number of decimal places to use
	`-unit` *<unit>*	Default time unit to use
Description	`Sets time format, including time unit and decimal places.` `Time units are assumed to be nanoseconds (ns) by default. The "-unit" option overrides the default time units. Legal time unit values are: ps, ns, us, ms.` `Time units are displayed with three decimal places by default. The "-decimal_places" option overrides the default number of decimal places to show.` `The smallest resolution of all times units is one picosecond (ps). Any additional specified precision will be truncated.`	
Example Usage	`# Create two clocks with a clock period of 8 nanoseconds.` `create_clock -period 8.000 clk1` `set_time_format -unit ps -decimal_places 0` `create_clock -period 8000 clk2`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified number of decimal places is invalid. Please specify an integral value >= 0.
	TCL_ERROR	1	ERROR: The default time unit can be set to ms, us, ns, or ps. Please specify one of these units instead.

3.1.5.35. dni::set_time_unit (::quartus::dcmd_dni)

The following table displays information for the `dni::set_time_unit` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::set_time_unit [-h | -help] [-long_help]` *<unit>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<unit>	Default time unit to use
Description	`Time units are assumed to be nanoseconds (ns) by default unless otherwise specified. Time or delay values are also displayed in nanoseconds by default without time units. The dni::set_time_unit COMMAND overrides the default time units assumed by the Timing Analyzer.`	

continued...

	Legal values are: ps, ns, us, ms The smallest resolution of all times units is one picosecond (ps). Any additional specified precision will be truncated.		
Example Usage	# Create two clocks with a clock period of 8 nanoseconds. create_clock -period 8.000 clk1 dni::set_time_unit ps create_clock -period 8000 clk2		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The default time unit can be set to ms, us, ns, or ps. Please specify one of these units instead.

3.1.5.36. dni::sizeof_collection (::quartus::dcmd_dni)

The following table displays information for the `dni::sizeof_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::sizeof_collection [-h | -help] [-long_help] [-categorize] <collection>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-categorize`	Return 0, 1, 2 indicating if the collection has 1, or or more than 1 item
	<collection>	The base collection
Description	Returns the number of objects in a collection.	
Example Usage	set size [::dni::sizeof_collection $collection]	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.37. dni::sort_collection (::quartus::dcmd_dni)

The following table displays information for the `dni::sort_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::sort_collection [-h | -help] [-long_help] [-descending] [-dictionary] [-limit` *<limit>* `] <collection> <criteria>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-descending`	Indicates that the collection is to be sorted in reverse order. By default, the sort proceeds in ascending order

continued...

	`-dictionary`		Sort strings dictionary order. For example "a30" would come after "a4"
	`-limit` *<limit>*		Only return the first unique values from the primary sort key
	<collection>		The base collection
	<criteria>		Specifies a list of one or more application or user-defined attributes to use as sort keys
Description			
	Sorts a collection based on one or more attributes, resulting in a new, sorted collection. The sort is ascending by default.		
Example Usage	`set size [::dni::sort_collection $collection {is_hierarchy}]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.38. dni::unload_design (::quartus::dcmd_dni)

The following table displays information for the `dni::unload_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120		
Syntax	`dni::unload_design [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description			
	Closes the design of the current opened sandbox and closes the sandbox.		
Example Usage	`dni::load_design -checkpoint "elaborated"` `...` `dni::unload_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.5.39. dni::write_sdc (::quartus::dcmd_dni)

The following table displays information for the `dni::write_sdc` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dcmd_dni on page 120	
Syntax	`dni::write_sdc [-h | -help] [-long_help] [-command` *<command>* `] [-design` *<design>* `] [-file` *<file>* `] [-include_file_info] [-line` *<line>* `] [-output_file` *<output_file>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-command` *<command>*	Only output constraint statements defined with this command
	`-design` *<design>*	Design for CDMS design related commands

continued...

	`-file <file>`	Only output constraint statements defined in this file
	`-include_file_info`	Include the file name and line number where the constraint is defined
	`-line <line>`	Only output constraint statements defined on this line number
	`-output_file <output_file>`	Path to output file where constraint statements are written
Description	`Write the SDC constraint statements currently loaded for this design.`	
Example Usage	`# Write the SDC constraints to the console` `dni::write_sdc` `# Write the SDC constraints to a file` `dni::write_sdc -output_file write_sdc_output.txt`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6. ::quartus::design

The following table displays information for the **::quartus::design** Tcl package:

Tcl Package and Version	::quartus::design 1.0
Description	`This package contains the set of Tcl functions for manipulating databases including the assignments database. Using this package makes it possible to create instance assignments without modifying the Quartus Prime Settings File (.qsf).`
Availability	`This package is loaded by default in the following executable:` `quartus_cdb` `This package is available for loading in the following executables:` `qacv` `qpro` `qpro_sh` `quartus` `quartus_fit` `quartus_map` `quartus_pow` `quartus_sh` `quartus_sta` `quartus_syn`
Tcl Commands	`design::commit_design (::quartus::design)` on page 142 `design::convert_partition (::quartus::design)` on page 142 `design::create_assignment (::quartus::design)` on page 143 `design::delete_assignments (::quartus::design)` on page 143 `design::disable_assignments (::quartus::design)` on page 144 `design::enable_assignments (::quartus::design)` on page 145 `design::export_design (::quartus::design)` on page 145 `design::export_partition (::quartus::design)` on page 146 `design::extract_metadata (::quartus::design)` on page 147 `design::get_assignment_info (::quartus::design)` on page 147 `design::get_assignment_names (::quartus::design)` on page 148 `design::get_assignments (::quartus::design)` on page 148 `design::get_entity_names (::quartus::design)` on page 149 `design::get_instances (::quartus::design)` on page 150 `design::import_design (::quartus::design)` on page 150 `design::import_partition (::quartus::design)` on page 151 `design::list_valid_snapshot_names (::quartus::design)` on page 152 `design::load_design (::quartus::design)` on page 152 `design::report_assignments (::quartus::design)` on page 153 `design::set_assignment_info (::quartus::design)` on page 153

3.1.6.1. design::commit_design (::quartus::design)

The following table displays information for the `design::commit_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::commit_design [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Commit any changes to the databases to disk.` `Assignments created or modified on a design loaded as writeable are not saved to the databases unless you explicitally call this command.`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot -writeable` `design::delete_assignments [design::get_assignments -name location]` `design::commit_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6.2. design::convert_partition (::quartus::design)

The following table displays information for the `design::convert_partition` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::convert_partition [-h | -help] [-long_help] -infile` *<QDB file name>* `-outfile` *<QDB file name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-infile` *<QDB file name>*	Input file name of the QDB archive (version-compatible format).
	`-outfile` *<QDB file name>*	Output file name of the QDB archive (current-version-only format).
Description	`Convert a partition's QDB file in version-compatible ASCII format into a QDB file in BINARY format for current version of Quartus.`	
Example Usage	`# The input QDB file is created by running design::export_partition` `# from compiled source design with Quartus of older or current version` `project_open onewire_nf` `design::export_partition core_ptn -snapshot synthesized -file src_ip.qdb -compatible` `project_close` `# Make sure you are using the same version of Quartus that will use to compile your design.` `project_open onewire_nf` `design::convert_partition -infile src_ip.qdb -outfile ip.qdb` `project_close`	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified archive <*string*> does not exist.

3.1.6.3. design::create_assignment (::quartus::design)

The following table displays information for the `design::create_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::create_assignment [-h | -help] [-long_help] [-from <from> ] -name <name> -to <to> -value <value>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-from <from>`		The source name of the assignment
	`-name <name>`		The type name of the assignment
	`-to <to>`		The destination name of the assignment
	`-value <value>`		The value of the assignment
Description	`Create a new assignment in the assignment database`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot -writeable` `design::create_assignment -name location -to in1 -value PIN_A5` `design::commit_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Both the -to or -from argument is required.
	TCL_ERROR	1	ERROR: Either the -to or -from argument is required.
	TCL_ERROR	1	ERROR: The -to argument is required.
	TCL_ERROR	1	ERROR: The value of an assignment cannot be empty.

3.1.6.4. design::delete_assignments (::quartus::design)

The following table displays information for the `design::delete_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::delete_assignments [-h | -help] [-long_help] <assignment>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*assignment*>	one or more assignment ids

continued...

Description	`Delete one or more assignments from the assignment database`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot -writeable` `design::delete_assignments [design::get_assignments -name location]` `design::commit_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied assignment id <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one periphery assignment ID must be supplied, but no assignments IDs were supplied.
	TCL_ERROR	1	ERROR: <*string*> assignment IDs were expected but <*string*> were supplied.

3.1.6.5. design::disable_assignments (::quartus::design)

The following table displays information for the `design::disable_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::disable_assignments [-h | -help] [-long_help] <assignment>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<*assignment*>		one or more assignment ids
Description	`Disables one or more assignments from the assignment database`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot -writeable` `design::disable_assignments [design::get_assignments -name location]` `design::commit_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied assignment id <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one periphery assignment ID must be supplied, but no assignments IDs were supplied.
	TCL_ERROR	1	ERROR: <*string*> assignment IDs were expected but <*string*> were supplied.

3.1.6.6. design::enable_assignments (::quartus::design)

The following table displays information for the `design::enable_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::enable_assignments [-h | -help] [-long_help] <assignment>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<assignment>	one or more assignment ids	
Description	`Enables one or more assignments from the assignment database that were previously disabled`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot -writeable` `design::enable_assignments [design::get_assignments -name location]` `design::commit_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied assignment id *<string>* is invalid.
	TCL_ERROR	1	ERROR: At least one periphery assignment ID must be supplied, but no assignments IDs were supplied.
	TCL_ERROR	1	ERROR: *<string>* assignment IDs were expected but *<string>* were supplied.

3.1.6.7. design::export_design (::quartus::design)

The following table displays information for the `design::export_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::export_design [-h | -help] [-long_help] -file <file> [-quartus_metadata <quartus_metadata> ] -snapshot <snapshot> [-user_metadata <user_metadata> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-file <file>`	The file.qdb to export to
	`-quartus_metadata <quartus_metadata>`	A space-separated list of Quartus Metadata to export. Valid Quartus Metadata options include <none| project_information|resource_utilization|all>.
	`-snapshot <snapshot>`	The snapshot you want to export. Valid snapshot <synthesized|final>.
	`-user_metadata <user_metadata>`	The absolute or relative path to the User Metadata configuration file.

continued...

Description	Export the specified metadata and loaded databases for the open project and revision and snapshot to <file>.qdb in a version-compatable format. This command is available only in the quartus_cdb executable.		
Example Usage	project_open onewire_nf design::export_design -file onewire.qdb -snapshot synthesized project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6.8. design::export_partition (::quartus::design)

The following table displays information for the `design::export_partition` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::export_partition [-h | -help] [-long_help] [-exclude_pr_subblocks] -file` *<QDB file name>* `[-include_sdc_entity_in_partition] [-preserve_sdc] [-quartus_metadata` *<quartus_metadata>* `] -snapshot` *<Snapshot(s) to be exported>* `[-user_metadata` *<user_metadata>* `]` *<Partition to be exported>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exclude_pr_subblocks`	Exclude PR subpartitions
	`-file` *<QDB file name>*	File name of the QDB archive.
	`-include_sdc_entity_in_partition`	Preserve SDC/TCL Entity files
	`-preserve_sdc`	Deprecated option to Preserve SDC/TCL Entity files
	`-quartus_metadata` *<quartus_metadata>*	A space-separated list of Quartus Metadata to export. Valid Quartus Metadata options include <none|project_information|resource_utilization|all>.
	`-snapshot` *<Snapshot(s) to be exported>*	Snapshot(s) to be exported. Valid snapshot options include <synthesized|final>.
	`-user_metadata` *<user_metadata>*	The absolute or relative path to the User Metadata configuration file.
	<Partition to be exported>	Name of the partition to be exported.
Description	Export the metadata and IP of the specified partition and snapshot.	
Example Usage	project_open onewire_nf design::export_partition core_ptn -snapshot synthesized -file ip.qdb project_close	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6.9. design::extract_metadata (::quartus::design)

The following table displays information for the `design::extract_metadata` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::extract_metadata [-h | -help] [-long_help] -dir <dir> -file <file> [-overwrite] -type <type>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-dir <dir>`	The extraction directory. This directory must exist prior to invoking this command.
	`-file <file>`	The Partition Database File (.qdb) holding the metadata to be extracted.
	`-overwrite`	Attempt to overwrite any existing files in the extraction directory.
	`-type <type>`	The type of QDB metadata to be extracted. Legal QDB metadata types include <quartus|user|all>.
Description	`Extracts the specified metadata from the given Partition Database File (.qdb) file to the provided extraction directory.`	
Example Usage	`# Create a Partition Database File (.qdb) with Quartus Metadata using the "synthesized" snapshot.` `project_open onewire_nf` `design::export_design -file onewire.qdb -snapshot synthesized -quartus_metadata all` `project_close` `# Create the extraction directory.` `file mkdir "extract/dir"` `# Extract all the Quartus Metadata from "onewire.qdb" to the extraction directory located at "extract/dir".` `design::extract_metadata -file onewire.qdb -type quartus -dir "extract/dir"`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6.10. design::get_assignment_info (::quartus::design)

The following table displays information for the `design::get_assignment_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::get_assignment_info [-h | -help] [-long_help] [-from] [-name] [-to] [-value] <assignment>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-from`	Return the source name of the assignment id
	`-name`	Return the type name of the assignment id
	`-to`	Return the destination name of the assignment id

continued...

	`-value`		Return the value of the assignment id
	<assignment>		assignment id
Description	`Get information about a given assignent ID`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot` `foreach asgn_id [design::get_assignments] {` `  puts "Found assignment [design::get_assignment_info -name $asgn_id]` `[design::get_assignment_info -to $asgn_id] = [design::get_assignment_info -value $asgn_id]"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied assignment id *<string>* is invalid.
	TCL_ERROR	1	ERROR: At least one periphery assignment ID must be supplied, but no assignments IDs were supplied.
	TCL_ERROR	1	ERROR: *<string>* assignment IDs were expected but *<string>* were supplied.

3.1.6.11. design::get_assignment_names (::quartus::design)

The following table displays information for the `design::get_assignment_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::get_assignment_names [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Get a list of valid assignment type names`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot` `puts "Valid assignment type names:"` `foreach asgn_type [lsort [design::get_assignment_names]] {` `  puts $asgn_type` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6.12. design::get_assignments (::quartus::design)

The following table displays information for the `design::get_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141
Syntax	`design::get_assignments [-h | -help] [-long_help] [-deleted] [-disabled] [-enabled] [-ignored] [-name <name> ]`

continued...

Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-deleted`	Return only deleted assignments
	`-disabled`	Return only disabled assignments
	`-enabled`	Return only enabled assignments
	`-ignored`	Return only ignored assignments
	`-name` <*name*>	Return only assignments of the provided type name
Description	`Get a list of assignment IDs for the currently loaded design.`	
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot` `foreach asgn_id [design::get_assignments] {` `  puts "Found assignment [design::get_assignment_info -name $asgn_id]` `[design::get_assignment_info -to $asgn_id] = [design::get_assignment_info -value $asgn_id]"` `}`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful
	TCL_ERROR	1 / ERROR: The assignment with id <*string*> is not enabled.
	TCL_ERROR	1 / ERROR: The supplied assignment type name <*string*> is invalid.
	TCL_ERROR	1 / ERROR: At least one assignment type name must be supplied, but no type namess were supplied.
	TCL_ERROR	1 / ERROR: <*string*> assignment type names were expected but <*string*> were supplied.

3.1.6.13. design::get_entity_names (::quartus::design)

The following table displays information for the `design::get_entity_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::get_entity_names [-h | -help] [-long_help] [` <*filter*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*filter*>	Object filter
Description	`Get a list of entity names in the loaded design`	
Example Usage	`This command currently contains no example usage.`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful

3.1.6.14. design::get_instances (::quartus::design)

The following table displays information for the `design::get_instances` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::get_instances [-h | -help] [-long_help] [-entity <entity> ] [ <filter> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-entity <entity>`		Return only instance names that have the supplied entity name
	<filter>		Object filter
Description	`Get a list of instances in the loaded design`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6.15. design::import_design (::quartus::design)

The following table displays information for the `design::import_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::import_design [-h | -help] [-long_help] -file <file> [-overwrite] [-timing_analysis_mode]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-file <file>`	The file.qdb to import from
	`-overwrite`	overwrites the databases in the active qdb directory
	`-timing_analysis_mode`	Import the design for Timing Analysis. User will not be able to generate programming file after importing design with this option. See -timing_analysis_mode option description below.
Description	`Import all the databases from the specified <file>.qdb.` `If overwrite is specified then databases will be overwritten in the active qdb directory.` `The database revision in the <file>.qdb must match the active revision.` `This command is available only in the quartus_cdb executable.` `------------------` `OPTIONS DESCRIPTION` `------------------` `-timing_analysis_mode` `Import the design for Timing Analysis. This option disables legality checks for certain configuration rules which may have changed from prior versions`	

continued...

	of Quartus Prime Pro. Use this option only if you were unable to successfully import your design without this option. After a design has been imported in timing analysis mode, the imported database will not be able to be used to generate programming files.
Example Usage	`# For the pro/quartus/sys/dsgn tests we need to export` `# a design first so there's a design to import` `project_open onewire_nf` `design::export_design -file onewire.qdb -snapshot synthesized` `project_close` `project_open onewire_nf` `design::import_design -file onewire.qdb -overwrite` `project_close`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified archive <*string*> does not exist.
	TCL_ERROR	1	ERROR: Databases already exist for the specified revision <*string*>. Use -overwrite to overwrite them.

3.1.6.16. design::import_partition (::quartus::design)

The following table displays information for the `design::import_partition` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::import_partition [-h | -help] [-long_help] -file` <*QDB file name*> `[-no_overwrite]` <*Partition to be imported*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-file` <*QDB file name*>	File name of the QDB archive.
	`-no_overwrite`	Don't delete existing snapshots when importing partition
	<*Partition to be imported*>	Partition name at which the imported IP will be rooted.
Description	`Import a partition into the current design.`	
Example Usage	`# You need to run design::export_partition from a source design` `# so that there's a partition to import from` `project_open onewire_nf` `design::export_partition root_partition -snapshot synthesized -file ip.qdb` `project_close` `# The imported DB file can be used as root_partition` `project_open onewire_nf` `design::import_partition root_partition -file ip.qdb` `project_close` `# Or non-root_partition` `project_open onewire_nf` `design::import_partition ip_sub -file ip.qdb` `project_close`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified archive <*string*> does not exist.

3.1.6.17. design::list_valid_snapshot_names (::quartus::design)

The following table displays information for the `design::list_valid_snapshot_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::list_valid_snapshot_names [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns the list of the valid design snapshot names.`		
Example Usage	`puts "Valid design snapshot names: [design::list_valid_snapshot_names]"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6.18. design::load_design (::quartus::design)

The following table displays information for the `design::load_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::load_design [-h | -help] [-long_help] [-flat_only] [-latest_snapshot] [-snapshot <snapshot> ] [-writeable]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-flat_only`		Indicates that databases should be loaded only from the flat partition.
	`-latest_snapshot`		Load the latest snapshot for the design
	`-snapshot <snapshot>`		Snapshot name to load database(s) from
	`-writeable`		Loads databases in a writeable mode.
Description	`Load the databases for the currently opened project. The databases are by default loaded in read-only mode and must be loaded with the -writeable option if manipulation to the databases is desired.`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot` `foreach asgn_id [design::get_assignments] {` `  puts "Found assignment [design::get_assignment_info -name $asgn_id] [design::get_assignment_info -to $asgn_id] = [design::get_assignment_info -value $asgn_id]"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: The specified snapshot name, <*string*>, is invalid.
	TCL_ERROR	1	ERROR: At least one snapshot must be supplied, but no snapshots were supplied.
	TCL_ERROR	1	ERROR: <*string*> snapshots were expected but <*string*> were supplied.

3.1.6.19. design::report_assignments (::quartus::design)

The following table displays information for the `design::report_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141		
Syntax	`design::report_assignments [-h | -help] [-long_help] [-deleted] [-disabled] [-enabled] [-ignored] [-name <name> ] [-panel_name <name> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-deleted`	Report only deleted assignments	
	`-disabled`	Report only disabled assignments	
	`-enabled`	Return only enabled assignments	
	`-ignored`	Report only ignored assignments	
	`-name <name>`	The type name of the assignments to report	
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel	
Description	`Create a report of all instance assignments in the loaded design`		
Example Usage	`project_open onewire_nf` `design::load_design -latest_snapshot` `design::report_assignments -name location`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.6.20. design::set_assignment_info (::quartus::design)

The following table displays information for the `design::set_assignment_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::design on page 141	
Syntax	`design::set_assignment_info [-h | -help] [-long_help] [-disable] [-enable] [-value <value> ] <assignment>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-disable`	Disable the assignment
	`-enable`	Enable the assignment

continued...

	`-value <value>`	Set the assignment value
	<assignment>	assignment id
Description		
	`Set information for a given assignent ID`	
Example Usage		
	`project_open onewire_nf` `design::load_design -latest_snapshot` `set asgn_id [lindex [design::get_assignments -name location] 0]` `puts "Setting location of [design::get_assignment_info -to $asgn_id] to PIN_A5"` `design::set_assignment_info -value PIN_A5 $asgn_id` `puts "New location of [design::get_assignment_info -to $asgn_id] is [design::get_assignment_info -value $asgn_id]"`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful
	TCL_ERROR	1 / ERROR: The supplied assignment id *<string>* is invalid.
	TCL_ERROR	1 / ERROR: The supplied assignment value *<string>* is invalid.
	TCL_ERROR	1 / ERROR: At least one periphery assignment ID must be supplied, but no assignments IDs were supplied.
	TCL_ERROR	1 / ERROR: Either the -to or -from argument is required.
	TCL_ERROR	1 / ERROR: The value of an assignment cannot be empty.
	TCL_ERROR	1 / ERROR: *<string>* assignment IDs were expected but *<string>* were supplied.

3.1.7. ::quartus::device

The following table displays information for the **::quartus::device** Tcl package:

Tcl Package and Version	::quartus::device 1.0
Description	
	`This package contains the set of Tcl functions` `for accessing information from the Quartus Prime` `device database.`
Availability	
	`This package is loaded by default in the following executables:` `qpro_sh` `quartus_cdb` `quartus_eda` `quartus_fit` `quartus_ipgenerate` `quartus_sh` `quartus_sim` `quartus_sta` `quartus_syn` `This package is available for loading in the following executables:` `qpro` `quartus` `quartus_si`
Tcl Commands	
	`get_family_list (::quartus::device)` on page 155 `get_part_info (::quartus::device)` on page 155 `get_part_list (::quartus::device)` on page 156 `report_device_info (::quartus::device)` on page 157 `report_family_info (::quartus::device)` on page 157 `report_part_info (::quartus::device)` on page 158

3.1.7.1. get_family_list (::quartus::device)

The following table displays information for the `get_family_list` Tcl command:

Tcl Package and Version	Belongs to ::quartus::device on page 154		
Syntax	`get_family_list [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns a list of available families.`		
Example Usage	`get_family_list`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal or missing <*string*> value, '<*string*>'. Specify a legal value.

3.1.7.2. get_part_info (::quartus::device)

The following table displays information for the `get_part_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::device on page 154	
Syntax	`get_part_info [-h | -help] [-long_help] [-default_voltage] [-device] [-device_group] [-family] [-family_variant] [-fast_grade_revision] [-grade_revision] [-hssi_speed_grade] [-iobank_revision] [-package] [-pdn_model_status] [-pin_count] [-pof_id] [-power_model] [-power_model_status] [-rohs_grade] [-sip_tile] [-speed_grade] [-subdevice_id_code] [-subdevice_id_mask] [-temperature_grade]` <*part*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-default_voltage`	Option to get the default core voltage (such as 0.9V or 1.1V)
	`-device`	Option to get device name (such as EP1S25 or EP1S80)
	`-device_group`	Option to get the device_group
	`-family`	Option to get family name (such as Stratix or Cyclone)
	`-family_variant`	Option to get family variant (such as Base, E or GX)
	`-fast_grade_revision`	Option to get the fast_grade_revision
	`-grade_revision`	Option to get the grade_revision
	`-hssi_speed_grade`	Option to get the hssi speed grade
	`-iobank_revision`	Option to get the iobank_revision
	`-package`	Option to get package name (such as FBGA or BGA)
	`-pdn_model_status`	Option to get the power distribution network (PDN) model status (such as ADVANCE, PRELIMINARY, or FINAL)

continued...

	-pin_count	Option to get total number of pins in the package
	-pof_id	Option to get the pof_id
	-power_model	Option to get the power model (such as STANDARD or LOW)
	-power_model_status	Option to get the power model status (such as PRELIMINARY or FINAL)
	-rohs_grade	Option to get the ROHS grade (ROHS5, ROHS6, Leaded)
	-sip_tile	Option to get SiP tile (such as L-tile, H-tile, E-tile)
	-speed_grade	Option to get speed grade (such as 5, 6, or 7)
	-subdevice_id_code	Option to get the subdevice_id_code
	-subdevice_id_mask	Option to get the subdevice_id_mask
	-temperature_grade	Option to get temperature grade of the package (such as COMMERCIAL or INDUSTRIAL)
	<part>	Part name

Description

```
Returns part characteristics for the specified part.

If you use multiple options, the command returns a list
in the following order:
    <family> <device> <package> <pin_count> <speed grade> <temperature_grade> <family_variant>
<power_model_status> <hssi_speed_grade> <power_model> <rohs_grade>
```

Example Usage

```
tcl> get_part_info -family EP1S25F780C5
tcl> Stratix

tcl> get_part_info -family -device EP1S25F780C5
tcl> Stratix EP1S25

tcl> get_part_info -device -package -pin_count -speed_grade EP1S25F780C5
tcl> EP1S25 FBGA 780 5
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Illegal die location name: *<string>*. Specify a legal die location name.
TCL_ERROR	1	ERROR: Illegal resource name: *<string>*. Specify a legal resource name.

3.1.7.3. get_part_list (::quartus::device)

The following table displays information for the `get_part_list` Tcl command:

Tcl Package and Version	Belongs to ::quartus::device on page 154
Syntax	`get_part_list [-h | -help] [-long_help] [-device` *<value>* `] [-family` *<value>* `] [-package` *<value>* `] [-pin_count` *<value>* `] [-speed_grade` *<value>* `]`
Arguments	`-h | -help` — Short help
	`-long_help` — Long help with examples and possible return values
	`-device` *<value>* — Option to match device name
	`-family` *<value>* — Option to match family name
	`-package` *<value>* — Option to match package name

continued...

	`-pin_count` <*value*>	Option to match pin count
	`-speed_grade` <*value*>	Option to match speed grade
Description	`Returns a list of available parts based on the options that are specified. Examples are as follows:` `  Return a list of all supported parts` `  get_part_list` `  Return a list of all supported parts for Cyclone` `  get_part_list -family Cyclone` `  Return a list of all supported parts with the FBGA package and 780 pins` `  get_part_list -package fbga -pin_count 780`	
Example Usage	`get_part_list -family "Stratix IV"`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal or missing <*string*> value, '<*string*>'. Specify a legal value.

3.1.7.4. report_device_info (::quartus::device)

The following table displays information for the `report_device_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::device on page 154	
Syntax	`report_device_info [-h | -help] [-long_help]` <*device*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*device*>	Device name
Description	`Returns a string value containing the report with information about the specified device, such as the following:` `  Available parts` `  Some additional information specific to the device`	
Example Usage	`set report [report_device_info APEX20K1000E]` `puts $report`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal device name: <*string*>. Specify a legal device name.

3.1.7.5. report_family_info (::quartus::device)

The following table displays information for the `report_family_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::device on page 154	
Syntax	`report_family_info [-h | -help] [-long_help]` <*family*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	<family>		Family name
Description	 `Returns a string value containing the report with information about the specified family, such as the following:` `Available devices` `Available packages` `Available speed grades` `Available pin counts` `Some additional information specific to the family`		
Example Usage	`set report [report_family_info Stratix]` `puts $report`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal family name: *<string>*. Specify a legal family name.

3.1.7.6. report_part_info (::quartus::device)

The following table displays information for the `report_part_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::device on page 154		
Syntax	`report_part_info [-h | -help] [-long_help] <part>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<part>		Part name
Description	 `Returns a string value containing the report with information about the specified part, such as the following:` `Family name` `Device name` `Package name` `Pin count` `Speed grade` `Any additional information`		
Example Usage	`set report [report_part_info EPM1270F256I5]` `puts $report`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal part name: *<string>*. Specify a legal part name. Use report_family_info, report_device_info, or get_part_list to find available parts.

3.1.8. ::quartus::dni_sdc

The following table displays information for the **::quartus::dni_sdc** Tcl package:

Tcl Package and Version	::quartus::dni_sdc 1.5
Description	 `This package contains no general description.`

continued...

Availability	This package is available for loading in the following executables: qpro quartus quartus_cdb quartus_sh quartus_syn
Tcl Commands	dni::create_generated_clock (::quartus::dni_sdc) on page 159 dni::remove_clock_groups (::quartus::dni_sdc) on page 161 dni::remove_clock_latency (::quartus::dni_sdc) on page 162 dni::remove_clock_uncertainty (::quartus::dni_sdc) on page 163 dni::remove_disable_timing (::quartus::dni_sdc) on page 164 dni::remove_input_delay (::quartus::dni_sdc) on page 164 dni::remove_output_delay (::quartus::dni_sdc) on page 165 dni::set_clock_groups (::quartus::dni_sdc) on page 166 dni::set_clock_latency (::quartus::dni_sdc) on page 167 dni::set_clock_uncertainty (::quartus::dni_sdc) on page 168 dni::set_data_delay (::quartus::dni_sdc) on page 170 dni::set_disable_timing (::quartus::dni_sdc) on page 172 dni::set_false_path (::quartus::dni_sdc) on page 173 dni::set_input_delay (::quartus::dni_sdc) on page 174 dni::set_input_transition (::quartus::dni_sdc) on page 176 dni::set_max_delay (::quartus::dni_sdc) on page 177 dni::set_max_skew (::quartus::dni_sdc) on page 179 dni::set_max_time_borrow (::quartus::dni_sdc) on page 180 dni::set_min_delay (::quartus::dni_sdc) on page 181 dni::set_multicycle_path (::quartus::dni_sdc) on page 183 dni::set_net_delay (::quartus::dni_sdc) on page 185 dni::set_operating_conditions (::quartus::dni_sdc) on page 186 dni::set_output_delay (::quartus::dni_sdc) on page 188 dni::set_sense (::quartus::dni_sdc) on page 189 dni::set_timing_derate (::quartus::dni_sdc) on page 190

3.1.8.1. dni::create_generated_clock (::quartus::dni_sdc)

The following table displays information for the `dni::create_generated_clock` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::create_generated_clock [-h | -help] [-long_help] [-add] [-combinational] [-comment` *<string>* `] [-divide_by` *<factor>* `] [-duty_cycle` *<percent>* `] [-edge_shift` *<shift_list>* `] [-edges` *<edge_list>* `] [-invert] [-master_clock` *<clock>* `] [-multiply_by` *<factor>* `] [-name` *<clock_name>* `] [-offset` *<time>* `] [-phase` *<degrees>* `] [-preinvert] [-source` *<clock_source>* `] [` *<targets>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add`	Add clock to existing clock node
	`-combinational`	Specifies that the source latency paths for this type of generated clock only includes the logic where the master clock propagates along combinational paths
	`-comment` *<string>*	Comment string
	`-divide_by` *<factor>*	Division factor
	`-duty_cycle` *<percent>*	Specifies the duty cycle as a percentage of the clock period--accepts floating point values
	`-edge_shift` *<shift_list>*	List of edge shifts
	`-edges` *<edge_list>*	List of edge values
	`-invert`	Invert the clock waveform
	`-master_clock` *<clock>*	Specifies clock of the source node

continued...

	`-multiply_by` *<factor>*	Multiplication factor
	`-name` *<clock_name>*	Name of generated clock
	`-offset` *<time>*	Specifies the offset as an absolute time shift
	`-phase` *<degrees>*	Specifies the phase shift in degrees
	`-preinvert`	Preinvert the clock waveform
	`-source` *<clock_source>*	Source pin for the generated clock
	<targets>	List or collection of targets

Description

```
Defines an internally generated clock. If -name is not specified, the
clock name is the same as the first target in the list or collection.
The clock name is used to refer to the clock in other commands.

If a clock with the same name is already assigned to a given target,
the create_generated_clock command overwrites the existing clock. If a
clock with a different name exists on the given target, the
create_generated_clock command is ignored unless the -add option is
used. The -add option can be used to assign multiple clocks to a pin
or port, and is recommended be used with -master_clock option.

The master clock must be defined on (or must propagate to) the port or
pin specified in -source. The waveform of the generated clock is
computed based on the master clock's waveform as observed at that port
or pin, including inversions that may occur between the master clock
source and that pin.

The source latency of the generated clock is based on the clock
network of the generated clock, and not the clock network of the node
specified using -source. This latency is added to any source latency
of the master clock.

If no target is specified, the clock is treated as a virtual clock. In
that case, the source latency of the generated clock will be equal to
the source latency of the master clock, plus any added latency
specified with set_clock_latency.

The -divide_by, -multiply_by, -invert, -duty_cycle, -edges, and
-edge_shift options modify the waveform relative to the waveform at
the source node.

Clock division and multiplication, using -divide_by and -multiply_by,
is performed relative to the first rising edge. Clock division is
based on edges in the master clock waveform, and scaled if the
division is an odd number. Use the -duty_cycle option to specify the
new duty cycle for clock multiplication. Use the -invert option to
invert the generated waveform. The  -duty_cycle option may be
specified as a decimal value (e.g. 22.5) or as a ratio of two numbers
(e.g. 45/2). The latter form may improve Timing Analyzer accuracy when
detecting relationships between related clocks.

Clock generation can also be specified with the -edges and -edge_shift
options. The -edges option accepts a list of three numbers specifying
the master clock edges to use for the first rising edge, the next
falling edge, and next rising edge. Edges of the master clock are
labeled according to the first rising edge (1), next falling edge (2),
next rising edge (3), etc. For example, a basic clock divider can be
specified equivalently with -divide_by 2 or -edges {1 3 5}. The
-edge_shift option accepts a list of three time values, the amount to
shift each of the three edges.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.

Note -preinvert and -combinational options are not supported.
```

Example Usage

```
# Create a clock and a divide-by-2 generated clock
create_clock -period 10 [get_ports clk]
create_generated_clock -divide_by 2 -source [get_ports clk] -name clkdiv [get_cells clkdiv]

# An equivalent generated clock
create_generated_clock -edges {1 3 5} -source [get_ports clk] -name clkdiv [get_cells clkdiv]

# Specify a clock multipler with a 60% duty cycle
create_generated_clock -multiply_by 2 -source [get_ports clk] -duty_cycle 60 [get_pins clkmult|
combout]

# Specify an inverted divide-by-2 clock relative to the output of the source clock
```

continued...

```
create_generated_clock -divide_by 2 -invert -source [get_ports clk] -name nclkdiv [get_cells
clkdiv]

# Specify a divide-by-2 clock
create_generated_clock -divide_by 2 -source [get_ports clk] -name clkdiv [get_cells clkdiv]

# Create a divide-by-2 generated clock generated off the falling edge of the source clock
create_generated_clock -edges {2 4 6} -source [get_ports clk] -name clkfall_div [get_cells
clkfall_div]

# Assign two clocks to an input port that are switched externally,
# along with an internal clock divider.
create_clock -period 10 -name clk100Mhz [get_ports clk]
create_clock -period 6.667 -name clk150Mhz -add [get_ports clk]
create_generated_clock -divide_by 2 -name clk50Mhz -source [get_ports clk] -master_clock
clk100Mhz -add [get_cells clkdiv]
create_generated_clock -divide_by 2 -name clk75Mhz -source [get_ports clk] -master_clock
clk150Mhz -add [get_cells clkdiv]
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.8.2. dni::remove_clock_groups (::quartus::dni_sdc)

The following table displays information for the `dni::remove_clock_groups` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::remove_clock_groups [-h | -help] [-long_help] -all [-asynchronous] [-logically_exclusive] [-physically_exclusive] [ <name_list> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all`	Specify remove all clock group settings
	`-asynchronous`	Specify mutually exclusive clocks (such as groups of primary clocks)
	`-logically_exclusive`	Specify logically exclusive clocks (meaning they are not actively used at the same time)
	`-physically_exclusive`	Specify physically exclusive clocks (meaning they are not physically present at the same time)
	<name_list>	Clock group name list
Description		

```
Remove all clock group assignments. This command removes any clock
groups that have been previously set. There is no way to remove
specific groups yet, therefore the -all option has to be given. All
other options are supported.
```

Example Usage

```
project_open top
create_timing_netlist
create_clock -period 10.000 -name clkA [get_ports sysclk[0]]
create_clock -period 10.000 -name clkB [get_ports sysclk[1]]

# Set clkA and clkB to be mutually exclusive clocks.
set_clock_groups -exclusive -group {clkA} -group {clkB}
set_clock_groups -exclusive -group {clkC} -group {clkD}

# Remove clock groups A, B, C, and D. Result is that there
# are no longer any mutually exclusive clocks.
remove_clock_groups -all
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.8.3. dni::remove_clock_latency (::quartus::dni_sdc)

The following table displays information for the `dni::remove_clock_latency` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::remove_clock_latency [-h | -help] [-long_help] [-early] [-fall] [-late] [-max] [-min] [-rise] -source <object_list>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-early`	Specifies the early clock latency
	`-fall`	Specifies the falling transition clock latency
	`-late`	Specifies the late clock latency
	`-max`	Specifies the clock latency at the worst-case operation condition
	`-min`	Specifies the clock latency at the best-case operation condition
	`-rise`	Specifies the rising transition clock latency
	`-source`	Specifies the source clock latency
	<object_list>	Valid destinations (string patterns are matched using Tcl string matching)
Description		

```
Removes clock latency for a given clock or clock target.

There are two types of latency: network and source. Network latency is
the clock network delay between the clock and register clock pins.
Source latency is the clock network delay between the clock and its
source (e.g., a system clock or a base clock of a generated clock).

The Timing Analyzer automatically computes network latencies for all
register and generated clocks. Overriding clock network latencies is
not supported by the Timing Analyzer. Therefore, the -source option
must always be specified. Remove_clock_latency requires this option as
well.

You can apply clock latency to a clock, which affects all targets of
the clock, or to a specific clock target. Therefore, you can remove
clock latency from a collection of clocks, or from a collection of
target nodes. remove_clock_latency removes all latencies from a clock
or node, so removing a node's clock latency with respect to a
particular clock, or removing only latencies with particular
conditions is not supported.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.
```

Example Usage

```
create_clock -name SYSCLK -period 10.000 [get_ports inclk]
create_generated_clock -name OUTCLK -divide_by 1 -source [get_ports inclk] [get_ports outclk]
create_generated_clock -name FDBKCLK -divide_by 1 -source [get_ports outclk] [get_ports
fdbkclk]

# Apply a simple 2.000 ns source latency to the system clock.
set_clock_latency -source 2.000 [get_clocks SYSCLK]

# Specify feedback clock latencies between output port outclk
# and the output port fdbkclk.
set_clock_latency -source -late -rise 0.800 [get_clocks FDBKCLK]
set_clock_latency -source -late -fall 0.750 [get_clocks FDBKCLK]
set_clock_latency -source -early -rise 0.500 [get_clocks FDBKCLK]
set_clock_latency -source -early -fall 0.460 [get_clocks FDBKCLK]
```

continued...

| | ```
# Remove all clock latency from FDBKCLK
remove_clock_latency -source [get_clocks FDBKCLK]
``` | | |
| **Return Value** | **Code Name** | **Code** | **String Return** |
| | TCL_OK | 0 | INFO: Operation successful |

3.1.8.4. dni::remove_clock_uncertainty (::quartus::dni_sdc)

The following table displays information for the `dni::remove_clock_uncertainty` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::remove_clock_uncertainty [-h | -help] [-long_help] [-fall] [-fall_from <fall_from_clock> ] [-fall_to <fall_to_clock> ] [-from <from_clock> ] [-hold] [-rise] [-rise_from <rise_from_clock> ] [-rise_to <rise_to_clock> ] [-setup] [-to <to_clock> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fall`	Indicates that the uncertainty applies to only the falling edge of the destination clock
	`-fall_from <fall_from_clock>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-fall_to <fall_to_clock>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-from <from_clock>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-hold`	Only apply the uncertainty value to hold and removal checks
	`-rise`	Indicates that the uncertainty applies to only the rising edge of the destination clock
	`-rise_from <rise_from_clock>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-rise_to <rise_to_clock>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-setup`	Only apply the uncertainty value to setup and recovery checks
	`-to <to_clock>`	Valid destinations (string patterns are matched using Tcl string matching)
| **Description** | ```
Removes clock uncertainty from a collection of clocks to a collection
of clocks. The source and destination clocks can be any arbitrary
collection of clocks. This command removes all uncertainty between two
clocks. If there does not exist uncertainty between two clocks
specified in remove_clock_uncertainty, the command does nothing for
those two clocks but continues to attempt to remove uncertainty
between other clocks specified.

The values of the -from and -to options are either collections or a
Tcl list of wildcards used to create collections of appropriate types.

Note -rise and -fall options are not supported yet.
``` | |

continued...

Example Usage	`set_clock_uncertainty -setup -rise_from {clk1 clk2} -fall_to {clk3 clk4} 200ps` `set_clock_uncertainty -from {clk5 clk6} -to {clk7 clk8} 300ps` `remove_clock_uncertainty -from {clk3 clk5} -to {clk4 clk7}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.5. dni::remove_disable_timing (::quartus::dni_sdc)

The following table displays information for the `dni::remove_disable_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::remove_disable_timing [-h | -help] [-long_help] [-all_loop_breaking] [-from <name> ] [-to <name> ] <cells>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all_loop_breaking`	Re-enables all stored loop-breaking timing arcse
	`-from <name>`	Valid source pin suffix
	`-to <name>`	Valid destination pin suffix
	<cells>	List of cells
Description	`Adds a previously disabled edge (arc) back to a given cell(s). If no -from/-to value is specified, the missing value is substituted by a "*".` `The values of the -from and -to are valid pin suffixes.` `Note -all_loop_breaking option is not supported yet.`	
Example Usage	`remove_disable_timing -from datain -to combout A|B` `remove_disable_timing -from carryin *`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.6. dni::remove_input_delay (::quartus::dni_sdc)

The following table displays information for the `dni::remove_input_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::remove_input_delay [-h | -help] [-long_help] [-clock <name> ] [-clock_fall] [-fall] [-level_sensitive] [-max] [-min] [-rise] <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clock <name>`	Clock name
	`-clock_fall`	Specifies that input delay is relative to the falling edge of the clock

continued...

	-fall	Specifies the falling input delay at the port
	-level_sensitive	Specifies that input delay is relative to a level-sensitive latch
	-max	Applies value as maximum data arrival time
	-min	Applies value as minimum data arrival time
	-rise	Specifies the rising input delay at the port
	<*targets*>	Collection or list of input ports
Description		

```
Removes input delay from a port. For each input port specified,
removes all input delays for that port. This means that rise, fall,
max, and min delays for each clock and reference pin on the input port
are all removed.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.
```

Example Usage

```
# Simple input delay with the same value for min/max and rise/fall
set_input_delay -clock clk 1.5 [get_ports {in1 in2}]
set_input_delay -clock clk2 1.5 [get_ports {in1 in2}]
set_input_delay -clock clk 1.6 [get_ports {in3 in4}]

# Remove input delay on ports in1 and in4,
# for all flags and reference ports and flags
remove_input_delay [get_ports {in1 in4}]
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.7. dni::remove_output_delay (::quartus::dni_sdc)

The following table displays information for the `dni::remove_output_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::remove_output_delay [-h | -help] [-long_help] [-clock` <*name*> `] [-clock_fall] [-fall] [-max] [-min] [-rise]` <*targets*>	
Arguments	-h | -help	Short help
	-long_help	Long help with examples and possible return values
	-clock <*name*>	Clock name
	-clock_fall	Specifies that input delay is relative to the falling edge of the clock
	-fall	Specifies the falling input delay at the port
	-max	Applies value as maximum data arrival time
	-min	Applies value as minimum data arrival time
	-rise	Specifies the rising input delay at the port
	<*targets*>	Collection or list of output ports
Description		

```
Removes output delay from a port. For each output port specified,
removes all output delays for that port. Rise, fall, max, and min
delays for each clock and reference pin on the output port are all
removed.
```

continued...

	The value of the targets is either a collection or a Tcl list of wildcards used to create a collection of the appropriate type. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See help for the use_timing_analyzer_style_escaping command for details.
Example Usage	# Simple output delay with the same value for min/max and rise/fall set_output_delay -clock clk 1.5 [get_ports {out1 out2}] set_output_delay -clock clk2 1.5 [get_ports {out1 out2}] set_output_delay -clock clk 1.6 [get_ports {out3 out4}] # Remove input delay on ports out1 and out4, # for all flags and reference ports and flags remove_output_delay [get_ports {out1 out4}]

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.8. dni::set_clock_groups (::quartus::dni_sdc)

The following table displays information for the `dni::set_clock_groups` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_clock_groups [-h | -help] [-long_help] [-allow_paths] [-asynchronous] [-comment <string> ] [-exclusive] -group <names> [-logically_exclusive] [-name <name> ] [-physically_exclusive]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-allow_paths`	Enables timing analysis between the specified clock groups
	`-asynchronous`	Specify mutually exclusive clocks (such as groups of primary clocks)
	`-comment <string>`	Comment string
	`-exclusive`	Specify mutually exclusive clocks (an alias for the -logically_exclusive option). Exists for backwards compatibility.
	`-group <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-logically_exclusive`	Specify logically exclusive clocks (meaning they are not actively used at the same time)
	`-name <name>`	Clock group name
	`-physically_exclusive`	Specify physically exclusive clocks (meaning they are not physically present at the same time)
Description	Clock groups provide a quick and convenient way to specify which clocks are not related. Asynchronous clocks are those that are completely unrelated (e.g., have different ideal clock sources). Logically exclusive clocks are not actively used in the design at the same time (e.g., multiplexed clocks), but the clock signals may physically exist on-chip at the same time and therefore may still influence each other through crosstalk effects. Physically exclusive clocks, in contrast, cannot be physically present in the device at the same time (e.g., multiple clocks defined on the same clock pin). The Timing Analyzer does not currently analyze crosstalk explicitly. Instead, the timing models use extra guard bands to account for any potential crosstalk-induced delays. As a result, the Timing Analyzer	

continued...

```
currently treats asynchronous, logically_exclusive, and
physically_exclusive clock groups the same. However, different parts
of the Timing Analyzer may treat asynchronous and exclusive groups
differently. Any commands that are affected by clock groups will say
so in their help text. But, no distinction is made between logically
and physically exclusive clock groups, since the only difference
between them is how they affect crosstalk.

The result of set_clock_groups is that all clocks in any group are cut
from all clocks in every other group. The use of a single -group
option tells the Timing Analyzer to cut this group of clocks from all
other clocks in the design, including clocks that are created in the
future. This command is similar to calling set_false_path from each
clock in every group to each clock in every other group and vice
versa, making set_clock_groups easier to specify for cutting clock
domains. However, cutting clocks with set_clock_groups also affects
the results of some other commands. Any commands that are affected by
clock groups will say so in their help text.

Note -allow_paths is not supported yet.
```

Example Usage

```
project_open top
create_timing_netlist
create_clock -period 10.000 -name clkA [get_ports sysclk[0]]
create_clock -period 10.000 -name clkB [get_ports sysclk[1]]

# Set clkA and clkB to be mutually exclusive clocks.
dni::set_clock_groups -logically_exclusive -group {clkA} -group {clkB}

# The previous line is equivalent to the following two commands.
set_false_path -from [get_clocks clkA] -to [get_clocks clkB]
set_false_path -from [get_clocks clkB] -to [get_clocks clkA]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.8.9. dni::set_clock_latency (::quartus::dni_sdc)

The following table displays information for the `dni::set_clock_latency` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_clock_latency [-h | -help] [-long_help] [-clock <clock_list> ] [-dynamic <jitter> ] [-early] [-fall] [-late] [-max] [-min] [-rise] -source <delay> <object_list>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clock <clock_list>`	Valid clock destinations (string patterns are matched using Tcl string matching)
	`-dynamic <jitter>`	Specifies the dynamic component of the clock latency, which represents the amount of jitter in the original clock source
	`-early`	Specifies the early clock latency
	`-fall`	Specifies the falling transition clock latency
	`-late`	Specifies the late clock latency
	`-max`	Specifies the clock latency at the worst-case operation condition
	`-min`	Specifies the clock latency at the best-case operation condition
	`-rise`	Specifies the rising transition clock latency
	`-source`	Specifies the source clock latency

continued...

	<delay>	Latency delay value
	<object_list>	Valid destinations (string patterns are matched using Tcl string matching)

Description

```
Specifies clock latency for a given clock or clock target.

There are two types of latency: network and source. Network latency is
the clock network delay between the clock and register clock pins.
Source latency is the clock network delay between the clock and its
source (e.g., the system clock or base clock of a generated clock).

The Timing Analyzer automatically computes network latencies for all
register and generated clocks. Overriding clock network latencies is
not supported by the Timing Analyzer. Therefore, the -source option
must always be specified.

You can apply clock latency to a clock, which affects all targets of
the clock, or to a specific clock target. If you specify a specific
clock target that is driven by more than one clock, use the -clock
option to specify which clock to use.Latencies assigned to a clock
target override any latencies assigned to a clock.

Different clock latencies can be specified for early (-early) and late
(-late) latencies, as well as for rising edges (-rise) and falling
edges (-fall). If only some combinations are specified, the other
combinations are used by default. For example, if only a -rise -early
latency and a -fall -early latency are specified, then the -rise -late
latency is assumed to be the same as the -rise -early latency and the
-fall -late latency is assumed to be the same as the -fall -early
latency. If neither -rise nor -fall are used or neither -early nor
-late are used, then the latency applies to both conditions.

Source latency can also be assigned to generated clocks. This may be
useful for specifying board level delays from a clock output port to a
clock input port when the clock input port is acting as a feedback
clock.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.

Note -dynamic is not supported yet.
```

Example Usage

```
create_clock -name SYSCLK -period 10.000 [get_ports inclk]
create_generated_clock -name OUTCLK -divide_by 1 -source [get_ports inclk] [get_ports outclk]
create_generated_clock -name FDBKCLK -divide_by 1 -source [get_ports outclk] [get_ports
fdbkclk]

# Apply a simple 2.000 ns source latency to the system clock.
set_clock_latency -source 2.000 [get_clocks SYSCLK]

# Specify feedback clock latencies between output port outclk
# and the input port fdbkclk.
set_clock_latency -source -late -rise 0.800 [get_clocks FDBKCLK]
set_clock_latency -source -late -fall 0.750 [get_clocks FDBKCLK]
set_clock_latency -source -early -rise 0.500 [get_clocks FDBKCLK]
set_clock_latency -source -early -fall 0.460 [get_clocks FDBKCLK]
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.10. dni::set_clock_uncertainty (::quartus::dni_sdc)

The following table displays information for the `dni::set_clock_uncertainty` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_clock_uncertainty [-h | -help] [-long_help] [-add] [-enable_same_physical_edge] [-fall] [-fall_from` *<fall_from_clock>* `] [-fall_to` *<fall_to_clock>* `] [-from` *<from_clock>* `] [-hold] [-rise] [-rise_from` *<rise_from_clock>* `] [-rise_to` *<rise_to_clock>* `] [-setup] [-to` *<to_clock>* `] <uncertainty>`	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`	Long help with examples and possible return values
	`-add`	Specifies that this assignment is an addition to the clock uncertainty derived by derive_clock_uncertainty call
	`-enable_same_physical_edge`	Enable setting uncertainty value for same physical clock edge
	`-fall`	Indicates that the uncertainty applies to only the falling edge of the destination clock
	`-fall_from` *<fall_from_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-fall_to` *<fall_to_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-from` *<from_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-hold`	Only apply the uncertainty value to hold and removal checks
	`-rise`	Indicates that the uncertainty applies to only the rising edge of the destination clock
	`-rise_from` *<rise_from_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-rise_to` *<rise_to_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-setup`	Only apply the uncertainty value to setup and recovery checks
	`-to` *<to_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	<uncertainty>	Uncertainty

Description

```
Specifies clock uncertainty or skew for clocks for clock-to-clock
transfers. You can specify uncertainty separately for setup and hold,
and you can specify separate rising and falling clock transitions. If
you omit to specify -setup or -hold, the uncertainty value will be
applied to both analysis types. Similarly, if you omit to specify
rising or falling clock transitions, the uncertainty value will be
applied to both transitions. The setup uncertainty is subtracted from
the data required time for each applicable path, and the hold
uncertainty is added to the data required time for each applicable
path.

Intel Quartus Prime software computes clock uncertainty for every
clock transfer. For particular transfers, you can use the
set_clock_uncertainty assignment to override the automatically derived
value. If multiple set_clock_uncertainty assignments apply to the same
clock transfer, the later value overrides the earlier ones.

Note: The Timing Analyzer does not apply clock uncertainty to
transfers involving the same physical launch and latch edge (that is,
the latch and launch edges are the same edge of a clock source and
occur at the same time) by default. Such transfers typically occur in
hold analysis, but may also occur in setup analysis with a multicycle
value of 0.You can use the -enable_same_physical_edge option to
override this behavior.

The values for -from, -to, and similar options are either collections
or a Tcl list of wildcards used to create collections of appropriate
types.

Note -rise and -fall options are not supported yet.
```

Example Usage

```
set_clock_uncertainty -setup -rise_from clk1 -fall_to clk2 200ps
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.8.11. dni::set_data_delay (::quartus::dni_sdc)

The following table displays information for the `dni::set_data_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_data_delay [-h | -help] [-long_help] [-add_latch_clock] [-add_launch_clock] [-allow_destination_borrowing] [-fall_from <names>] [-fall_to <names>] [-from <names>] [-get_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period|max_clock_period>] [-no_synchronizer] [-override] [-rise_from <names>] [-rise_to <names>] [-through <names>] [-to <names>] [-value_multiplier <multiplier>] [<value>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add_latch_clock`	Include the latch clock path in timing analysis
	`-add_launch_clock`	Include the launch clock path in timing analysis
	`-allow_destination_borrowing`	Allow time borrowing at the destination
	`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-get_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period|max_clock_period>`	Compute constraint as a multiple of the clock period
	`-no_synchronizer`	Prevent this data delay from triggering a synchronizer
	`-override`	Make this constraint override non-datapath-only setup constraints, instead of applying it in addition to them (equivalent to set_data_delay & set_false_path -setup -no_synchronizer, unless -add_launch_clock is used as well)
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-value_multiplier <multiplier>`	Value by which the clock period should be multiplied to compute requirement
	<value>	Time Value
Description		

```
Specifies a maximum datapath delay exception for a given path.

The maximum delay analysis includes Tco of the launching register,
and Tsu of the latching register.  By default, it does not include
clock arrival times at the launching or latching register. To include
launch clock arrival times, use the -allow_launch_clock option. To
include latch clock arrival times, use the -allow_latch_clock option.
If the path starts or ends at a port, the analysis does not
```

continued...

```
include delays due to set_input_delay or set_output_delay.

Use -get_value_from_clock_period to set the delay requirement
for each path to be equal to the launching or latching clock period,
or whichever of the two has a smaller or larger period.
If -value_multiplier is used, the requirement will be multiplied by
that value.  If there are no clocks clocking the endpoints
of the path (such as if the path begins or ends at an unconstrained I/O),
the constraint will be ignored.

The datapath delay constraint is applied in addition to other constraints on
the given path, including the default constraint. Furthermore, the datapath delay
constraint is analyzed independently from other SDC constraints, including
set_false_path and set_clock_groups, and cannot be overridden by other SDC constraints.
For example, you can use set_data_delay to specify an upper limit on logic and routing
delay for paths cut by set_false_path.

To both cut a path for (clock-aware) timing and constrain its datapath delay, the path
must be constrained with both set_false_path and set_data_delay.

        The -from and -to values are collections of clocks, ports, pins, or
        cells in the design.

Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or ports
driven by the -to clock.

If pin names or collections are used, the -from value must be a clock
pin and the -to value must be any non-clock input pin to a register.
Assignments from clock pins or to and from cells apply to all
registers in the cell or driven by the clock pin.

The -through values are collections of pins or nets in the design.  An
exception applied through a node in the design applies only to paths
through the specified node.
the Timing Analyzer allows you to specify the -through argument multiple times to
describe paths that go through multiple points.
For instance, users can select all paths that go through node X, and then go through node Y.
This helps you narrow down and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
-from destination nodes.  The rise or fall value of the option
indicates that the "from" nodes are driven by the rising or falling
edge of the clock that feeds this node taking into consideration any
logical inversions along the clock path.  The "-from" option is the
combination of both rising and falling "from" nodes.  If the "from"
collection is a clock collection, the assignment applies to those
nodes that are driven by the respective rising or falling clock edge.

The -rise_to and -fall_to options behave similarly to the "from"
options described previously.  These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value taking into consideration any logical
inversions that are along the clock path.

The values of the -from, -to, -through, and other similar options are
either collections or a Tcl list of wildcards used to create
collections of appropriate types. The values used must follow standard
Tcl or Timing Analyzer-extension substitution rules. See help for the
use_timing_analyzer_style_escaping command for details.

If the source of a path with a set_data_delay constraint has any time
borrowed, the delay budget will be reduced by the time borrowed.

By default, the delay budget will not be increased by time borrowed at the
destination of a path constrained by a set_data_delay constraint, and
negative slack on a set_data_delay constraint will not cause time borrowing
to happen. To change this behavior, use the -allow_destination_borrowing
option.
```

continued...

Example Usage	<pre># Apply a 10ns max data delay on paths between two unrelated clocks set_data_delay -from [get_clocks clkA] -to [get_clocks clkB] 10.000 # Apply a 2ns max data delay from an input port to any valid destination set_data_delay -from [get_ports in[*]] -to * 2.000 # Require net delay to be at most 90% of the period of the clock driving the inst9 register set_data_delay -get_value_from_clock_period dst_clock_period -value_multiplier 0.9 -from [get_clocks clk] -to [get_cells inst9] # Apply a 2ns max data delay for an input port only to nodes driven by # the rising edge of clock CLK set_data_delay -from [get_ports in[*]] -rise_to [get_clocks CLK] 2.000</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.12. dni::set_disable_timing (::quartus::dni_sdc)

The following table displays information for the `dni::set_disable_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_disable_timing [-h | -help] [-long_help] [-from <name> ] [-restore] [-to <name> ] <cells>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-from <name>`	Valid source pin suffix
	`-restore`	Restores the specified arcs so that they are no longer disabled
	`-to <name>`	Valid destination pin suffix
	<cells>	List of cells
Description	<pre>Disables a timing edge (arc) from inside a given cell or cells. Disabling a timing edge prevents timing analysis through that edge. If either -from or -to (or both) are unspecified, the missing value or values are replaced by a "*" character. The values of the -from and -to are valid pin suffixes. Note -restore option is not supported yet.</pre>	
Example Usage	<pre>set_disable_timing -from datain -to combout A|B set_disable_timing -from carryin *</pre>	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.13. dni::set_false_path (::quartus::dni_sdc)

The following table displays information for the `dni::set_false_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_false_path [-h | -help] [-long_help] [-comment <string>] [-fall] [-fall_from <names>] [-fall_through <names>] [-fall_to <names>] [-from <names>] [-hold] [-latency_insensitive] [-reset_path] [-rise] [-rise_from <names>] [-rise_through <names>] [-rise_to <names>] [-setup] [-through <names>] [-to <names>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment <string>`	Comment string
	`-fall`	Marks falling delays false, as measured on the path endpoint
	`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-hold`	Specifies the false_path value (applies only to clock hold or removal checks)
	`-latency_insensitive`	Mark this false path as one that should still be optimized
	`-reset_path`	Removes existing point-to-point exception information on the specified paths
	`-rise`	Marks rising delays false, as measured on the path endpoint
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Specifies the false_path value (applies only to clock setup or recovery checks)
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
Description		

```
Specifies a false-path exception, removing (or cutting) paths from
timing analysis.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design. If the -from or -to values are not
specified, the collection is converted automatically into [get_keepers
*]. It is worth noting that if the counterpart of the unspecified
```

continued...

	collection is a clock collection, it is more efficient to explicitly specify this collection as a clock collection only if the clock collection also generates the desired assignment. Applying exceptions between clocks applies the exception from all register or ports driven by the -from clock to all registers or ports driven by the -to clock. Applying exceptions between a pair of clocks is more efficient than for specific node to node or node to clock paths. If pin names or collections are used, the -from value must be a clock pin and the -to value must be any non-clock input pin to a register. Assignments from clock pins or to and from cells applies to all registers in the cell or driven by the clock pin. The -through values are collections of pins or nets in the design. An exception applied through a node in the design applies only to paths through the specified node. The Timing Analyzer allows you to specify the -through argument multiple times to describe paths that go through multiple points. For instance, users can select all paths that go through node X, and then go through node Y. This helps you narrow down and select the specific paths that you are interested in. The -rise_from and -fall_from options can be used in place of the -from destination nodes. The rise or fall value of the option indicates that the "from" nodes are driven by the rising or falling edge of the clock that feeds this node, taking into consideration any logical inversions along the clock path. The -from option is the combination of both rising and falling "from" nodes. If the "from" collection is a clock collection, the assignment applies to those nodes that are driven by the respective rising or falling clock edge. The -rise_to and -fall_to options behave similarly to the "from" options described previously. These assignments restrict the given assignment to only those nodes or clocks that correspond to the specified rise or fall value, taking into consideration any logical inversions that are along the clock path. The -setup and -hold options allow the false path to only be applied to the corresponding setup/recovery or hold/removal checks. The default if neither value is specified is to apply the false path to both -setup and -hold. The values of the -from, -to, -through, and other similar options are either collections or a Tcl list of wildcards used to create collections of appropriate types. Note -rise, -fall, -rise_through, -fall_through and -reset_path options are not supported yet.
Example Usage	`# Set a false-path between two unrelated clocks` `set_false_path -from [get_clocks clkA] -to [get_clocks clkB]` `# Set a false-path for a specific path` `set_false_path -from [get_pins regA|clk] -to [get_pins regB|aclr]` `# Set a false-path from a node to a falling clock` `set_false_path -from [get_pins regA|clk] -fall_to [get_clocks clkB]`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.14. dni::set_input_delay (::quartus::dni_sdc)

The following table displays information for the `dni::set_input_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_input_delay [-h | -help] [-long_help] [-add_delay] -clock <name> [-clock_fall] [-fall] [-level_sensitive] [-max] [-min] [-network_latency_included] [-reference_pin <name> ] [-rise] [-source_latency_included] <delay> <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-add_delay`	Create additional delay constraint instead of overriding previous constraints
	`-clock` *<name>*	Clock name
	`-clock_fall`	Specifies that input delay is relative to the falling edge of the clock
	`-fall`	Specifies the falling input delay at the port
	`-level_sensitive`	Specifies that input delay is relative to a level-sensitive latch
	`-max`	Applies value as maximum data arrival time
	`-min`	Applies value as minimum data arrival time
	`-network_latency_included`	Specifies that input delay includes added network latency
	`-reference_pin` *<name>*	Specifies a pin or port in the design to which the input delay is relative
	`-rise`	Specifies the rising input delay at the port
	`-source_latency_included`	Specifies that input delay includes added source latency
	<delay>	Time value
	<targets>	List of input port type objects

Description

```
Specifies the data arrival times at the specified input ports relative
the clock specified by the -clock option. The clock must refer to a
clock name in the design.

Input delays can be specified relative to the rising edge (default) or
falling edge (-clock_fall) of the clock.

Input delays can be specified relative to a pin or a port (-reference_pin)
in the clock network. Clock arrival times to the reference pin or port
are added to data arrival times.

Input delays can already include clock source latency. By default the
clock source latency of the related clock is added to the input delay
value, but when the -source_latency_included option is specified, the
clock source latency is not added because it was factored into the
input delay value.

The maximum input delay (-max) is used for clock setup checks or
recovery checks and the minimum input delay (-min) is used for clock
hold checks or removal checks. If only -min or -max (or neither) is
specified for a given port, the same value is used for both.

Separate rising (-rise) and falling (-fall) arrival times at the port
can be specified. If only one of -rise and -fall are specified for a
given port, the same value is used for both.

By default, set_input_delay removes any other input delays to the port
except for those with the same -clock, -clock_fall, and -reference_pin
combination. Multiple input delays relative to different clocks, clock
edges, or reference pins can be specified using the -add_delay option.
Future input delays relative to different clocks, clock edges, or
reference pins that do not specify the -add_delay option will still
remove previous input delays that specified the -add_delay option.

Note that -level_sensitive and -network_latency_included are not
supported yet.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.
```

Example Usage

```
# Simple input delay with the same value for min/max and rise/fall:
# 1) set on ports with names of the form myin*
set_input_delay -clock clk 1.5 [get_ports myin*]
# 2) set on all input ports
set_input_delay -clock clk 1.5 [all_inputs]

# Input delay with respect to the falling edge of clock
set_input_delay -clock clk -clock_fall 1.5 [get_ports myin*]
```

continued...

```
# Input delays for different min/max and rise/fall combinations
set_input_delay -clock clk -max -rise 1.4 [get_ports myin*]
set_input_delay -clock clk -max -fall 1.5 [get_ports myin*]
set_input_delay -clock clk -min -rise 0.7 [get_ports myin*]
set_input_delay -clock clk -min -fall 0.8 [get_ports myin*]

# Adding multiple input delays with respect to more than one clock
set_input_delay -clock clkA -min 1.2 [get_ports myin*]
set_input_delay -clock clkA -max 1.8 [get_ports myin*]
set_input_delay -clock clkA -clock_fall 1.6 [get_ports myin*] -add_delay
set_input_delay -clock clkB -min 2.1 [get_ports myin*] -add_delay
set_input_delay -clock clkB -max 2.5 [get_ports myin*] -add_delay

# This is a common mistake where input delays are accidentally removed
set_input_delay -clock clkA -min 0.2 [get_ports myout*]
set_input_delay -clock clkB -min 1.1 [get_ports myout*] -add_delay
# The following removes the clkB entry. You need to always use -add_delay
#     when more than one clock, clock_fall or reference_pin exists
set_input_delay -clock clkA -max 0.8 [get_ports myout*]
# The following removes the clkA entry. You need to always use -add_delay
#     when more than one clock, clock_fall or reference_pin exists
set_input_delay -clock clkB -max 1.5 [get_ports myout*]

# Specifying an input delay relative to an external clock output port
set_input_delay -clock clk -reference_pin [get_ports clkout] 0.8 [get_ports myin*]

# Specifying an input delay relative to the clock pin of a register
set_input_delay -clock clk -reference_pin [get_pins regA|clk] 0.8 [get_ports myin*]
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.8.15. dni::set_input_transition (::quartus::dni_sdc)

The following table displays information for the `dni::set_input_transition` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_input_transition [-h | -help] [-long_help] [-clock` *<name>* `] [-clock_fall] [-fall] [-max] [-min] [-rise]` *<transition>* *<ports>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clock` *<name>*	Clock name
	`-clock_fall`	Specifies that input delay is relative to the falling edge of the clock
	`-fall`	Specifies the falling output delay at the port
	`-max`	Applies value as maximum data required time
	`-min`	Applies value as minimum data required time
	`-rise`	Specifies the rising output delay at the port
	<transition>	Time value
	<ports>	Collection or list of input or bidir ports

Description

```
This constraint does not affect calculations performed by the Timing
Analyzer. It only affects PrimeTime analysis.
If you set this constraint in the Timing Analyzer the constraint is
written out to the SDC file when you call write_sdc.
```

continued...

Example Usage	`dni::set_input_transition 50 [all_inputs]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.16. dni::set_max_delay (::quartus::dni_sdc)

The following table displays information for the `dni::set_max_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_max_delay [-h | -help] [-long_help] [-comment <string>] [-fall] [-fall_from <names>] [-fall_through <names>] [-fall_to <names>] [-from <names>] [-ignore_clock_latency] [-reset_path] [-rise] [-rise_from <names>] [-rise_through <names>] [-rise_to <names>] [-through <names>] [-to <names>] <value>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment <string>`	Comment string
	`-fall`	Marks falling delays false, as measured on the path endpoint
	`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-ignore_clock_latency`	Indicates that the launch and capture clock latencies are to be ignored when computing slack on the specified paths
	`-reset_path`	Removes existing point-to-point exception information on the specified paths
	`-rise`	Marks rising delays false, as measured on the path endpoint
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`<value>`	Time Value
Description	`Specifies a maximum delay exception for a given path.` `The maximum delay is similar to changing the setup relationship`	

continued...

```
(latching clock edge - launching clock edge), except that it can be
applied to input or output ports without input or output delays
assigned to them. Maximum delays are always relative to any clock
network delays (if the source or destination is a register) or any
input or output delays (if the source or destination is a port).
Therefore, input delays and clock latencies are added to the data
arrival times. Clock latencies also added to data required times and
output delays are subtracted from data required times.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design. If the -from or -to values are not
specified, the collection is converted automatically into [get_cells
*]. It is worth noting that if the counterpart to the unspecified
collection is a clock collection, it is more efficient to explicitly
specify this collection as a clock collection but only if the clock
collection also generates the desired assignment.

Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or ports
driven by the -to clock. Applying exceptions between a pair of clocks
is more efficient than for specific node to node or node to clock
paths.

If pin names or collections are used, the -from value must be a clock
pin and the -to value must be any non-clock input pin to a register.
Assignments from clock pins or to and from cells applies to all
registers in the cell or driven by the clock pin.

The -through values are collections of pins or nets in the design. An
exception applied through a node in the design applies only to paths
through the specified node. The Timing Analyzer allows you to specify
the -through argument multiple times to describe paths that go through
multiple points. For instance, users can select all paths that go
through node X, and then go through node Y. This helps you narrow down
and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
-from destination nodes. The rise or fall value of the option
indicates that the "from" nodes are driven by the rising or falling
edge of the clock that feeds this node taking into consideration any
logical inversions along the clock path. The "-from" option is the
combination of both rising and falling "from" nodes. If the "from"
collection is a clock collection, the assignment applies to those
nodes that are driven by the respective rising or falling clock edge.

The -rise_to and -fall_to options behave similarly to the "from"
options described previously. These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value taking into consideration any logical
inversions that are along the clock path.

The values of the -from, -to, -through, and other similar options are
either collections or a Tcl list of wildcards used to create
collections of appropriate types.

Note -rise, -fall, -rise_through, -fall_through, -ignore_clock_latency
and -reset_path options are not supported yet.
```

Example Usage

```
# Apply a 10ns max delay between two unrelated clocks
set_max_delay -from [get_clocks clkA] -to [get_clocks clkB] 10.000

# Apply a 2ns max delay for an input port (TSU)
set_max_delay -from [get_ports in[*]] -to [get_cells *] 2.000

# Apply a 2ns max delay for an output port (TCO)
set_max_delay -from [get_cells *] -to [get_ports out[*]] 2.000

# Apply a 2ns max delay for an input port to an output port (TPD)
set_max_delay -from [get_ports in[*]] -to [get_ports out[*]] 2.000

# Apply a 2ns max delay for an input port only to nodes driven by
# the rising edge of clock CLK
set_max_delay -from [get_ports in[*]] -rise_to [get_clocks CLK] 2.000
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.8.17. dni::set_max_skew (::quartus::dni_sdc)

The following table displays information for the `dni::set_max_skew` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_max_skew [-h | -help] [-long_help] [-fall_from_clock <names>] [-fall_to_clock <names>] [-from <names>] [-from_clock <names>] [-get_skew_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period>] [-rise_from_clock <names>] [-rise_to_clock <names>] [-skew_value_multiplier <multiplier>] [-to <names>] [-to_clock <names>] [<skew>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-get_skew_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period>`	Compute skew constraint as a multiple of the clock period
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-skew_value_multiplier <multiplier>`	Value by which the clock period should be multiplied to compute skew requirement
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	<skew>	Required skew
Description		

```
Use the set_max_skew constraint to perform maximum
allowable skew analysis between sets of registers or
ports. In order to constrain skew across multiple
paths, all such paths must be defined within a single
set_max_skew constraint. The set_max_skew timing
constraint is not affected by the set_max_delay,
set_min_delay, and set_multicycle_path constraints,
but is affected by the set_clock_groups -exclusive
constraint. Paths between exclusive clocks are not
analyzed for skew, and no two paths are compared for
skew if their clocks are exclusive to each other.
However, paths whose clocks are asynchronous are
still analyzed for skew.

Legal values for the -from and -to options are
collections of clocks, ports, pins, or cells in a design.

Applying maximum skew constraints between clocks
applies the constraint from all register or ports
driven by the clock specified with the -from option to
all registers or ports driven by the clock specified
with the -to option.
```

continued...

	If pin names or collections are used, the -from value must be a clock pin and the -to value must be any non-clock input pin to a register. Assignments from clock pins or to and from cells apply to all registers contained in the cell or driven by the clock pin. Similarly, -to and -from partition specifications apply to all registers in the specified partition. Max skew analysis includes data arrival times, clock arrival times, register micro parameters, clock uncertainty, on-die variation and ccpp removal. Use -get_skew_value_from_clock_period to set the skew requirement to be equal to the launching or latching clock period, or whichever of the two has a smaller period. If -skew_value_multiplier is used, the requirement is multiplied by that value. If this option is used, then the positional skew option may not be set. If the set of skew paths is clocked by more than one clock, the Timing Analyzer will use the one with the smallest period to compute the skew constraint. When this constraint is used, results of max skew analysis are displayed in the Report Max Skew (report_max_skew) report from the Timing Analyzer. Since skew is defined between two or more paths, no results are displayed if the -from/-from_clock and -to/-to_clock filters satisfy fewer than two paths.
Example Usage	`# Constrain the skew on an input port to all registers it feeds` `set_max_skew -from [get_ports din] 0.200` `# Constrain the skew on output bus dout[*]` `set_max_skew -to [get_ports dout\[*\]] 0.200` `# Constrain skew to be less than 90% of the period of any clock in the source` `# register set` `set_max_skew -to [get_cells inst1|*] -get_skew_value_from_clock_period src_clock_period -` `skew_value_multiplier 0.900` `# Report the results of max skew assignments` `report_max_skew -panel_name "Report Max Skew" -npaths 10 -detail path_only`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.18. dni::set_max_time_borrow (::quartus::dni_sdc)

The following table displays information for the `dni::set_max_time_borrow` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_max_time_borrow [-h | -help] [-long_help] <value> <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<value>	Time Value
	<targets>	Collection or list of latches
Description	Specifies the maximum borrowed time for level-sensitive latches. The actual borrowed time will be determined automatically, but will never exceed the amount you specify. For any latches without a set_max_time_borrow constraint, no limit will apply (except for the physical limit of what is possible on the device, as described below). Time borrowing is specified with respect to the earliest possible time a signal can be clocked into the latch node. For example, for a positive latch, if the earliest possible arrival time of the rising clock edge is 1.025ns, then a signal that has an arrival time of 1.035ns (where this arrival time already includes the micro-setup time	

continued...

	of the latch) will require at least 0.010ns of time borrowing. Regardless of how the borrowed time is determined (automatically without a limit or automatically with a set_max_time_borrow constraint), the borrowed time can never exceed what is physically possible to borrow on the device. The maximum amount that can be borrowed is the period of time when the latch is open (e.g. half the clock period if the clock has a 50% duty cycle), but this time is reduced by clock propagation time spread and clock uncertainty between the latch-opening and latch-closing clock edges, and is further reduced by the closing-edge setup time of the latch. Some of these factors vary from corner to corner, as well as from clock to clock (if multiple clocks drive the latch). Time borrowing analysis will only occur in the Timing Analysis (Signoff) stage, or when manually running the Timing Analyzer. The Fitter will not utilize time borrowing information and will assume zero time borrowed. Thus, the use of level-sensitive latches with high-speed clocks is not recommended, unless other constraints (such as set_max_delay) are manually set to ensure optimal Fitter behavior. The targets of this command must be level-sensitive latches (all other targets will be ignored). The targets can be specified as either a collections or a Tcl list of wildcards used to create collections of appropriate types.
Example Usage	`# Borrow at most 3ns at all "lat*" latches:` `set_max_time_borrow 3 [get_cells lat*]`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.19. dni::set_min_delay (::quartus::dni_sdc)

The following table displays information for the `dni::set_min_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_min_delay [-h | -help] [-long_help] [-comment` *<string>* `] [-fall] [-fall_from` *<names>* `] [-fall_through` *<names>* `] [-fall_to` *<names>* `] [-from` *<names>* `] [-ignore_clock_latency] [-reset_path] [-rise] [-rise_from` *<names>* `] [-rise_through` *<names>* `] [-rise_to` *<names>* `] [-through` *<names>* `] [-to` *<names>* `]` *<value>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment` *<string>*	Comment string
	`-fall`	Marks falling delays false, as measured on the path endpoint
	`-fall_from` *<names>*	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through` *<names>*	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to` *<names>*	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from` *<names>*	Valid sources (string patterns are matched using Tcl string matching)
	`-ignore_clock_latency`	Indicates that the launch and capture clock latencies are to be ignored when computing slack on the specified paths

continued...

	`-reset_path`	Removes existing point-to-point exception information on the specified paths
	`-rise`	Marks rising delays false, as measured on the path endpoint
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	<value>	Time Value

Description

```
Specifies a minimum delay exception for a given path.

The minimum delay is similar to changing the hold relationship
(launching clock edge - latching clock edge), except that it can be
applied to input or output ports without input or output delays
assigned to them. Minimum delays are always relative to any clock
network delays (if the source or destination is register) or any input
or output delays (if the source or destination is a port). Therefore,
input delays and clock latencies are added to the data arrival times.
Clock latencies also added to data required times and output delays
are subtracted from data required times.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design. If the -from or -to values are not
specified, the collection is converted automatically into [get_cells
*]. It is worth noting that if the counterpart of the unspecified
collection is a clock collection, it is more efficient to explicitly
specify this collection as a clock collection, but only if the clock
collection also generates the desired assignment.

Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or ports
driven by the -to clock. Also, applying exceptions between a pair of
clocks is more efficient than for specific node to node or node to
clock paths.

If pin names or collections are used, the -from value must be a clock
pin and the -to value must be any non-clock input pin to a register.
Assignments from clock pins or to and from cells applies to all
registers in the cell or driven by the clock pin.

The -through values are collections of pins or nets in the design. An
exception applied through a node in the design applies only to paths
through the specified node. The Timing Analyzer allows you to specify
the -through argument multiple times to describe paths that go through
multiple points. For instance, users can select all paths that go
through node X, and then go through node Y. This helps you narrow down
and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
destination nodes specified using the -from option. The rise or fall
value of the option indicates that the "from" nodes are driven by the
rising or falling edge of the clock that feeds this node taking into
consideration any logical inversions along the clock path. The -from
option is the combination of both rising and falling "from" nodes. If
the -from collection is a clock collection, the assignment applies to
those nodes that are driven by the respective rising or falling clock
edge.

The -rise_to and -fall_to options behave similarly to the "from"
options described previously. These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value taking into consideration any logical
inversions that are along the clock path.

The values of the -from, -to, -through, and other similar options are
either collections or a Tcl list of wildcards used to create
collections of appropriate types.
```

continued...

	Note -rise, -fall, -rise_through, -fall_through, -ignore_clock_latency and -reset_path options are not supported yet.
Example Usage	# Apply a 0ns min delay between two unrelated clocks set_min_delay -from [get_clocks clkA] -to [get_clocks clkB] 0.000 # Apply a 0ns min delay for an input port (TH) set_min_delay -from [get_ports in[*]] -to [get_cells *] -.000 # Apply a 0.5ns min delay for an output port (MIN_TCO) set_min_delay -from [get_cells *] -to [get_ports out[*]] 0.500 # Apply a 0.5ns min delay for an input port to an output port (MIN_TPD) set_min_delay -from [get_ports in[*]] -to [get_ports out[*]] 0.500 # Apply a 0.5ns min delay for an input port only to nodes driven by # the falling edge of clock CLK set_max_delay -from [get_ports in[*]] -fall_to [get_clocks CLK] 0.500

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.20. dni::set_multicycle_path (::quartus::dni_sdc)

The following table displays information for the `dni::set_multicycle_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_multicycle_path [-h | -help] [-long_help] [-comment <string>] [-end] [-fall] [-fall_from <names>] [-fall_through <names>] [-fall_to <names>] [-from <names>] [-hold] [-reset_path] [-rise] [-rise_from <names>] [-rise_through <names>] [-rise_to <names>] [-setup] [-start] [-through <names>] [-to <names>] <value>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment <string>`	Comment string
	`-end`	Specifies that the multicycle is relative to the destination clock waveform (default)
	`-fall`	Marks falling delays false, as measured on the path endpoint
	`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-hold`	Specifies that the multicycle value applies to clock hold or removal checks
	`-reset_path`	Removes existing point-to-point exception information on the specified paths
	`-rise`	Marks rising delays false, as measured on the path endpoint

continued...

	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Specifies that the multicycle value applies to clock setup or recovery checks (default)
	`-start`	Specifies that the multicycle is relative to the source clock waveform
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	<value>	Number of clock cycles

Description

```
Specifies a multicycle exception for a given set of paths.

Multicycles can be specified relative to the source clock (-start) or
destination clock (-end). This is useful when the source clock and
destination clock are operating at different frequencies. For
example, if the source clock is twice as fast (half period) as the
destination clock, a -start multicycle of 2 is usually required.

Hold multicycles (-hold) are computed relative to setup multicycles
(-setup). The value of the hold multicycle represents the number clock
edges away from the default hold multicycle. The default hold
multicycle value is 0.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design. If the -from or -to values are not
specified, the collection is converted automatically into [get_cells
*]. It is worth noting that if the counterpart of the unspecified
collection is a clock collection, it is more efficient to explicitly
specify this collection as a clock collection but only if the clock
collection also generates the desired assignment.

Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or ports
driven by the -to clock. Also, applying exceptions between a pair of
clocks is more efficient than for specific node to node or node to
clock paths.

If pin names or collections are used, the -from value must be a clock
pin and the -to value must be any non-clock input pin to a register.
Assignments from clock pins or to and from cells applies to all
registers in the cell or driven by the clock pin.

The -through values are collections of pins or nets in the design. An
exception applied through a node in the design applies only to paths
through the specified node. The Timing Analyzer allows you to specify
the -through argument multiple times to describe paths that go through
multiple points. For instance, users can select all paths that go
through node X, and then go through node Y. This helps you narrow down
and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
"-from" destination nodes. The rise or fall value of the option
indicates that the "from" nodes are driven by the rising or falling
edge of the clock that feeds this node taking into consideration any
logical inversions along the clock path. The "-from" option is the
combination of both rising and falling "from" nodes. If the "from"
collection is a clock collection, the assignment applies to those
nodes that are driven by the respective rising or falling clock edge.

The -rise_to and -fall_to options behave similarly to the "from"
options described previously. These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value taking into consideration any logical
inversions that are along the clock path.

The values of the -from, -to, -through, and similar options are either
collections or a Tcl list of wildcards used to create collections of
appropriate types.
```

continued...

	Note -rise, -fall, -rise_through, -fall_through and -reset_path options are not supported yet.
Example Usage	create_clock -period 10.000 -name CLK [get_ports clk] create_generated_clock -divide_by 2 -source [get_ports clk] -name CLKDIV2 [get_cells clkdiv] # Apply a source multicycle of 2 with a hold multicycle of 1 for all # paths from the CLK domain to the CLKDIV2 domain. set_multicycle_path -start -setup -from [get_clocks CLK] -to [get_clocks CLKDIV2] 2 set_multicycle_path -start -hold -from [get_clocks CLK] -to [get_clocks CLKDIV2] 1 # Apply a multicycle constraint of 3 (with a default hold multicycle of 0) for a # specific path in the design. set_multicycle_path -end -setup -from [get_pins rega|clk] -to [get_pins regb|*] 3 # Apply a multicycle constraint of 2 to a given cell, except for the reset pin. set_multicycle_path -end -setup -to [get_cells regb] 2 set_multicycle_path -end -setup -to [get_pins regb|aclr] 1 #Apply a multicycle constraint of 3 rising from a clock and falling to a node set_multicycle_path -end -setup -rise_from [get_clocks CLK] -fall_to [get_pins regb|datab] 3

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.21. dni::set_net_delay (::quartus::dni_sdc)

The following table displays information for the `dni::set_net_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_net_delay [-h | -help] [-long_help] [-allow_ipin_as_to_target] -from` *<names>* `[-get_value_from_clock_period` *<src_clock_period|dst_clock_period|min_clock_period|max_clock_period>* `] [-max] [-min] [-to` *<names>* `] [-value_multiplier` *<multiplier>* `] [` *<delay>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-allow_ipin_as_to_target`	Allows input pins as the -to target
	`-from` *<names>*	Valid source pins, ports, or nets (string patterns are matched using Tcl string matching)
	`-get_value_from_clock_period` *<src_clock_period|dst_clock_period|min_clock_period|max_clock_period>*	Compute net delay constraint as a multiple of the clock period
	`-max`	Specifies maximum delay
	`-min`	Specifies minimum delay
	`-to` *<names>*	Valid destination pins, ports, or nets (string patterns are matched using Tcl string matching)
	`-value_multiplier` *<multiplier>*	Value by which the clock period should be multiplied to compute net delay requirement
	<delay>	Required delay
Description	Use the set_net_delay command to query the net delays and perform minimum or maximum timing analysis across nets. The -from and -to options can be string patterns or pin, port, or net collections. When a pin or net collection is used, the collection should include output pins or nets.	

continued...

```
If the -to option is unused or if the -to filter is an
"*" character, all the output pins and registers on
timing netlist become valid destination points.

When you use the -min option, slack is calculated by
looking at the minimum delay on the edge. If you use the
-max option, slack is calculated with the maximum edge
delay.

Use -get_value_from_clock_period to set the net delay requirement
to be equal to the launching or latching clock period,
or whichever of the two has a smaller or larger period.
If -value_multiplier is used, the requirement will be multiplied by
that value.  If the set of nets is clocked by more than one clock,
the Timing Analyzer will use the one with the smallest period to compute
the constraint for a -max constraint, and the largest period for
a -min constraint.  If there are no clocks clocking the endpoints
of the net (e.g. if the endpoints of the nets are not registers or
constrained ports), then the net delay constraint will be ignored.
```

Example Usage

```
project_open my_project
create_timing_netlist
read_sdc
update_timing_netlist

# add min delay constraint
set_net_delay -min 0.160 -from [get_pins inst9|combout] -to [get_pins *|dataf]

# add max delay constraint
set_net_delay -max 0.500 -from [get_pins inst8|combout]

# this is same as the previous call
set_net_delay -max 0.500 -from inst8|combout -to *

# Require net delay to be at most 90% of the period of the clock driving the inst9 register
set_net_delay -max -get_value_from_clock_period dst_clock_period -value_multiplier 0.9 -from
[get_pins inst8|combout] -to [get_pins inst9|q]

update_timing_netlist

report_net_delay -panel "Net Delay"
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.8.22. dni::set_operating_conditions (::quartus::dni_sdc)

The following table displays information for the `dni::set_operating_conditions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_operating_conditions [-h | -help] [-long_help] [-analysis_type` *<bc_wc|on_chip_variation|single>* `] [-condition` *<condition>* `] [-force_dat] [-grade` *<c|i|m|e|a>* `] [-library` *<lib>* `] [-max` *<max_condition>* `] [-max_library` *<max_lib>* `] [-max_phys` *<max_proc>* `] [-min` *<min_condition>* `] [-min_library` *<min_lib>* `] [-min_phys` *<min_proc>* `] [-model` *<fast|slow>* `] [-object_list` *<object_list>* `] [-speed` *<speed>* `] [-temperature` *<value_in_C>* `] [-voltage` *<value_in_mV>* `] [` *<list_of_operating_conditions>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-analysis_type` *<bc_wc|on_chip_variation|single>*	Specifies how to use the operating conditions
	`-condition` *<condition>*	Specifies conditions that define environmental characteristics to use during maximum and minimum delay analysis
	`-force_dat`	Option to force delay annotation (only done when selecting an unanalyzed corner)

continued...

	`-grade` *<c|i|m|e|a>*	Option to specify temperature grade
	`-library` *<lib>*	Specifies the library containing definitions of the operating conditions for both maximum and minimum delay analysis
	`-max` *<max_condition>*	Specifies the operating condition to use for maximum delay analysis
	`-max_library` *<max_lib>*	Specifies the library containing definitions of the operating conditions for maximum delay analysis
	`-max_phys` *<max_proc>*	Specifies the name of the process resource to search for the resistance and capacitance values for maximum delay analysis
	`-min` *<min_condition>*	Specifies the operating condition to use for minimum delay analysis
	`-min_library` *<min_lib>*	Specifies the library containing definitions of the operating conditions for minimum delay analysis
	`-min_phys` *<min_proc>*	Specifies the name of the process resource to search for the resistance and capacitance values for minimum delay analysis
	`-model` *<fast|slow>*	Option to specify timing model
	`-object_list` *<object_list>*	Specifies the cells or ports on which to set operating conditions
	`-speed` *<speed>*	Speed grade
	`-temperature` *<value_in_C>*	Operating temperature
	`-voltage` *<value_in_mV>*	Operating voltage
	<list_of_operating_conditions>	list or collection of Operating conditions Tcl objects or names
Description		

```
Use this command to specify operating conditions different from the
initial conditions used to create the timing netlist. When a timing
model is not specified, the slow model is used.

Voltage and temperature options must be used together. These two
options are not available for all devices. The
get_available_operating_conditions command returns the list of
available operating conditions for your device.

Use the -speed option to analyze the design at a different speed grade
of the selected device.

Use the -grade option to analyze the design at a different temperature
grade. This option is provided to support what-if analysis and is not
recommended for final sign-off analysis.

By default, delay annotation is skipped if previously performed. Use
-force_dat to rerun delay annotation.

Note -analysis_type, -min, -max, -min_library, -max_library,
-min_phys, -max_phys, -library, -object_list and condition options are
not supported yet,
```

Example Usage

```
#do report timing for different operating conditions one by one
foreach_in_collection op [get_available_operating_conditions] {
    set_operating_conditions $op
    update_timing_netlist
    report_timing
}

#set aggregated report timing for all operating conditions when corner aggregation is enabled
set_operating_conditions [get_available_operating_conditions]
report_timing

#manually set operating conditions
set_operating_conditions -model fast -temperature 85 -voltage 1200
update_timing_netlist
```

continued...

	`` #change device speed grade and set operating conditions set_operating_conditions -speed 3 -model slow -temperature 0 -voltage 1100 update_timing_netlist ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.23. dni::set_output_delay (::quartus::dni_sdc)

The following table displays information for the `dni::set_output_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_output_delay [-h | -help] [-long_help] [-add_delay] -clock <name> [-clock_fall] [-fall] [-max] [-min] [-network_latency_included] [-reference_pin <name> ] [-rise] [-source_latency_included] <delay> <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add_delay`	Create additional delay constraint instead of overriding previous constraints
	`-clock <name>`	Clock name
	`-clock_fall`	Specifies output delay relative to the falling edge of the clock
	`-fall`	Specifies the falling output delay at the port
	`-max`	Applies value as maximum data required time
	`-min`	Applies value as minimum data required time
	`-network_latency_included`	Specifies that input delay includes added network latency
	`-reference_pin <name>`	Specifies a pin or port in the design to which the output delay is relative
	`-rise`	Specifies the rising output delay at the port
	`-source_latency_included`	Specifies input delay already includes added source latency
	<delay>	Time value
	<targets>	Collection or list of output ports
Description	``` Specifies the data required times at the specified output ports relative the clock specified by the -clock option. The clock must refer to a clock name in the design. Output delays can be specified relative to the rising edge (default) or falling edge (-clock_fall) of the clock. Output delays can be specified relative to a pin or a port (-reference_pin) in the clock network. Clock arrival times to the reference pin or port are added to the data required time. Output delays can include clock source latency. By default the clock source latency of the related clock is added to the output delay value, but when the -source_latency_included option is specified, the clock source latency is not added because it was factored into the output delay value. The maximum output delay (-max) is used for clock setup checks or recovery checks and the minimum output delay (-min) is used for clock hold checks or removal checks. If only one of -min and -max (or ```	

continued...

```
neither) is specified for a given port, the same value is used for
both.

Separate rising (-rise) and falling (-fall) required times at the port
can be specified. If only one of -rise and -fall are specified for a
given port, the same value is used for both.

By default, set_output_delay removes any other output delays to the port
except for those with the same -clock, -clock_fall, and -reference_pin
combination. Multiple output delays relative to different clocks, clock
edges, or reference pins can be specified using the -add_delay option.
Future output delays relative to different clocks, clock edges, or
reference pins that do not specify the -add_delay option will still
remove previous output delays that specified the -add_delay option.

Note that -network_latency_included is not supported yet.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.
```

Example Usage

```
# Simple output delay with the same value for min/max and rise/fall:
# 1) set on ports with names of the form myout*
set_output_delay -clock clk 0.5 [get_ports myout*]
# 2) set on all output ports
set_output_delay -clock clk 0.5 [all_outputs]

# Output delay with respect to the falling edge of clock
set_output_delay -clock clk -clock_fall 0.5 [get_ports myout*]

# Output delays for different min/max and rise/fall combinations
set_output_delay -clock clk -max -rise 0.5 [get_ports myout*]
set_output_delay -clock clk -max -fall 0.4 [get_ports myout*]
set_output_delay -clock clk -min -rise 0.4 [get_ports myout*]
set_output_delay -clock clk -min -fall 0.3 [get_ports myout*]

# Adding multiple output delays with respect to more than one clock
set_output_delay -clock clkA -min 0.2 [get_ports myout*]
set_output_delay -clock clkA -max 0.8 [get_ports myout*]
set_output_delay -clock clkA -clock_fall 0.6 [get_ports myout*] -add_delay
set_output_delay -clock clkB -min 1.1 [get_ports myout*] -add_delay
set_output_delay -clock clkB -max 1.5 [get_ports myout*] -add_delay

# This is a common mistake where output delays are accidentally removed
set_output_delay -clock clkA -min 0.2 [get_ports myout*]
set_output_delay -clock clkB -min 1.1 [get_ports myout*] -add_delay
# The following removes the clkB entry. You need to always use -add_delay
#     when more than one clock, clock_fall or reference_pin exists
set_output_delay -clock clkA -max 0.8 [get_ports myout*]
# The following removes the clkA entry. You need to always use -add_delay
#     when more than one clock, clock_fall or reference_pin exists
set_output_delay -clock clkB -max 1.5 [get_ports myout*]

# Specifying an output delay relative to an external clock output port
set_output_delay -clock clk -reference_pin [get_ports clkout] 0.8 [get_ports myout*]

# Specifying an output delay relative to the clock pin of a register
set_output_delay -clock clk -reference_pin [get_pins regA|clk] 0.8 [get_ports myout*]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.8.24. dni::set_sense (::quartus::dni_sdc)

The following table displays information for the `dni::set_sense` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_sense [-h | -help] [-long_help] [-clocks <clock_list> ] [-negative] [-positive] [-pulse <pulse_type> ] [-stop_propagation] [-type <type> ] <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clocks <clock_list>`	Clock objects to which the assignment applies

continued...

	`-negative`	Apply negative unate sense
	`-positive`	Apply positive unate sense
	`-pulse` *<pulse_type>*	Specifies the type of pulse clock applied to all pins in the target variable with respect to clock source
	`-stop_propagation`	Stops the propagation of specified clocks from the specified pins or cell timing arcs
	`-type` *<type>*	Specifies whether the type of sense being applied refers to clock networks or data networks
	<targets>	List or collection of targets
Description		

```
Restrict unateness at a pin on a clock path, with respect to the clock
source.

If the -clocks option is used, the assignment will only apply to
analysis of the specified clock domains. Otherwise, it applies to all
clocks passing through the given pins.

If the specified sense does not exist at the given pin, the assignment
is ignored.

Note -type, -pulse and -stop_propagation options are not supported.
```

Example Usage

```
set_sense -positive Mux|combout
set_sense -negative -clocks [get_clocks clk] XOR|combout
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.8.25. dni::set_timing_derate (::quartus::dni_sdc)

The following table displays information for the `dni::set_timing_derate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::dni_sdc on page 158	
Syntax	`dni::set_timing_derate [-h | -help] [-long_help] [-cell_check] [-cell_delay] [-clock] [-data] [-early] [-fall] [-late] [-max] [-min] [-net_delay] [-operating_conditions` *<operating_conditions>* `] [-pocvm_coefficient_scale_factor] [-pocvm_guardband] [-rise]` *<derate_value>* `[` *<cells>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cell_check`	Applies the derating value to the setup and hold time requirements of cells
	`-cell_delay`	Specifies that derating factors are only to apply to cell delays
	`-clock`	Applies the derating value only to elements in the clock network
	`-data`	Applies the derating value only to elements in the data network
	`-early`	Specifies the minimum derating factor. This factor specifies how early the signal can arrive
	`-fall`	Applies the derating value only to the delays of paths that have a falling transition at the specified objects

continued...

	-late		Specifies the maximum derating factor. This factor specifies how late the signal can arrive
	-max		Applies the derating value only to the maximum operating condition
	-min		Applies the derating value only to the minimum operating condition
	-net_delay		Specifies that derating factors are only to apply to net delays
	-operating_conditions <*operating_conditions*>		Operating conditions Tcl object
	-pocvm_coefficient_scale_factor		This option applies only to the parametric on-chip variation (POCV) context
	-pocvm_guardband		This option applies only to the parametric on-chip variation (POCV) context
	-rise		Applies the derating value only to the delays of paths that have a rising transition at the specified objects
	<*derate_value*>		Timing derate value
	<*cells*>		List of cell type objects
Description	Sets the global derate factors for the current design. The maxmimum and minimum delays of all timing arcs in the design are multiplied by the factors specified with the -late and -early options respectively. Only positive derate factors are allowed. If neither the -cell_delay nor -net_delay option is used, the derating factors apply to both cell and net delays. For net delay derates, the derate factor is applied to nets driven by matching cells. Specifying a derate value of less than 1.0 for the -late option or a derate value of greater than 1.0 for the -early option reduces delay pessimisim, which might lead to optimistic results from timing analysis. The effect of set_timing_derate command is deferred until the next time update_timing_netlist is called. To reset derate factors to original values, use the reset_timing_derate command. This assignment is for timing analysis only, and is not considered during timing-driven compilation. Note -min, -max, -data, -clock, -rise, -fall, -cell_check, -pocvm_guardband and -pocvm_coefficient_scale_factor options are not supported yet.		
Example Usage	set_timing_derate -early 0.9 [get_cells *] set_timing_derate -late 1.1 [get_cells *]		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9. ::quartus::drc

The following table displays information for the **::quartus::drc** Tcl package:

Tcl Package and Version	::quartus::drc 1.0
Description	This package contains no general description.

continued...

Availability	This package is available for loading in the following executables: qacv qpro quartus quartus_cdb quartus_fit quartus_sta quartus_syn quartus_tlg
Tcl Commands	drc::add_check_op (::quartus::drc) on page 192 drc::add_check_parameter (::quartus::drc) on page 193 drc::add_object (::quartus::drc) on page 194 drc::add_object_with_properties (::quartus::drc) on page 194 drc::add_property (::quartus::drc) on page 195 drc::add_rule (::quartus::drc) on page 196 drc::add_rule_violation (::quartus::drc) on page 197 drc::add_violation_record (::quartus::drc) on page 197 drc::add_waiver (::quartus::drc) on page 198 drc::check_design (::quartus::drc) on page 198 drc::delete_waivers (::quartus::drc) on page 199 drc::get_objects (::quartus::drc) on page 199 drc::get_option (::quartus::drc) on page 200 drc::get_property (::quartus::drc) on page 200 drc::get_stage_info (::quartus::drc) on page 201 drc::get_waivers (::quartus::drc) on page 202 drc::list_properties (::quartus::drc) on page 202 drc::report_waivers (::quartus::drc) on page 203 drc::set_option (::quartus::drc) on page 203 drc::set_property (::quartus::drc) on page 204 drc::should_run_drc (::quartus::drc) on page 204 drc::update_check_op (::quartus::drc) on page 205 drc::update_rule (::quartus::drc) on page 205

3.1.9.1. drc::add_check_op (::quartus::drc)

The following table displays information for the `drc::add_check_op` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191	
Syntax	`drc::add_check_op [-h | -help] [-long_help] -bind_to_tcl_execute` *<per_rule|per_operation>* `[-device_families` *<device_families>* `] -exec_proc_name` *<exec_proc_name>* `[-executables` *<executables>* `] [-finalize_proc_name` *<finalize_proc_name>* `] -name` *<name>* `[-setup_proc_name` *<setup_proc_name>* `] [-stages` *<stages>* `] [-tcl_proc_source` *<tcl_proc_source>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-bind_to_tcl_execute` *<per_rule|per_operation>*	Check operation to rule binding mode
	`-device_families` *<device_families>*	Device family, check operation is valid for
	`-exec_proc_name` *<exec_proc_name>*	Check Operation execution proc name
	`-executables` *<executables>*	Allowed executables, check operation could be invoked from
	`-finalize_proc_name` *<finalize_proc_name>*	Check Operation cleanup proc name
	`-name` *<name>*	Check Operation name
	`-setup_proc_name` *<setup_proc_name>*	Check Operation intialization proc name
	`-stages` *<stages>*	Allowed stages, check operation could be invoked from

continued...

	`-tcl_proc_source` *<tcl_proc_source>*		Check operation proc source file name
Description	`Add Check Operation .`		
Example Usage	`add_check_op -name {TEST} -bind_to_tcl_execute {per_operation} -exec_proc_name {TEST_EXEC_PROC}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.2. drc::add_check_parameter (::quartus::drc)

The following table displays information for the `drc::add_check_parameter` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::add_check_parameter [-h | -help] [-long_help] -check_op` *<check_op>* `[-is_user` *<is_user>* `] [-param_description` *<param_description>* `] -param_name` *<param_name>* `[-param_range` *<param_range>* `] -param_value` *<param_value>* `-value_type` *<integer|double|string|string_list|bool>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-check_op` *<check_op>*		Check Operation name
	`-is_user` *<is_user>*		Is parameter user visible
	`-param_description` *<param_description>*		Parameter description
	`-param_name` *<param_name>*		Parameter name
	`-param_range` *<param_range>*		Legal range for parameter
	`-param_value` *<param_value>*		Default value for parameter
	`-value_type` *<integer|double|string|string_list|bool>*		Parameter value type integer|double|string|string_list|bool
Description	`Add Check Operation .`		
Example Usage	`add_check_parameter -check_op {TEST} -param_name {TEST_PARAM} -value_type {string} -param_value {TEST_VALUE}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.3. drc::add_object (::quartus::drc)

The following table displays information for the `drc::add_object` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::add_object [-h | -help] [-long_help] [-bind_to_tcl_execute <bind_to_tcl_execute> ] [-category <category> ] [-name <name> ] [-number <number> ] [-parent <parent> ] -type <type>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-bind_to_tcl_execute <bind_to_tcl_execute>`	Specifies the Tcl execution mode of the new check_operation DRC object (none / per_rule / per_operation).	
	`-category <category>`	The category of the new DRC object.	
	`-name <name>`	The name of the new check_operation/rule_set DRC object.	
	`-number <number>`	The numeric part of the new DRC object's name.	
	`-parent <parent>`	The parent object of the new violation_record DRC object.	
	`-type <type>`	The type of the new DRC object.	
Description	`Adds a new DRC object.`		
Example Usage	`drc::add_object -type rule -category CLK -number 1`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to add DRC object.
	TCL_ERROR	1	ERROR: Invalid object type '<*string*>'.
	TCL_ERROR	1	ERROR: Category is needed for finding RULE or VIOLATION.
	TCL_ERROR	1	ERROR: Object name is needed.
	TCL_ERROR	1	ERROR: Object number is needed.
	TCL_ERROR	1	ERROR: Parent object is needed for violation record.

3.1.9.4. drc::add_object_with_properties (::quartus::drc)

The following table displays information for the `drc::add_object_with_properties` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191	
Syntax	`drc::add_object_with_properties [-h | -help] [-long_help] [-bind_to_tcl_execute <bind_to_tcl_execute> ] [-category <category> ] [-name <name> ] [-number <number> ] [-parent <parent> ] -properties <properties> -type <type>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-bind_to_tcl_execute` <*bind_to_tcl_execute*>	Specifies the Tcl execution mode of the new check_operation DRC object (none / per_rule / per_operation).
	`-category` <*category*>	The category of the new DRC object.
	`-name` <*name*>	The name of the new check_operation/rule_set DRC object.
	`-number` <*number*>	The numeric part of the new DRC object's name.
	`-parent` <*parent*>	The parent object of the new violation_record DRC object.
	`-properties` <*properties*>	The property list of the new violation_record DRC object.
	`-type` <*type*>	The type of the new DRC object.
Description	`Adds a new DRC object with properties.`	
Example Usage	`drc::add_object_with_properties -type violation_record -name <violation> -parent <result_record_id> -properties [list [list "fields" [list "HighFanout_DRV1", "1530"]], [list "fields:1:location_schema" "mod_A|mod_A_1|out"]]` `drc::add_object_with_properties -type violation_record -name <violation> -parent <result_record_id> -properties {{"fields" {"HighFanout_DRV1", "1530"}} {"fields:1:location_schema" "mod_A|mod_A_1|out"}}`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to add DRC object.
	TCL_ERROR	1	ERROR: Invalid object type '<*string*>'.
	TCL_ERROR	1	ERROR: Property list should consist of name-value pairs.
	TCL_ERROR	1	ERROR: Category is needed for finding RULE or VIOLATION.
	TCL_ERROR	1	ERROR: Object name is needed.
	TCL_ERROR	1	ERROR: Object number is needed.
	TCL_ERROR	1	ERROR: Parent object is needed for violation record.

3.1.9.5. drc::add_property (::quartus::drc)

The following table displays information for the `drc::add_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191	
Syntax	`drc::add_property [-h | -help] [-long_help] -name` <*name*> `-object` <*object*> `-value` <*value*> `[-value_type` <*value_type*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name` <*name*>	property name.
	`-object` <*object*>	The DRC object to which the new property belongs.
	`-value` <*value*>	property value.
	`-value_type` <*value_type*>	property value type.
Description	`Add a property (a name/value pair) to a generic DRC object`	

continued...

Example Usage	`drc::add_property -object <DRC object> -name <property name> -value <property value> -value_type <value type>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to add object property.
	TCL_ERROR	1	ERROR: Invalid DRC object '%'s.
	TCL_ERROR	1	ERROR: Invalid property value.
	TCL_ERROR	1	ERROR: Invalid property value type.
	TCL_ERROR	1	ERROR: Property '<*string*>' already exists. Cannot be added.

3.1.9.6. drc::add_rule (::quartus::drc)

The following table displays information for the `drc::add_rule` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::add_rule [-h | -help] [-long_help] -category` *<category>* `-check_operation` *<check_operation>* `[-description` *<description>* `] -id` *<id>* `[-is_default` *<is_default>* `] [-recommendation` *<recommendation>* `] -severity` *<severity>* `[-short_description` *<short_description>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-category` *<category>*		Rule category
	`-check_operation` *<check_operation>*		Check operation associated
	`-description` *<description>*		Detailed description
	`-id` *<id>*		Rule ID
	`-is_default` *<is_default>*		Is Default Rule
	`-recommendation` *<recommendation>*		Detailed recommendation
	`-severity` *<severity>*		Rule Severity
	`-short_description` *<short_description>*		Rule title
Description	`Add Check Operation .`		
Example Usage	`add_rule -category {TEST} -id {TEST_ID} -check_operation {TEST_CHECK_OP} -severity {TEST_SEVERITY}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.7. drc::add_rule_violation (::quartus::drc)

The following table displays information for the `drc::add_rule_violation` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::add_rule_violation [-h | -help] [-long_help] [-argument_spec <argument_spec> ] [-category <category> ] [-format_msg <format_msg> ] [-id <id> ] [-num_args <num_args> ] [-object <rule_object> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-argument_spec <argument_spec>`	Argument specifications {{type name description [sort_method] [order]}...}	
	`-category <category>`	Rule category for violation	
	`-format_msg <format_msg>`	Violation message format	
	`-id <id>`	Rule ID for violation	
	`-num_args <num_args>`	Number of arguments	
	`-object <rule_object>`	Rule object for violation	
Description	`Add Rule Violation.`		
Example Usage	`add_rule_violation`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.8. drc::add_violation_record (::quartus::drc)

The following table displays information for the `drc::add_violation_record` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::add_violation_record [-h | -help] [-long_help] -parent <parent>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-parent <parent>`	Parent result record for violation	
Description	`Add Violation Record.`		
Example Usage	`add_violation_record -parent {PARENT_RESULT_RECORD_ID}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.9. drc::add_waiver (::quartus::drc)

The following table displays information for the `drc::add_waiver` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::add_waiver [-h | -help] [-long_help] -description <description> [-no_warn] -owner <owner> -query_string <query_string> -rule_id <rule_id> [-stages <stages> ] [-tag <tag> ] [-timestamp <timestamp> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-description <description>`		Waiver description, explaining the rationale of the waiver, as to why violations waived by this waiver are irrelevant or should be ignored.
	`-no_warn`		Display no warning for adding waiver which will be applied to the next DA run.
	`-owner <owner>`		User ID of creator. Meant for waiver audit trail.
	`-query_string <query_string>`		Query string uses one ore more violation column arguments to define patterns of violations that should be ignored.
	`-rule_id <rule_id>`		Alpha-numeric rule ID pattern to define the scope of this waiver.
	`-stages <stages>`		one or more stage(s) where rules are defined, for which the waiver can be applied to.
	`-tag <tag>`		User-specified tags for tracking different types of violations across the whole project.
	`-timestamp <timestamp>`		The input timestamp when to add the waiver.
Description	`Creates a waiver for a given rule pattern with a specific query string.`		
Example Usage	`::drc::add_waiver -description {a waiver} -rule_id {TMC-20004} -query_string {Slack < -5.14 && "Start Point" =~ 'dir'} -stages [list analysis_planned_quartus_sta Plan]  -tag {test} -owner {owner}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.10. drc::check_design (::quartus::drc)

The following table displays information for the `drc::check_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191	
Syntax	`drc::check_design [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Run the specified rule set on a snapshot for the open project.`	

continued...

Example Usage	`drc::check_design [-rule_set my_ruleset (if missing, the rule set is default)] [-executable quartus_sta (if missing, the default is the current process name)] [-rpt_file <DA DRC ASCII report file path>]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to complete checks.

3.1.9.11. drc::delete_waivers (::quartus::drc)

The following table displays information for the `drc::delete_waivers` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191	
Syntax	`drc::delete_waivers [-h | -help] [-long_help] [-all] [-owner <owner> ] [-query_string <query_string> ] [-rule_id <rule_id> ] [-stages <stages> ] [-tag <tag> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all`	The flag needed explicitly for deleting all the waivers without other arguments
	`-owner <owner>`	User ID of creator. Meant for waiver audit trail.
	`-query_string <query_string>`	Query string uses one ore more violation column arguments to define patterns of violations that should be ignored.
	`-rule_id <rule_id>`	Alpha-numeric rule ID pattern to define the scope of this waiver.
	`-stages <stages>`	one or more stage(s) where rules are defined, for which the waiver can be applied to.
	`-tag <tag>`	User-specified tags for tracking different types of violations across the whole project.
Description	`Delete waivers based on the input arguments`	
Example Usage	`# This delete all the waivers.` `::drc::delete_waivers`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.12. drc::get_objects (::quartus::drc)

The following table displays information for the `drc::get_objects` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191	
Syntax	`drc::get_objects [-h | -help] [-long_help] [-category <category> ] [-name <name> ] [-number <number> ] [-parent <parent> ] -type <type>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-category <category>`		The category of the rule/violation DRC objects to be retrieved.
	`-name <name>`		The literal name of the check_operation/rule_set DRC objects to be retrieved.
	`-number <number>`		The numerical part of the name of the rule/violation DRC objects to be retrieved.
	`-parent <parent>`		The parent object of the violation_record DRC object to be retrieved.
	`-type <type>`		The type of DRC objects to be retrieved.
Description	`Get a list of specific existing DRC objects`		
Example Usage	`drc::get_objects -type rule`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to get DRC objects.
	TCL_ERROR	1	ERROR: Parent object is needed for violation record.

3.1.9.13. drc::get_option (::quartus::drc)

The following table displays information for the `drc::get_option` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::get_option [-h | -help] [-long_help] -name <name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		option name.
Description	`Get option for the DRC system`		
Example Usage	`drc::get_option -name <option name>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: DRC option '<*string*>' is invalid.

3.1.9.14. drc::get_property (::quartus::drc)

The following table displays information for the `drc::get_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::get_property [-h | -help] [-long_help] -name <name> -object <object>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values

continued...

	`-name <name>`		property name.
	`-object <object>`		The DRC object to which the property belongs.
Description	`Get the value of a generic DRC object's property.`		
Example Usage	`drc::get_property -object <DRC object> -name <property name>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid DRC object '%'s.
	TCL_ERROR	1	ERROR: Property '<*string*>' is invalid.

3.1.9.15. drc::get_stage_info (::quartus::drc)

The following table displays information for the `drc::get_stage_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::get_stage_info [-h | -help] [-long_help] [-executable <executable> ] [-rule_set <rule_set> ] [-snapshot <snapshot> ] [-stage <stage> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-executable <executable>`		The executable name to find stages.
	`-rule_set <rule_set>`		The rule set name to find stages.
	`-snapshot <snapshot>`		The snapshot name to find stages.
	`-stage <stage>`		The stage name to get exectuable, snapshot, and rule set.
Description	`The Utility to find stage info in terms of all available stages, and executables and snapshots associate with snapshots. No error info provided.`		
Example Usage	`# Get all the stage names` `::drc::get_stage_info` `# Get executable and snapshot for a stage` `::drc::get_stage_info -stage $stage_name` `# Get stages with the input executable name` `::drc::get_stage_info -executable $executable_name` `# Get stages with the input snapshot name` `::drc::get_stage_info -snapshot $snapshot_name` `# Get stage with the input snapshot and executable name` `::drc::get_stage_info -snapshot $snapshot_name -executable $executable_name`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.16. drc::get_waivers (::quartus::drc)

The following table displays information for the `drc::get_waivers` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::get_waivers [-h | -help] [-long_help] [-owner <owner> ] [-query_string <query_string> ] [-rule_id <rule_id> ] [-stages <stages> ] [-tag <tag> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-owner <owner>`		User ID of creator. Meant for waiver audit trail.
	`-query_string <query_string>`		Query string uses one ore more violation column arguments to define patterns of violations that should be ignored.
	`-rule_id <rule_id>`		Alpha-numeric rule ID pattern to define the scope of this waiver.
	`-stages <stages>`		one or more stage(s) where rules are defined, for which the waiver can be applied to.
	`-tag <tag>`		User-specified tags for tracking different types of violations across the whole project.
Description	`List all the waiver objects found in the memory filtered based on the input arguments.`		
Example Usage	`set waiver_objs [::drc::get_waivers]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.17. drc::list_properties (::quartus::drc)

The following table displays information for the `drc::list_properties` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::list_properties [-h | -help] [-long_help] -object <object>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-object <object>`		The DRC object whose properties are returned as a list of property names.
Description	`Get the list of property names from a generic DRC object`		
Example Usage	`drc::list_properties -object <DRC object>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid DRC object '%'s.

3.1.9.18. drc::report_waivers (::quartus::drc)

The following table displays information for the `drc::report_waivers` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::report_waivers [-h | -help] [-long_help] -file <file> [-force] [-owner <owner> ] [-query_string <query_string> ] [-rule_id <rule_id> ] [-stages <stages> ] [-tag <tag> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-file <file>`	The file where to store all the waivers found in the memory	
	`-force`	the flag which can force to update DESIGN_ASSISTANT_WAIVER_FILE	
	`-owner <owner>`	User ID of creator. Meant for waiver audit trail.	
	`-query_string <query_string>`	Query string uses one ore more violation column arguments to define patterns of violations that should be ignored.	
	`-rule_id <rule_id>`	Alpha-numeric rule ID pattern to define the scope of this waiver.	
	`-stages <stages>`	one or more stage(s) where rules are defined, for which the waiver can be applied to.	
	`-tag <tag>`	User-specified tags for tracking different types of violations across the whole project.	
Description	`Output the definition of all the waivers in the memory to a file filtered based on the input arguments.`		
Example Usage	`::drc::report_waivers -file waivers.dawf`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.19. drc::set_option (::quartus::drc)

The following table displays information for the `drc::set_option` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191	
Syntax	`drc::set_option [-h | -help] [-long_help] -name <name> -value <value>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	option name.
	`-value <value>`	option value.
Description	`Set options for the DRC system`	

continued...

Example Usage	`drc::set_option -name <option name> -value <option value>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: DRC option '<*string*>' is invalid.

3.1.9.20. drc::set_property (::quartus::drc)

The following table displays information for the `drc::set_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::set_property [-h | -help] [-long_help] -name <name> -object <object> -value <value>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		property name.
	`-object <object>`		The DRC object to which the property belongs.
	`-value <value>`		property value.
Description	`Update property in term of name/value pair to a generic DRC object`		
Example Usage	`drc::set_property -object <DRC object> -name <property name> -value <property value>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid DRC object '%'s.
	TCL_ERROR	1	ERROR: Property '<*string*>' is invalid.

3.1.9.21. drc::should_run_drc (::quartus::drc)

The following table displays information for the `drc::should_run_drc` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::should_run_drc [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Determine if DRC should run.`		
Example Usage	`should_run_drc`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.22. drc::update_check_op (::quartus::drc)

The following table displays information for the `drc::update_check_op` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191		
Syntax	`drc::update_check_op [-h | -help] [-long_help] [-device_families` *<device_families>* `] [-executables` *<executables>* `] -name` *<name>* `[-stages` *<stages>* `] [-tcl_proc_source` *<tcl_proc_source>* `]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-device_families` *<device_families>*	Device family, check operation is valid for	
	`-executables` *<executables>*	Allowed executables, check operation could be invoked from	
	`-name` *<name>*	Check operation name	
	`-stages` *<stages>*	Allowed stages, check operation could be invoked from	
	`-tcl_proc_source` *<tcl_proc_source>*	Check operation proc source file name	
Description	`Add Check Operation .`		
Example Usage	`update_check_op -name {TEST}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.9.23. drc::update_rule (::quartus::drc)

The following table displays information for the `drc::update_rule` Tcl command:

Tcl Package and Version	Belongs to ::quartus::drc on page 191	
Syntax	`drc::update_rule [-h | -help] [-long_help] -category` *<category>* `[-description` *<description>* `] [-exec_proc` *<exec_proc>* `] [-finalize_proc` *<finalize_proc>* `] -id` *<id>* `[-is_user` *<is_user>* `] [-param_description` *<param_description>* `] [-param_value` *<param_value>* `] [-parameter` *<parameter>* `] [-recommendation` *<recommendation>* `] [-setup_proc` *<setup_proc>* `] [-short_description` *<short_description>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-category` *<category>*	Rule category
	`-description` *<description>*	Rule description
	`-exec_proc` *<exec_proc>*	Execution proc for the rule
	`-finalize_proc` *<finalize_proc>*	Cleanup proc for the rule
	`-id` *<id>*	Rule ID
	`-is_user` *<is_user>*	Is rule parameter user visible

continued...

	`-param_description` *<param_description>*		Parameter description
	`-param_value` *<param_value>*		Parameter default value
	`-parameter` *<parameter>*		Parameter for the rule
	`-recommendation` *<recommendation>*		Rule recommendation
	`-setup_proc` *<setup_proc>*		Initialization proc for the rule
	`-short_description` *<short_description>*		Rule title
Description	`Update Check Operation.`		
Example Usage	`update_rule -category {TEST} -id {TEST_ID}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10. ::quartus::eco

The following table displays information for the **::quartus::eco** Tcl package:

Tcl Package and Version	::quartus::eco 1.0
Description	This package contains commands to perform Engineering Change Orders on a Post-Fit netlist. ECO compilations are supported on Intel Stratix 10 and Intel Agilex device families.
Availability	This package is available for loading in the following executable: quartus_fit
Tcl Commands	adjust_pll_refclk (::quartus::eco) on page 207 create_new_node (::quartus::eco) on page 207 create_wirelut (::quartus::eco) on page 208 eco::report_partitions (::quartus::eco) on page 221 eco_reroute (::quartus::eco) on page 209 eco_unload_design (::quartus::eco) on page 209 fitter_report_timing (::quartus::eco) on page 210 fitter_timing_summary (::quartus::eco) on page 211 get_available_snapshots (::quartus::eco) on page 211 get_eco_checkpoint (::quartus::eco) on page 212 get_loaded_snapshot (::quartus::eco) on page 212 get_lutmask_equation (::quartus::eco) on page 212 get_node_location (::quartus::eco) on page 213 make_connection (::quartus::eco) on page 213 modify_io_current_strength (::quartus::eco) on page 215 modify_io_delay_chain (::quartus::eco) on page 215 modify_io_slew_rate (::quartus::eco) on page 216 modify_lutmask (::quartus::eco) on page 216 place_node (::quartus::eco) on page 217 remove_connection (::quartus::eco) on page 218 remove_node (::quartus::eco) on page 219 report_connections (::quartus::eco) on page 219 report_legal_locations (::quartus::eco) on page 220 report_nodes_at_location (::quartus::eco) on page 221 report_ports (::quartus::eco) on page 222 report_routing (::quartus::eco) on page 222 report_unplaced_nodes (::quartus::eco) on page 223 restore_eco_checkpoint (::quartus::eco) on page 223 unplace_node (::quartus::eco) on page 224 update_mif_files (::quartus::eco) on page 224

3.1.10.1. adjust_pll_refclk (::quartus::eco)

The following table displays information for the `adjust_pll_refclk` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206	
Syntax	`adjust_pll_refclk [-h | -help] [-long_help] -refclk <refclk_freq> -to <iopll_name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-refclk <refclk_freq>`	New refclk frequency value in MHz
	`-to <iopll_name>`	Instance name of upstream IOPLL to be adjusted
Description	The adjust_pll_refclk command can change IOPLL frequencies by modifying the input reference clock frequency. Assumptions and Limitations: - The original refclk/outclk ratios will be maintained - None of the IOPLLs being reconfigured can generate IP clocks (the frequencies of the LVDS, PhyLite and EMIF clocks cannot change) - Cascaded IOPLLs must be connected directly (no clock gates in between them) - IOPLLs cannot be in 'nondedicated' compensation modes - For all IOPLL outclks duty cycle = 50 and phase shift = 0	
Example Usage	`load_package eco` `adjust_pll_refclk -to "*pll_main*" -refclk 100`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful

3.1.10.2. create_new_node (::quartus::eco)

The following table displays information for the `create_new_node` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206	
Syntax	`create_new_node [-h | -help] [-long_help] -name <node_name> -type <lut|ff>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <node_name>`	Name of the new node
	`-type <lut|ff>`	Node type
Description	The create_new_node command will create a new node with the specified type and return its id. The node name is hierarchical based, and the ECO Fitter will try to infer the name hierarchy. For example, if a node a|b|c|d needs to be created, users should make sure that hierarchy a|b|c exists in the netlist. If a|b|c lies under a partition, the new node will be created under that partition. To connect to the created node, use make_connection command to connect to the LUT's DATAA/B/C/D/E/F input ports, and use modify_lutmask command to change its LUT-mask. The new node needs to be placed with place_node command and then new connections will be routed right after.	
Example Usage	`create_new_node -name eco_new_lut -type lut` `make_connection -from src_a -to eco_new_lut -port DATAA` `make_connection -from src_b -to eco_new_lut -port DATAB`	

continued...

	`make_connection -from eco_new_lut -to dst_reg -port D` `modify_lutmask -to eco_new_lut -eqn A&B` `place_node -name eco_new_lut`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.3. create_wirelut (::quartus::eco)

The following table displays information for the `create_wirelut` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206	
Syntax	`create_wirelut [-h | -help] [-long_help] [-from` *<source_output_net_name>* `] [-from_node` *<from_node>* `] [-from_port` *<from_port>* `] [-location` *<location>* `] -name` *<node_name>* `[-port` *<dest_node_port>* `] [-to` *<dest_node_name>* `] [-to_node` *<to_node>* `] [-to_port` *<to_port>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-from` *<source_output_net_name>*	Source of the connection
	`-from_node` *<from_node>*	ID of the source node
	`-from_port` *<from_port>*	ID of the source port
	`-location` *<location>*	PLACE_REGION assignment type location
	`-name` *<node_name>*	Name of the new node
	`-port` *<dest_node_port>*	Input port name of the destination node
	`-to` *<dest_node_name>*	Name of the destination node
	`-to_node` *<to_node>*	ID of the destination node
	`-to_port` *<to_port>*	ID of the destination port
Description	`The create_wirelut command will create and insert a wire LUT node in the specified connection.` `The ECO Fitter will place the newly created LUT and route the modified connections automatically,` `if -location argument is specified.` `Otherwise, the new wire LUT should be placed with place_node command.` `The name for the new node is hierarchy based, and the ECO Fitter will try to infer the name hierarchy.` `For example, if a node a|b|c|d needs to be created, users should make sure that hierarchy a|b|c exists in the netlist. If the source or destination node lies under a partition, the new wire LUT will` `be inserted under that partition.`	
Example Usage	`create_wirelut -name my_wirelut -from src_output -to dest_node -port D -location "X136 Y63 X149 Y82"` `create_wirelut -name my_wirelut -from src_output -to dest_node -port D` `place_node -name my_wirelut` `set to_node_id [get_netlist_node_id -name my_node]` `set iports [get_netlist_ports -node $to_node_id -type iport]` `set iport 0` `foreach_in_collection i $iports {` `  set iport $i` `  break`	

continued...

	} create_wirelut -name my_wirelut -from ~GND -to_node $to_node_id -to_port $iport place_node -name my_wirelut		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.4. eco_reroute (::quartus::eco)

The following table displays information for the `eco_reroute` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206	
Syntax	`eco_reroute [-h | -help] [-long_help] [-hold_slack` *<hold_slack>* `] [-keep_best] -node` *<node>* `-port` *<port>* `[-setup_slack` *<setup_slack>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-hold_slack` *<hold_slack>*	targeted hold slack (ns)
	`-keep_best`	keep best attempted slack if target slack not met
	`-node` *<node>*	node name
	`-port` *<port>*	port name
	`-setup_slack` *<setup_slack>*	targeted setup slack (ns)
Description	The eco_reroute command reroutes the iterm with user specified slacks.	
Example Usage	eco_reroute -node <node_name> -port <port_name> eco_reroute -node <node_name> -port <port_name> -setup_slack <setup_slack> -hold_slack <hold_slack> -max_iteration <max_iteration> -keep_best eco_reroute -node ff -port D -hold_slack 0.3 -setup_slack -2 -max_iteration 10	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.5. eco_unload_design (::quartus::eco)

The following table displays information for the `eco_unload_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`eco_unload_design [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	The eco_unload_design command unloads the current design. It also discards all changes that were made but not committed.		
Example Usage	eco_unload_design		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.6. fitter_report_timing (::quartus::eco)

The following table displays information for the `fitter_report_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`fitter_report_timing [-h | -help] [-long_help] [-detail <summary|path_only|path_and_clock|full_path>] [-extra_info <basic|all|none>] [-from <names>] [-from_clock <names>] [-hold] [-npaths <number>] [-panel_name <name>] [-recovery] [-removal] [-setup] [-through <names>] [-to <names>] [-to_clock <names>]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-detail <summary|path_only|path_and_clock|full_path>`	Option to determine how much detail should be shown in the path report	
	`-extra_info <basic|all|none>`	Option to determine how much detail should be shown in the Extra Info report	
	`-from <names>`	Valid destinations (string patterns are matched using Tcl string matching)	
	`-from_clock <names>`	Valid destinations (string patterns are matched using Tcl string matching)	
	`-hold`	Option to report clock hold paths	
	`-npaths <number>`	Specifies the number of paths to report (default=1)	
	`-panel_name <name>`	Sends the results to the timing report	
	`-recovery`	Option to report recovery paths	
	`-removal`	Option to report removal paths	
	`-setup`	Option to report clock setup paths	
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)	
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)	
	`-to_clock <names>`	Valid destinations (string patterns are matched using Tcl string matching)	
Description	`Reports the worst-case paths and associated slack.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.7. fitter_timing_summary (::quartus::eco)

The following table displays information for the `fitter_timing_summary` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`fitter_timing_summary [-h | -help] [-long_help] [-hold] [-panel_name <name> ] [-recovery] [-removal] [-setup]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-hold`		Option to report clock hold paths
	`-panel_name <name>`		Sends the results to the timing summary
	`-recovery`		Option to report recovery paths
	`-removal`		Option to report removal paths
	`-setup`		Option to report clock setup paths
Description	`Reports the worst-case Clock Setup and Clock Hold slacks and endpoint TNS (total negative slack) per clock domain. Total negative slack is the sum of all slacks less than zero for either destination registers or ports in the clock domain.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.8. get_available_snapshots (::quartus::eco)

The following table displays information for the `get_available_snapshots` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`get_available_snapshots [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`The get_available_snapshots command reports all available snapshots.`		
Example Usage	`get_available_snapshots`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.9. get_eco_checkpoint (::quartus::eco)

The following table displays information for the `get_eco_checkpoint` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`get_eco_checkpoint [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`The get_eco_checkpoint command prints the current checkpoint id in the console. It also returns the checkpoint id in a tcl object.`		
Example Usage	`get_eco_checkpoint`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.10. get_loaded_snapshot (::quartus::eco)

The following table displays information for the `get_loaded_snapshot` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`get_loaded_snapshot [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`The get_loaded_snapshot command gets the loaded snapshot.`		
Example Usage	`get_loaded_snapshot`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.11. get_lutmask_equation (::quartus::eco)

The following table displays information for the `get_lutmask_equation` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`get_lutmask_equation [-h | -help] [-long_help] [-name` *<name>* `] [-node` *<node_id>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name` *<name>*		name of the lut

continued...

	-node <*node_id*>		Node ID
Description	The get_lutmask_equation command reports the LUT equation of a given LUT.		
Example Usage	get_lutmask_equation -name <name_string> # print lutmask equation of all nodes of type lcell_comb project_open top eco_load_design set nodes [get_netlist_nodes -type lcell_comb] foreach_in_collection i $nodes { get_lutmask_equation -node $i } project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.12. get_node_location (::quartus::eco)

The following table displays information for the get_node_location Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	get_node_location [-h | -help] [-long_help] [-name <*node_name*>] [-node <*node_id*>]		
Arguments	-h | -help		Short help
	-long_help		Long help with examples and possible return values
	-name <*node_name*>		Name of node to get location for
	-node <*node_id*>		Node ID
Description	The get_node_location command will print the location of the node in the console and return the location in a tcl object. If the node is not placed, then the string "Unplaced" will be printed and returned.		
Example Usage	get_node_location -name node # print location of nodes of type FF project_open top eco_load_design set nodes [get_netlist_nodes -type FF] foreach_in_collection i $nodes { puts "Location = [get_node_location -node $i]" } project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.13. make_connection (::quartus::eco)

The following table displays information for the make_connection Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206
Syntax	make_connection [-h | -help] [-long_help] [-from <*output_net_name*>] [-from_node <*from_node*>] [-from_port <*from_port*>] [-port <*dest_node_port*>] [-tieoff <*VCC*|*GND*>] [-to <*dest_node_name*>] [-to_node <*to_node*>] [-to_port <*to_port*>]

continued...

Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-from` *<output_net_name>*		Source of the connection
	`-from_node` *<from_node>*		ID of the source node
	`-from_port` *<from_port>*		ID of the source port
	`-port` *<dest_node_port>*		Input port name of the destination node
	`-tieoff` *<VCC|GND>*		VCC or GND tieoff
	`-to` *<dest_node_name>*		Name of the destination node
	`-to_node` *<to_node>*		ID of the destination node
	`-to_port` *<to_port>*		ID of the destination port
Description			
	The make_connection command will connect the source signal to the destination block port. If the port has an existing connection, the command will remove the previous connection and connect it to the specified signal. make_connection expects a source and destination through: from - output net of the source block of the new connection, OR from_node AND from_port - node ID and output port ID of source, OR tieoff - tieoff value to AND port - name of the destination node and the input port name, OR to_node AND to_port - node ID and input port ID of the destination Note that the changed path will be routed immediately. If the path contains a node that is created during ECO compilation, then the paths will be routed after the node is placed with place_node command.		
Example Usage	`make_connection -from top|a_out -to top|x -port D` This example will connect top|a_out to the D input port of node top|x. `make_connection -tieoff VCC -to top|x -port D` This example will tie the D port of node top|x to VCC, either internally or via lcell. `load_package netlist` `load_package eco` `project_open top` `set from_node [get_netlist_node_id -name my_node]` `set oports [get_netlist_ports -node $from_node -type oport]` `set oport 0` `foreach_in_collection i $oports {` `  set oport $i` `  break` `}` `set to_node [get_netlist_node_id -name my_ff]` `set iports [get_netlist_ports -node $to_node -type iport]` `set iport 0` `foreach_in_collection i $iports {` `  set iport $i` `  break` `}` `make_connection -ARG(from_node) $from_node -ARG(from_port) $oport -ARG(to_node) $to_node -ARG(to_port) $iport` This example iterates over the netlist and node ports, then makes a connection between the first oport of my_node and the first iport of my_ff		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.14. modify_io_current_strength (::quartus::eco)

The following table displays information for the `modify_io_current_strength` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`modify_io_current_strength [-h | -help] [-long_help] -to` *<dest_pin_name>* *<current_strength>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-to` *<dest_pin_name>*		Name of the destination pin node
	<current_strength>		Current strength value
Description	`The modify_io_current_strength command will modify the current strength of the targeted pin.` `modify_io_current_strength expects 1 positional argument and an option argument:` `to - the name of the destination pin`		
Example Usage	`modify_io_current_strength 3mA -to top|ipin`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.15. modify_io_delay_chain (::quartus::eco)

The following table displays information for the `modify_io_delay_chain` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`modify_io_delay_chain [-h | -help] [-long_help] -to` *<dest_pin_name>* `-type` *<input|output|oe|io_12_lane_input_data|io_12_lane_input_strobe>* *<delay_chain>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-to` *<dest_pin_name>*		Name of the destination pin node
	`-type` *<input|output|oe|io_12_lane_input_data|io_12_lane_input_strobe>*		Type of the pin
	<delay_chain>		Delay chain value
Description	`The modify_io_delay_chain command will modify the delay chain setting of the targeted pin with the specified type.` `modify_io_delay_chain expects 1 positional argument and 2 option arguments:` `to   - the name of the destination pin` `type - the type of the pin`		
Example Usage	`modify_io_delay_chain 3 -to top|ipin -type INPUT`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.16. modify_io_slew_rate (::quartus::eco)

The following table displays information for the `modify_io_slew_rate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`modify_io_slew_rate [-h | -help] [-long_help] -to <dest_pin_name> <slew_rate>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-to <dest_pin_name>`	Name of the destination pin node	
	<slew_rate>	Slew rate value	
Description	`The modify_io_slew_rate command will modify the slew rate of the targeted pin.` `modify_io_slew_rate expects 1 positional argument and an option argument:` `to - the name of the destination pin`		
Example Usage	`modify_io_slew_rate 1 -to top|ipin`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.17. modify_lutmask (::quartus::eco)

The following table displays information for the `modify_lutmask` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206	
Syntax	`modify_lutmask [-h | -help] [-long_help] [-eqn <equation> ] [-mask <mask> ] [-num <num_of_inputs> ] -to <dest_node_name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-eqn <equation>`	LUT equation
	`-mask <mask>`	New lutmask
	`-num <num_of_inputs>`	Number of inputs
	`-to <dest_node_name>`	Name of the destination node
Description	`The modify_lutmask command will modify the LUT-mask of the matching node,` `with LUT-mask value in binary or hexadecimal, or with equivalent LUT-mask value` `computed from specified logical equation.` `modify_lutmask expects 2 arguments (1 from mask or eqn):` `to   - name of the destination atom` `mask - the LUT-mask value to be modified - in binary or hexadecimal` `eqn  - the logical equation of the inputs (A, B, C, D, E, F)` `     - the supported lexical tokens include AND('&'), OR('|'), XOR('^'), NOT('!'),` `OPEN_BRACE('('), CLOSE_BRACE(')')`	

continued...

Example Usage	<pre>modify_lutmask -to top|lut_a -mask 0xFF00FF00 modify_lutmask -to top|lut_b -mask 0b111111111001010 modify_lutmask -to top|lut_c -eqn {a&b&c}</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.18. place_node (::quartus::eco)

The following table displays information for the `place_node` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`place_node [-h | -help] [-long_help] [-force] [-location` *<location>* `] [-name` *<node_name>* `] [-node` *<node_id>* `] [-sample` *<sample>* `] [-timing_driven]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-force`	Force overwriting existing location constraint on the node	
	`-location` *<location>*	Exact location or region	
	`-name` *<node_name>*	Name of the new node	
	`-node` *<node_id>*	Node ID	
	`-sample` *<sample>*	Number of locations to try to place the node	
	`-timing_driven`	Try to place the node at a location meeting timing	
Description	<pre>The place_node command will place the specified node either automatically, or within the region if -location is specified. If -sample is specified, then the command will randomly pick locations and place the node at the first valid location found. If -timing_driven is specified then the command will try to place the node at a location that meets timing. If none of the locations meet timing, then the node will be placed at the location with the highest slacks. The place_node command can be used on nodes that have been placed. -location argument takes in an exact location (-location "X20 Y20"), a region (-location "X20 Y20 X30 Y30"), or an ALM sublocation (-location "FF_X20_Y20_N10"). -force argument will force overwrite existing location constraints on the node, if any. -force will not overwrite Partial Reconfiguration regions.</pre>		
Example Usage	<pre>place_node -name eco_new_lut -location "X136 Y63 X149 Y82" place_node -name eco_new_lut -location "X5 Y10" place_node -name eco_new_lut -location "X5 Y10" -timing_driven place_node -name eco_new_ff -location "FF_X20_Y20_N10" place_node -name eco_new_ff -location "FF_X20_Y20_N10" -force place_node -name eco_new_lut -location "X5 Y10 X100 Y200" -sample 10 place_node -name eco_new_ff -location "X5 Y10 X100 Y200" -sample 100 -timing_driven # place all FFs project_open top eco_load_design set nodes [get_netlist_nodes -type FF] foreach_in_collection i $nodes { place_node -node $i }</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.19. remove_connection (::quartus::eco)

The following table displays information for the `remove_connection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`remove_connection [-h | -help] [-long_help] [-from <output_net_name> ] [-from_node <from_node> ] [-from_port <from_port> ] [-port <dest_node_port> ] [-to <dest_node_name> ] [-to_node <to_node> ] [-to_port <to_port> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-from <output_net_name>`	Source of the connection	
	`-from_node <from_node>`	ID of the source node	
	`-from_port <from_port>`	ID of the source port	
	`-port <dest_node_port>`	Input port name of the destination node	
	`-to <dest_node_name>`	Name of the destination node	
	`-to_node <to_node>`	ID of the destination node	
	`-to_port <to_port>`	ID of the destination port	
Description			
	The remove_connection command will disconnect the source signal from the destination block port, and set the input port to a disconnected state. remove_connection expects a source and destination through: from - output net of the source block of the new connection, OR from_node AND from_port - node ID and output port ID of source to AND port - name of the destination node and the input port name, OR to_node AND to_port - node ID and input port ID of the destination Note: Connection path containing Hyper Registers is not allowed to be removed.		
Example Usage			
	`remove_connection -from top|a_out -to top|x -port D` This example will disconnect top|a_out from the D input port of node top|x, and set top|x:D to a disconnected state. `load_package netlist` `load_package eco` `project_open top` `set from_node [get_netlist_node_id -name my_node]` `set oports [get_netlist_ports -node $from_node -type oport]` `set oport 0` `foreach_in_collection i $oports {` `  set oport $i` `  break` `}` `set to_node [get_netlist_node_id -name my_ff]` `set iports [get_netlist_ports -node $to_node -type iport]` `set iport 0` `foreach_in_collection i $iports {` `  set iport $i` `  break` `}` `remove_connection -ARG(from_node) $from_node -ARG(from_port) $oport -ARG(to_node) $to_node -ARG(to_port) $iport` This example iterates over the netlist and node ports, then removes the connection between the first oport of my_node and the first iport of my_ff		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.20. remove_node (::quartus::eco)

The following table displays information for the `remove_node` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`remove_node [-h | -help] [-long_help] [-name <node_name> ] [-node <node_id> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <node_name>`	Name of the node to remove	
	`-node <node_id>`	Node ID	
Description	`The remove_node command will remove the specified node from final netlist.` `The node's source and destination signals will be disconnected.`		
Example Usage	`remove_node -name my_ff` `# remove all nodes of type FF` `project_open top` `eco_load_design` `set nodes [get_netlist_nodes -type FF]` `foreach_in_collection i $nodes {` `  remove_node -node $i` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.21. report_connections (::quartus::eco)

The following table displays information for the `report_connections` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206	
Syntax	`report_connections [-h | -help] [-long_help] [-from <from> ] [-from_port <from_port> ] [-limit <limit> ] [-return_result] [-timing] [-to <to> ] [-to_port <to_port> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-from <from>`	Name of the source node
	`-from_port <from_port>`	Name of the source port
	`-limit <limit>`	Limit number of output connections reported
	`-return_result`	Return the result in a tcl object
	`-timing`	Report slacks
	`-to <to>`	Name of the destination node
	`-to_port <to_port>`	Name of the destination port
Description	`The report_connections command will report connections between one of the following:` `  - (-from)  report all connections from a node` `  - (-from -from_port)  report connections form a port of a node` `  - (-to)  report all connections to a node`	

continued...

	`` - (-to -to_port) report connection to a port of a node - (-from -to) report connection between 2 nodes By default, 100 connections will be reported. In command-line mode, the result will be posted as info messages to the console. If -return_result is specified then the result will also be returned as a tcl object. ```
Example Usage	``` report_connections -from my_ff report_connections -from my_ff -from_port Q report_connections -to my_ff report_connections -to my_ff -to_port D report_connections -from my_ff -to my_ff1 report_connections -from my_ff -from_port Q -limit 10 report_connections -from my_ff -from_port Q -return_result ```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.22. report_legal_locations (::quartus::eco)

The following table displays information for the `report_legal_locations` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206	
Syntax	`report_legal_locations [-h | -help] [-long_help] [-check_routing] -location` *<location>* `[-name` *<node_name>* `] [-node` *<node_id>* `] [-patient] [-report_illegal] [-return_result]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-check_routing`	Try to legalize the location by checking routing
	`-location` *<location>*	Region to search for legal locations
	`-name` *<node_name>*	Name of node to check legal locations for
	`-node` *<node_id>*	Node ID
	`-patient`	Override restriction on search region size
	`-report_illegal`	Report illegal locations and reasons within the search region
	`-return_result`	Return the result in a tcl object
Description	``` The report_legal_locations command will search for all legal locations within the region specified by -location for the specified node to be placed. The command can be used on nodes that have or have not been placed. Specify -patient to override the restriction on the search region size. Specify -report_illegal to report illegal location reasons. In command-line mode, the result will be posted as info messages to the console. If -return_result is specified then the result will also be returned as a tcl object. If -check_routing is specified then a location will be determined illegal if the legalization step fails and slacks will be reported for legal locations. Otherwise the legalization step will be skipped. Note that the -check_routing option is only supported when all nodes but the target have been placed. The report_legal_locations command does not work in "quartus_fit --eco" mode. ```	
Example Usage	``` report_legal_locations -name node -location "X136 Y63 X145 Y72" report_legal_locations -name node -location "X5 Y10 X5 Y10" report_legal_locations -name node -location "X5 Y10 X5 Y10" -report_illegal report_legal_locations -name node -location "X5 Y10 X5 Y10" -return_result report_legal_locations -name node -location "X5 Y10 X5 Y10" -check_routing report_legal_locations -name node -location "X136 Y63 X149 Y82" -patient report_legal_locations -name node -location "X136 Y63 X149 Y82" -patient -return_result ```	

continued...

	`` # check legal locations of multiple FFs in a specified region project_open top eco_load_design set nodes [get_netlist_nodes -type FF] foreach_in_collection i $nodes { report_legal_locations -node $i -location "X136 Y63 X145 Y72" } project_close ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.23. report_nodes_at_location (::quartus::eco)

The following table displays information for the `report_nodes_at_location` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`report_nodes_at_location [-h | -help] [-long_help] -location` *<location>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-location` *<location>*		location string to search
Description	``` The report_nodes_at_location command reports the nodes at a given location. ```		
Example Usage	``` report_nodes_at_location -location <location_string> ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.24. eco::report_partitions (::quartus::eco)

The following table displays information for the `eco::report_partitions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`eco::report_partitions [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	``` This command currently contains no help description. ```		
Example Usage	``` eco::report_partitions ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.25. report_ports (::quartus::eco)

The following table displays information for the `report_ports` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`report_ports [-h | -help] [-long_help] [-name <node_name> ] [-node <node_id> ] [-return_result] [-timing]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <node_name>`	Name of the node to query	
	`-node <node_id>`	Node ID	
	`-return_result`	Return the result in a tcl object	
	`-timing`	Report the slacks on each port	
Description	`The report_ports command will report all input ports of a node.` `In command-line mode, the result will be posted as info messages to the console.` `If -timing is specified then the slacks on each port will be reported.` `If -return_result is specified then the result will also be returned as a` `tcl object.`		
Example Usage	`report_ports -name my_ff` `report_ports -name my_ff -timing` `report_ports -name my_ff -return_result` `# get input ports of a FF` `project_open top` `eco_load_design` `set node [get_netlist_node_id -name my_ff]` `report_ports -node $node` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.26. report_routing (::quartus::eco)

The following table displays information for the `report_routing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206	
Syntax	`report_routing [-h | -help] [-long_help] [-return_result] -to <to> -to_port <to_port>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-return_result`	Return the result in a tcl object
	`-to <to>`	Name of the destination node
	`-to_port <to_port>`	Name of the destination port
Description	`The report_connections command will report routing between a connection.` `In command-line mode, the result will be posted as info messages to the console.` `If -return_result is specified then the result will also be returned as a` `tcl object.`	

continued...

Example Usage	`report_routing -to my_ff1 -to_port D`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.27. report_unplaced_nodes (::quartus::eco)

The following table displays information for the `report_unplaced_nodes` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`report_unplaced_nodes [-h | -help] [-long_help] [-return_result]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-return_result`	Return the result in a tcl object	
Description	`The report_unplaced_nodes command will report all unplaced nodes.` `In command-line mode, the result will be posted as info messages to the console.` `If -return_result is specified then the result will also be returned as a` `tcl object.`		
Example Usage	`report_unplaced_nodes` `report_unplaced_nodes -return_result`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.28. restore_eco_checkpoint (::quartus::eco)

The following table displays information for the `restore_eco_checkpoint` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`restore_eco_checkpoint [-h | -help] [-long_help] <cid>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<cid>	Checkpoint ID of a change	
Description	`The restore_eco_checkpoint command reverts the design back to the targeted checkpoint.`		
Example Usage	`restore_eco_checkpoint 12345`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.29. unplace_node (::quartus::eco)

The following table displays information for the `unplace_node` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`unplace_node [-h | -help] [-long_help] [-name <node_name> ] [-node <node_id> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <node_name>`	Name of the node to unplace	
	`-node <node_id>`	Node ID	
Description	`The unplace_node command will unplace a given node.`		
Example Usage	`unplace_node -name my_ff` `# unplace all nodes of type FF` `project_open top` `eco_load_design` `set nodes [get_netlist_nodes -type FF]` `foreach_in_collection i $nodes {` `  unplace_node -node $i` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.10.30. update_mif_files (::quartus::eco)

The following table displays information for the `update_mif_files` Tcl command:

Tcl Package and Version	Belongs to ::quartus::eco on page 206		
Syntax	`update_mif_files [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Update memory contents from the Memory Initialization` `File (.mif) or Hexadecimal (Intel-Format) File (.hex) for all` `RAM or CAM atoms.`		
Example Usage	`  load_package eco` `  update_mif_files`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.11. ::quartus::external_memif_toolkit

The following table displays information for the **::quartus::external_memif_toolkit** Tcl package:

Tcl Package and Version	::quartus::external_memif_toolkit 1.0
Description	This package contains the set of Tcl functions for interacting with external memory interfaces and debug components
Availability	This package is available for loading in the following executables: qpro_sh quartus_sh
Tcl Commands	apply_setting (::quartus::external_memif_toolkit) on page 225 calibrate_termination (::quartus::external_memif_toolkit) on page 226 configure_driver (::quartus::external_memif_toolkit) on page 227 create_connection_report (::quartus::external_memif_toolkit) on page 227 create_toolkit_report (::quartus::external_memif_toolkit) on page 228 driver_margining (::quartus::external_memif_toolkit) on page 229 establish_connection (::quartus::external_memif_toolkit) on page 230 generate_eye_diagram (::quartus::external_memif_toolkit) on page 231 get_connection_commands (::quartus::external_memif_toolkit) on page 232 get_connection_info (::quartus::external_memif_toolkit) on page 232 get_connection_interfaces (::quartus::external_memif_toolkit) on page 233 get_connection_report_info (::quartus::external_memif_toolkit) on page 234 get_connection_report_types (::quartus::external_memif_toolkit) on page 235 get_connection_types (::quartus::external_memif_toolkit) on page 236 get_connections (::quartus::external_memif_toolkit) on page 236 get_device_names (::quartus::external_memif_toolkit) on page 237 get_hardware_names (::quartus::external_memif_toolkit) on page 237 get_setting_types (::quartus::external_memif_toolkit) on page 238 get_toolkit_report_types (::quartus::external_memif_toolkit) on page 239 initialize_connections (::quartus::external_memif_toolkit) on page 239 link_project_to_device (::quartus::external_memif_toolkit) on page 240 read_setting (::quartus::external_memif_toolkit) on page 241 reindex_connections (::quartus::external_memif_toolkit) on page 242 reset_tg2 (::quartus::external_memif_toolkit) on page 243 run_connection_command (::quartus::external_memif_toolkit) on page 243 set_active_interface (::quartus::external_memif_toolkit) on page 244 set_stress_pattern (::quartus::external_memif_toolkit) on page 245 terminate_connection (::quartus::external_memif_toolkit) on page 245 terminate_connections (::quartus::external_memif_toolkit) on page 246 unlink_project_from_device (::quartus::external_memif_toolkit) on page 247 write_connection_target_report (::quartus::external_memif_toolkit) on page 247

3.1.11.1. apply_setting (::quartus::external_memif_toolkit)

The following table displays information for the `apply_setting` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`apply_setting [-h | -help] [-long_help] -id <name> [-index <index> ] [-rank <rank> ] -type <type> -value <value>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id <name>`	The connection ID to communicate with
	`-index <index>`	Index of the target setting within the connection
	`-rank <rank>`	Rank (shadow register) of the target setting within the connection
	`-type <type>`	The type of setting on the EMIF interface
	`-value <value>`	Value to apply

continued...

Description	Applies the specified memory interface setting for the specified target connection.		
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `foreach conn [get_connections] {` `  foreach type [get_setting_types -id $conn] {` `  apply_setting -id $conn -type $type -index 0 -value 0` `  }` `}` `terminate_connections` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The specified setting type <*string*> is illegal. Please specify a valid setting type. The list of valid types is available by running get_setting_types.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: The specified setting is outside the legal range.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.2. calibrate_termination (::quartus::external_memif_toolkit)

The following table displays information for the `calibrate_termination` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`calibrate_termination [-h | -help] [-long_help] -id <name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-id <name>`	The connection ID to communicate with	
Description	Test termination settings on the interface.		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.3. configure_driver (::quartus::external_memif_toolkit)

The following table displays information for the `configure_driver` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`configure_driver [-h | -help] [-long_help] [-data <data> ] -id <name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-data <data>`	Address/data pairs to be written to the driver configuration block	
	`-id <name>`	The connection ID to communicate with	
Description	`Configures the traffic generator.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.4. create_connection_report (::quartus::external_memif_toolkit)

The following table displays information for the `create_connection_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`create_connection_report [-h | -help] [-long_help] -id <name> -report_type <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id <name>`	The connection ID to communicate with
	`-report_type <name>`	The report type to generate

continued...

Description	Create a report of the specified type for the connection id. The resulting report is then available for query using the report TCL package.		
Example Usage	load_package external_memif_toolkit load_package report project_open dut initialize_connections set hw_name [lindex [get_hardware_names] 0] set dev_name [lindex [get_device_names -hardware_name $hw_name] 0] link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof set conn [lindex [get_connections -type emif] 0] establish_connection -id $conn create_connection_report -id $conn -report_type summary load_report_database -type emit set report_panel_names [get_report_panel_names] post_message -type info "Found the following report panels:" foreach panel_name $report_panel_names { post_message -type info " $panel_name" } unload_report terminate_connections project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: An error occurred trying to create the report <*string*> for the target <*string*>.
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The specified report type <*string*> is illegal. The legal report types are: <*string*> .
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.5. create_toolkit_report (::quartus::external_memif_toolkit)

The following table displays information for the `create_toolkit_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`create_toolkit_report [-h | -help] [-long_help] -report_type` <*name*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-report_type` <*name*>	The report type to generate

continued...

Description	Create a toolkit report of the specified type. These reports are general toolkit reports, not connection specific reports. The resulting report is then available for query using the report TCL package.
Example Usage	`load_package external_memif_toolkit` `load_package report` `project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `create_toolkit_report -report_type discovered_connections` `create_toolkit_report -report_type detailed_connections` `load_report_database -type emit` `set report_panel_names [get_report_panel_names]` `post_message -type info "Found the following report panels:"` `foreach panel_name $report_panel_names {` `  post_message -type info "  $panel_name"` `}` `unload_report` `terminate_connections` `project_close`

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: An error occurred trying to create the report <*string*> for the target <*string*>.
TCL_ERROR	1	ERROR: The specified report type <*string*> is illegal. The legal report types are: <*string*> .
TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.6. driver_margining (::quartus::external_memif_toolkit)

The following table displays information for the `driver_margining` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`driver_margining [-h | -help] [-long_help] [-adjust_delays] -fail_id` <*fail_id*> `-id` <*name*> `-pass_id` <*pass_id*> `-pnf_ids` <*pnf_ids*> `-resetn_id` <*resetn_id*> `[-skip_dm] [-skip_read] [-skip_write] [-step_size` <*step_size*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-adjust_delays`	Adjust delays on the interface based on driver margining results
	`-fail_id` <*fail_id*>	Connection ID of the In-System Probe which controls the fail signal from the driver
	`-id` <*name*>	The connection ID to communicate with

continued...

	`-pass_id <pass_id>`		Connection ID of the In-System Probe which controls the pass signal from the driver
	`-pnf_ids <pnf_ids>`		TCL list of PNF (pass not fail) connection IDs of In-System Probes from the driver
	`-resetn_id <resetn_id>`		Connection ID of the In-System Source which controls the resetn for the driver
	`-skip_dm`		Skip driver margining on dm
	`-skip_read`		Skip driver margining on read
	`-skip_write`		Skip driver margining on write
	`-step_size <step_size>`		Granularity of the driver margining operation, in terms of delay settings. Smaller values are more precise, but consume more time.
Description	`Performs read and write driver margining on the selected EMIF connection.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.7. establish_connection (::quartus::external_memif_toolkit)

The following table displays information for the `establish_connection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`establish_connection [-h | -help] [-long_help] [-connection_name <name> ] -id <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-connection_name <name>`	Optional nickname to use for a connection.
	`-id <name>`	The connection ID to communicate with
Description	`Establishes a connection to a connection target.`	
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof`	

continued...

	`` set conn [lindex [get_connections -type emif] 0] establish_connection -id $conn terminate_connections project_close ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Could not establish a connection to target <*string*>.
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The specified connection name <*string*> is illegal as a connection already exists with that name. Please specify a valid connection name.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.8. generate_eye_diagram (::quartus::external_memif_toolkit)

The following table displays information for the `generate_eye_diagram` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`generate_eye_diagram [-h | -help] [-long_help] -id <name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-id <name>`		The connection ID to communicate with
Description	``` Generates eye diagrams for the selected EMIF connection. ```		
Example Usage	``` This command currently contains no example usage. ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.9. get_connection_commands (::quartus::external_memif_toolkit)

The following table displays information for the `get_connection_commands` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`get_connection_commands [-h | -help] [-long_help] -id <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id <name>`	The connection ID to communicate with
Description	`Returns a TCL list of supported commands for the specific target connection.`	
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `foreach conn [get_connections] {` `  puts "Connection : $conn"` `  foreach cmd [get_connection_commands -id $conn] {` `  puts "  Command : $cmd"` `  }` `}` `terminate_connections` `project_close`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.10. get_connection_info (::quartus::external_memif_toolkit)

The following table displays information for the `get_connection_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`get_connection_info [-h | -help] [-long_help] [-hpath] -id <name> [-sld_type]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-hpath`	Queries the hierarchy name of the connection
	`-id <name>`	The connection ID to communicate with

continued...

	`-sld_type`	Queries SLD type of a connection
Description	`Queries information about a connection target.`	
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `foreach conn [get_connections] {` `  puts "Connection : $conn"` `  puts "  Hierarchy Path : [get_connection_info -id $conn -hpath]"` `}` `terminate_connections` `project_close`	

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: No information about the connection ID was queried. Please specify one query parameter.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.11. get_connection_interfaces (::quartus::external_memif_toolkit)

The following table displays information for the `get_connection_interfaces` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`get_connection_interfaces [-h | -help] [-long_help] -id <name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-id <name>`	The connection ID to communicate with	
Description	`Returns the interfaces available for the specified connection id.`		
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `set conn [lindex [get_connections -type emif] 0]` `establish_connection -id $conn` `get_connection_interfaces -id $conn` `terminate_connections` `project_close`		

Return Value	Code Name	Code	String Return

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.12. get_connection_report_info (::quartus::external_memif_toolkit)

The following table displays information for the `get_connection_report_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`get_connection_report_info [-h | -help] [-long_help] -id <name> [-name] -report_type <name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-id <name>`	The connection ID to communicate with	
	`-name`	Queries the name of the report	
	`-report_type <name>`	The report type to generate	
Description	`Queries info about a report type.`		
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `foreach conn [get_connections] {` `  puts "Connection : $conn"` `  foreach rpt [get_connection_report_types -id $conn] {` `  puts "  Report : $rpt"` `  puts "  Report name : [get_connection_report_info -id $conn -report_type $rpt -name]"` `  }` `}` `terminate_connections` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.

continued...

	TCL_ERROR	1	ERROR: The specified report type <*string*> is illegal. The legal report types are: <*string*> .
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.13. get_connection_report_types (::quartus::external_memif_toolkit)

The following table displays information for the `get_connection_report_types` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`get_connection_report_types [-h | -help] [-long_help] -id <name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-id <name>`	The connection ID to communicate with	
Description	`Returns a TCL list of supported report types for the specific target connection.`		
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `foreach conn [get_connections] {` `  puts "Connection : $conn"` `  puts "  Supported report types: [get_connection_report_types -id $conn]"` `}` `terminate_connections` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.14. get_connection_types (::quartus::external_memif_toolkit)

The following table displays information for the `get_connection_types` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`get_connection_types [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns a list of valid connection target types.`		
Example Usage	`puts "Valid connection types are: [get_connection_types]"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.11.15. get_connections (::quartus::external_memif_toolkit)

The following table displays information for the `get_connections` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`get_connections [-h | -help] [-long_help] [-type <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-type <name>`		The type of the connection to connect to
Description	`Returns a TCL list of connection IDs for connections.`		
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `foreach conn [get_connections] {` `  puts "Connection : $conn"` `  puts "  Hierarchy Path : [get_connection_info -id $conn -hpath]"` `}` `terminate_connections` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Supplying hardware_name and device_name is mutually exclusive to supplying type
	TCL_ERROR	1	ERROR: Both hardware_name and device_name must be supplied if either is supplied

continued...

	TCL_ERROR	1	ERROR: Connection type <*string*> is not a recognized connection type.
	TCL_ERROR	1	ERROR: No device name called <*string*> could be found on hardware named <*string*>.
	TCL_ERROR	1	ERROR: No hardware name called <*string*> could be found.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.16. get_device_names (::quartus::external_memif_toolkit)

The following table displays information for the `get_device_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`get_device_names [-h | -help] [-long_help] -hardware_name <name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-hardware_name <name>`		The name of the hardware connection to use
Description	`Returns a list of device names found for the given hardware.`		
Example Usage	`project_open dut` `initialize_connections` `foreach hw_name [get_hardware_names] {` `  foreach dev_name [get_device_names -hardware_name $hw_name] {` `  puts "Found device name $dev_name on hardware $hw_name"` `  }` `}` `terminate_connections` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No hardware name called <*string*> could be found.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.17. get_hardware_names (::quartus::external_memif_toolkit)

The following table displays information for the `get_hardware_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`get_hardware_names [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

Description	Returns a list of hardware names found.		
Example Usage	`project_open dut` `initialize_connections` `foreach hw_name [get_hardware_names] {` `  puts "Found hardware name $hw_name"` `}` `terminate_connections` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.18. get_setting_types (::quartus::external_memif_toolkit)

The following table displays information for the `get_setting_types` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`get_setting_types [-h | -help] [-long_help] -id <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id <name>`	The connection ID to communicate with
Description	Returns a TCL list of supported memory interface setting types for the specific target connection.	
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `foreach conn [get_connections] {` `  puts "Connection : $conn"` `  puts "  Supported setting types: [get_setting_types -id $conn]"` `}` `terminate_connections` `project_close`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.19. get_toolkit_report_types (::quartus::external_memif_toolkit)

The following table displays information for the `get_toolkit_report_types` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`get_toolkit_report_types [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Returns a TCL list of supported toolkit report types. These reports are general toolkit reports, not connection specific reports.	
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `puts "Supported toolkit report types: [get_toolkit_report_types]"` `terminate_connections` `project_close`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.20. initialize_connections (::quartus::external_memif_toolkit)

The following table displays information for the `initialize_connections` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`initialize_connections [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Initializes the internal data structures of the toolkit. This command must be run before any other toolkit commands are executed.	
Example Usage	`project_open dut` `initialize_connections` `terminate_connections` `project_close`	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unable to find active revision. Specify an active revision name using set_current_revision <revision name>.
	TCL_ERROR	1	ERROR: The current project specifies a part that cannot be loaded. Verify the device family is correctly installed.
	TCL_ERROR	1	ERROR: The connection to the hardware drivers could not be established.
	TCL_ERROR	1	ERROR: Illegal connections detected
	TCL_ERROR	1	ERROR: The current project settings are invalid. Verify that all project settings are valid.
	TCL_ERROR	1	ERROR: The JTAG Debug Information (.jdi) file called <*string*> could not be found. Ensure this file exists or run quartus_asm to create it.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Could not initialize System Console. <*string*>
	TCL_ERROR	1	ERROR: Toolkit has already been initialized. Use terminate_connections to de-initialize the toolkit.

3.1.11.21. link_project_to_device (::quartus::external_memif_toolkit)

The following table displays information for the `link_project_to_device` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`link_project_to_device [-h | -help] [-long_help] -device_name <name> -hardware_name <name> [-jdi_file <name> ] [-sof_file <name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-device_name <name>`	The name of the device to connect to
	`-hardware_name <name>`	The name of the hardware connection to use
	`-jdi_file <name>`	Specifies the JTAG Debugging Information file to use when creating the design link.
	`-sof_file <name>`	Specifies the SOF file to use when creating the design link.
Description	`Links the currently opened project to the specified target device on the specified hardware.`	
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `terminate_connections` `project_close`	
Return Value	**Code Name**	**Code**	**String Return**

continued...

TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: An error occurred while linking the project to the device.
TCL_ERROR	1	ERROR: No device name called <*string*> could be found on hardware named <*string*>.
TCL_ERROR	1	ERROR: No hardware name called <*string*> could be found.
TCL_ERROR	1	ERROR: The JTAG Debug Information (.jdi) file called <*string*> could not be found. Ensure this file exists or run quartus_asm to create it.
TCL_ERROR	1	ERROR: A JTAG Debug Information (.jdi) file or a SOF file (.sof) is required for linking a project to a device, but no file was supplied. Ensure this file exists or run quartus_asm to create it.
TCL_ERROR	1	ERROR: The currently opened project has already been linked to a device. Use unlink_project_from_device to unlink the project from a device.
TCL_ERROR	1	ERROR: The SOF (.sof) file called <*string*> could not be found. Ensure this file exists or run quartus_asm to create it.
TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.22. read_setting (::quartus::external_memif_toolkit)

The following table displays information for the `read_setting` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`read_setting [-h | -help] [-long_help] -id <name> [-index <index> ] [-rank <rank> ] -type <type>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id <name>`	The connection ID to communicate with
	`-index <index>`	Index of the target setting within the connection
	`-rank <rank>`	Rank (shadow register) of the target setting within the connection
	`-type <type>`	The type of setting on the EMIF interface
Description	`Reads the specified memory interface setting for the specified target connection.`	
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `foreach conn [get_connections] {` `  foreach type [get_setting_types -id $conn] {` `  read_setting -id $conn -type $type -index 0` `  }` `}`	

continued...

	terminate_connections project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The specified setting type <*string*> is illegal. Please specify a valid setting type. The list of valid types is available by running get_setting_types.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: The specified setting is outside the legal range.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.23. reindex_connections (::quartus::external_memif_toolkit)

The following table displays information for the `reindex_connections` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`reindex_connections [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	Reinitializes the list of known hardware connections.		
Example Usage	project_open dut initialize_connections foreach hw_name [get_hardware_names] { puts "Found hardware name $hw_name" } reindex_connections foreach hw_name [get_hardware_names] { puts "Found hardware name $hw_name" } terminate_connections project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.24. reset_tg2 (::quartus::external_memif_toolkit)

The following table displays information for the `reset_tg2` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`reset_tg2 [-h | -help] [-long_help] -resetn_id <resetn_id> -tg2_id <tg2_id>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-resetn_id <resetn_id>`	Connection ID of the resetn	
	`-tg2_id <tg2_id>`	Connection ID of the Traffic Generator 2.0 associated with this interface	
Description	`Resets a Traffic Generator 2.0 instance.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.25. run_connection_command (::quartus::external_memif_toolkit)

The following table displays information for the `run_connection_command` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`run_connection_command [-h | -help] [-long_help] -command_name <name> -id <name> [-payload <name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-command_name <name>`	The command name to execute
	`-id <name>`	The connection ID to communicate with
	`-payload <name>`	TCL list of parameter data to use when executing the command
Description	`Executes a command of the specified type and for the connection id.`	
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]`	

continued...

```
set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]
link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof

set conn [lindex [get_connections -type emif] 0]
establish_connection -id $conn

run_connection_command -id $conn -command_name nop

terminate_connections

project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: An error occurred trying to execute the command <*string*> for the target <*string*>.
	TCL_ERROR	1	ERROR: The specified command name <*string*> is illegal. The legal command names are: <*string*> .
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: No information about the report was queried. Please specify one query parameter.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.26. set_active_interface (::quartus::external_memif_toolkit)

The following table displays information for the `set_active_interface` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`set_active_interface [-h | -help] [-long_help] -id <name> -interface_id <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id <name>`	The connection ID to communicate with
	`-interface_id <name>`	The interface ID to communicate with
Description		

```
Selects the active interface to debug for the specified connection id.
```

Example Usage

```
This command currently contains no example usage.
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.

continued...

	TCL_ERROR	1	ERROR: The specified interface ID <*string*> is illegal. Please specify a valid interface ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.27. set_stress_pattern (::quartus::external_memif_toolkit)

The following table displays information for the `set_stress_pattern` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`set_stress_pattern [-h | -help] [-long_help] -id <name> -value <value>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-id <name>`	The connection ID to communicate with	
	`-value <value>`	1 to enable stress pattern in calibration, 0 to disable	
Description	`Enable or disable the stress pattern for future calibrations.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.28. terminate_connection (::quartus::external_memif_toolkit)

The following table displays information for the `terminate_connection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`terminate_connection [-h | -help] [-long_help] -id <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id <name>`	The connection ID to communicate with
Description	`Terminates a connection to a connection target.`	

continued...

Example Usage	<pre>project_open dut initialize_connections set hw_name [lindex [get_hardware_names] 0] set dev_name [lindex [get_device_names -hardware_name $hw_name] 0] link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof set conn [lindex [get_connections -type emif] 0] establish_connection -id $conn terminate_connection -id $conn terminate_connections project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Connection ID <*string*> could not be terminated because the connection was not established.
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.29. terminate_connections (::quartus::external_memif_toolkit)

The following table displays information for the `terminate_connections` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`terminate_connections [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	<pre>Deletes the internal data structures of the toolkit.</pre>	
Example Usage	<pre>project_open dut puts "Preparing to initialize connections" initialize_connections puts "Preparing to delete connections" terminate_connections project_close</pre>	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.30. unlink_project_from_device (::quartus::external_memif_toolkit)

The following table displays information for the `unlink_project_from_device` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225		
Syntax	`unlink_project_from_device [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Unlinks the currently opened project from the currently linked target device on the specified hardware.`		
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `unlink_project_from_device` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `terminate_connections` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The JTAG Debug Information (.jdi) file called <*string*> could not be found. Ensure this file exists or run quartus_asm to create it.
	TCL_ERROR	1	ERROR: The currently opened project has not been linked to a device. Run link_project_to_device to link a project to a device.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.11.31. write_connection_target_report (::quartus::external_memif_toolkit)

The following table displays information for the `write_connection_target_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::external_memif_toolkit on page 225	
Syntax	`write_connection_target_report [-h | -help] [-long_help] -file` <*name*> `-id` <*name*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-file` <*name*>	File name of report to write
	`-id` <*name*>	The connection ID to communicate with

continued...

Description	Writes the reports for the given connection target to a filename.		
Example Usage	`project_open dut` `initialize_connections` `set hw_name [lindex [get_hardware_names] 0]` `set dev_name [lindex [get_device_names -hardware_name $hw_name] 0]` `link_project_to_device -hardware_name $hw_name -device_name $dev_name -sof_file dut.sof` `set conn [lindex [get_connections] 0]` `write_connection_target_report -id $conn -file report.rpt` `terminate_connections` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified connection ID <*string*> is illegal. Please specify a valid connection ID.
	TCL_ERROR	1	ERROR: No reports for connection ID <*string*> have been generated. Please run create_report_for_target before writing reports to file.
	TCL_ERROR	1	ERROR: Toolkit has not been initialized. Use initialize_connections to initialize the toolkit.

3.1.12. ::quartus::fif

The following table displays information for the **::quartus::fif** Tcl package:

Tcl Package and Version	::quartus::fif 1.0
Description	This package contains the set of Tcl functions for using the Fault Injection File (FIF) Driver.
Availability	This package is loaded by default in the following executable: quartus_fif
Tcl Commands	check (::quartus::fif) on page 248 dump (::quartus::fif) on page 249 dump_cram_frame (::quartus::fif) on page 249 dump_mem (::quartus::fif) on page 250 dump_pr_bitstream (::quartus::fif) on page 250 generate (::quartus::fif) on page 251 get_frame_count (::quartus::fif) on page 251 get_frame_size (::quartus::fif) on page 252 get_sector_information_sdm_based_fpga (::quartus::fif) on page 252 get_sensitive_location (::quartus::fif) on page 253 get_sensitive_location_sdm_based_fpga (::quartus::fif) on page 253 setup (::quartus::fif) on page 254 setup_sdm_based_fpga (::quartus::fif) on page 254 terminate (::quartus::fif) on page 255

3.1.12.1. check (::quartus::fif)

The following table displays information for the check Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248
Syntax	`check [-h | -help] [-long_help] -frame <frame> -index <index>`

continued...

Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-frame` *<frame>*	CRAM frame ID
	`-index` *<index>*	CRAM frame bit location
Description	`Check is the specified location contains sensitive bit.` `Returns 1, if the specified location contains sensitive bit.` `Returns 0, otherwise.`	
Example Usage	`check -frame 3 -index 100`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.

3.1.12.2. dump (::quartus::fif)

The following table displays information for the `dump` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248	
Syntax	`dump [-h | -help] [-long_help] [-all]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all`	Option to display all information
Description	`Dump FIF driver contents.`	
Example Usage	`dump` `dump -all`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error *<string>*.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.

3.1.12.3. dump_cram_frame (::quartus::fif)

The following table displays information for the `dump_cram_frame` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248	
Syntax	`dump_cram_frame [-h | -help] [-long_help] [-all] -frame` *<frame>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-all`		Option to display all information
	`-frame <frame>`		CRAM Frame ID
Description	`Dump CRAM frame information.`		
Example Usage	`dump_cram_frame -frame 3`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.

3.1.12.4. dump_mem (::quartus::fif)

The following table displays information for the `dump_mem` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248		
Syntax	`dump_mem [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Dump memory usage.`		
Example Usage	`dump_mem`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.12.5. dump_pr_bitstream (::quartus::fif)

The following table displays information for the `dump_pr_bitstream` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248		
Syntax	`dump_pr_bitstream [-h | -help] [-long_help] -id <id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-id <id>`		Static PR Bitstream ID
Description	`Dump static PR bitstream.`		
Example Usage	`dump_pr_bitstream -id 3`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.
	TCL_ERROR	1	ERROR: Parameter <*string*> has exceeded range.
	TCL_ERROR	1	ERROR: <*string*> mismatch. <*string*>

3.1.12.6. generate (::quartus::fif)

The following table displays information for the `generate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248		
Syntax	`generate [-h | -help] [-long_help] [-error_count` <*error_count*> `] [-error_index` <*error_index*> `] -frame` <*frame*> `-output` <*output*>		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-error_count` <*error_count*>	Number of errors to be injected	
	`-error_index` <*error_index*>	CRAM frame error bit locations	
	`-frame` <*frame*>	CRAM frame ID	
	`-output` <*output*>	Output file name	
Description	`Generates PR RBF file. Specifies the 'error_count' and 'error_index' to generate a PR RBF file with fault injected. Otherwise, the command will generate a PR RBF file without fault for external scrubbing.`		
Example Usage	`generate -frame 3 -output scrub.rbf` `generate -frame 3 -output inject.rbf -error_count 1 -error_index 100` `generate -frame 3 -output inject.rbf -error_count 2 -error_index 100 200`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.
	TCL_ERROR	1	ERROR: Missing parameter <*string*>.
	TCL_ERROR	1	ERROR: Parameter <*string*> has exceeded range.
	TCL_ERROR	1	ERROR: <*string*> mismatch. <*string*>
	TCL_ERROR	1	ERROR: Error writing to file (<*string*>).

3.1.12.7. get_frame_count (::quartus::fif)

The following table displays information for the `get_frame_count` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248	
Syntax	`get_frame_count [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`		Long help with examples and possible return values
Description	`Returns total frame count.`		
Example Usage	`get_frame_count`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.

3.1.12.8. get_frame_size (::quartus::fif)

The following table displays information for the `get_frame_size` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248		
Syntax	`get_frame_size [-h | -help] [-long_help] -frame` <*frame*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-frame` <*frame*>		CRAM frame ID
Description	`Returns frame size of the specific frame.`		
Example Usage	`get_frame_size -frame 3`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.

3.1.12.9. get_sector_information_sdm_based_fpga (::quartus::fif)

The following table displays information for the `get_sector_information_sdm_based_fpga` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248	
Syntax	`get_sector_information_sdm_based_fpga [-h | -help] [-long_help] -sector_index` <*sector_index*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-sector_index` <*sector_index*>	Sector Index

continued...

Description	Query information about sector with given index number.		
Example Usage	`get_sector_information_sdm_based_fpga -sector_index 25`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.
	TCL_ERROR	1	ERROR: Missing parameter <*string*>.

3.1.12.10. get_sensitive_location (::quartus::fif)

The following table displays information for the `get_sensitive_location` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248	
Syntax	`get_sensitive_location [-h | -help] [-long_help] [-count` <*count*> `] -frame` <*frame*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-count` <*count*>	Option to return specific number of indexes
	`-frame` <*frame*>	CRAM frame ID
Description	Returns a list of sensitive locations of the specific frame.	
Example Usage	`get_sensitive_location -frame 3` `get_sensitive_location -frame 3 -count 10`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.

3.1.12.11. get_sensitive_location_sdm_based_fpga (::quartus::fif)

The following table displays information for the `get_sensitive_location_sdm_based_fpga` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248	
Syntax	`get_sensitive_location_sdm_based_fpga [-h | -help] [-long_help] -bit_index` <*bit_index*> `-frame_index` <*frame_index*> `-sector_index` <*sector_index*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-bit_index` <*bit_index*>		Bit Index
	`-frame_index` <*frame_index*>		Frame Index
	`-sector_index` <*sector_index*>		Sector Index
Description	`Check if the bit is sensitive or not at specific location.`		
Example Usage	`get_sensitive_location_sdm_based_fpga -sector_index 25 -frame_index 4 -bit_index 3`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.

3.1.12.12. setup (::quartus::fif)

The following table displays information for the `setup` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248		
Syntax	`setup [-h | -help] [-long_help] -fif` <*fif*> `-rbf` <*rbf*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-fif` <*fif*>		FIF filename
	`-rbf` <*rbf*>		RBF filename
Description	`Setup FIF driver.`		
Example Usage	`setup -fif test.fif -rbf test.rbf`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has already been setup.

3.1.12.13. setup_sdm_based_fpga (::quartus::fif)

The following table displays information for the `setup_sdm_based_fpga` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248	
Syntax	`setup_sdm_based_fpga [-h | -help] [-long_help] -fif` <*fif*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-fif <fif>`		FIF filename
Description	`Setup FIF driver for SDM based FPGA.`		
Example Usage	`setup_sdm_based_fpga -fif test.fif`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver operation failed with error <*string*>.
	TCL_ERROR	1	ERROR: FIF driver has already been setup.
	TCL_ERROR	1	ERROR: FIF file is not compatible.

3.1.12.14. terminate (::quartus::fif)

The following table displays information for the `terminate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::fif on page 248		
Syntax	`terminate [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Terminate FIF driver.`		
Example Usage	`terminate`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FIF driver has not been setup. Use setup command to setup the FIF driver.

3.1.13. ::quartus::flng

The following table displays information for the **::quartus::flng** Tcl package:

Tcl Package and Version	::quartus::flng 1.0
Description	`This package contains no general description.`
Availability	`This package is loaded by default in the following executables:` `hdb_debug` `qpro` `qpro_sh` `quartus` `quartus_cdb` `quartus_eda` `quartus_fit` `quartus_ipgenerate` `quartus_map` `quartus_sh` `quartus_syn` `quartus_tlg` `qunb`

continued...

Tcl Commands	flng::add_object (::quartus::flng) on page 256 flng::add_property (::quartus::flng) on page 257 flng::bind_flow (::quartus::flng) on page 257 flng::delete_object (::quartus::flng) on page 258 flng::get_default_flow_run_name (::quartus::flng) on page 258 flng::get_flow_list (::quartus::flng) on page 259 flng::get_next_available_id (::quartus::flng) on page 259 flng::get_object (::quartus::flng) on page 260 flng::get_objects (::quartus::flng) on page 260 flng::get_option (::quartus::flng) on page 261 flng::get_property (::quartus::flng) on page 261 flng::get_task_command (::quartus::flng) on page 262 flng::get_task_status_property (::quartus::flng) on page 262 flng::init_repository (::quartus::flng) on page 263 flng::list_properties (::quartus::flng) on page 264 flng::monitor_flow (::quartus::flng) on page 264 flng::run_flow (::quartus::flng) on page 265 flng::run_flow_command (::quartus::flng) on page 265 flng::set_option (::quartus::flng) on page 266 flng::set_property (::quartus::flng) on page 267 flng::write_task_assignment_digest (::quartus::flng) on page 267 flng::write_task_checkpoint_written (::quartus::flng) on page 268 flng::write_task_finished (::quartus::flng) on page 269 flng::write_task_started (::quartus::flng) on page 270

3.1.13.1. flng::add_object (::quartus::flng)

The following table displays information for the `flng::add_object` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::add_object [-h | -help] [-long_help] [-name <name> ] [-number <number> ] -type <type>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		The name of the new flow or task object.
	`-number <number>`		The numeric part of the new FLNG object's name.
	`-type <type>`		The type of the new FLNG object.
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to add object.
	TCL_ERROR	1	ERROR: Invalid object type '<*string*>'.
	TCL_ERROR	1	ERROR: Object name is needed.
	TCL_ERROR	1	ERROR: Object number is needed.

3.1.13.2. flng::add_property (::quartus::flng)

The following table displays information for the `flng::add_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::add_property [-h | -help] [-long_help] -name <name> -object <object> -value <value> [-value_type <value_type> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <name>`	property name.	
	`-object <object>`	The object to which the new property belongs.	
	`-value <value>`	property value.	
	`-value_type <value_type>`	property value type.	
Description	`Add a property (a name/value pair) to a generic object`		
Example Usage	`flng::add_property -object <object> -name <property name> -value <property value> -value_type <value type>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to add object property: '<*string*>'.
	TCL_ERROR	1	ERROR: Invalid object '%'s.
	TCL_ERROR	1	ERROR: Invalid property value.
	TCL_ERROR	1	ERROR: Invalid property value type.
	TCL_ERROR	1	ERROR: Property '<*string*>' already exists. Cannot be added.

3.1.13.3. flng::bind_flow (::quartus::flng)

The following table displays information for the `flng::bind_flow` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::bind_flow [-h | -help] [-long_help] [-end <end> ] -flow <flow> [-start <start> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-end <end>`	The name of the last task to bind in a flow.
	`-flow <flow>`	The name of the flow to bind qsf.
	`-start <start>`	The name of the first task to bind in a flow
Description	`This command currently contains no help description.`	

continued...

Example Usage	`flng::bind_flow -flow flowname`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.13.4. flng::delete_object (::quartus::flng)

The following table displays information for the `flng::delete_object` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::delete_object [-h | -help] [-long_help] -object <object>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-object <object>`		Object to delete
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to delete object.

3.1.13.5. flng::get_default_flow_run_name (::quartus::flng)

The following table displays information for the `flng::get_default_flow_run_name` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::get_default_flow_run_name [-h | -help] [-long_help] -flow <flow>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-flow <flow>`		The flow name
Description	`Get the default flow run name.`		
Example Usage	`flng::get_default_flow_run`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to get object.

3.1.13.6. flng::get_flow_list (::quartus::flng)

The following table displays information for the `flng::get_flow_list` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::get_flow_list [-h | -help] [-long_help] -project <project> -project_path <project_path> -revision <revision>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-project <project>`		The name of the project
	`-project_path <project_path>`		The project path
	`-revision <revision>`		The name of the revision
Description	`This command currently contains no help description.`		
Example Usage	`flng::get_flow_list -project project -revision revision`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.13.7. flng::get_next_available_id (::quartus::flng)

The following table displays information for the `flng::get_next_available_id` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::get_next_available_id [-h | -help] [-long_help] -type <type>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-type <type>`		The type of the FLNG object.
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to get next available id
	TCL_ERROR	1	ERROR: Invalid object type '<*string*>'.

3.1.13.8. flng::get_object (::quartus::flng)

The following table displays information for the `flng::get_object` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::get_object [-h | -help] [-long_help] [-name <name> ] [-number <number> ] [-properties <properties> ] -type <type>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		The literal name of the objects to be retrieved.
	`-number <number>`		The numerical part of the name of the objects to be retrieved.
	`-properties <properties>`		Find object matching the specified properties.
	`-type <type>`		The type of objects to be retrieved.
Description	`Find a flow engine object matching type and name.`		
Example Usage	`flng::get_object -type task -name synthesis`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to get object.

3.1.13.9. flng::get_objects (::quartus::flng)

The following table displays information for the `flng::get_objects` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::get_objects [-h | -help] [-long_help] [-name <name> ] [-number <number> ] [-properties <properties> ] -type <type>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		The literal name of the objects to be retrieved.
	`-number <number>`		The numerical part of the name of the objects to be retrieved.
	`-properties <properties>`		Include only objects that the specified properties
	`-type <type>`		The type of objects to be retrieved.
Description	`Get a list of specific existing Flow Engine objects`		
Example Usage	`flng::get_objects -type task`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Fail to get objects.

3.1.13.10. flng::get_option (::quartus::flng)

The following table displays information for the `flng::get_option` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::get_option [-h | -help] [-long_help] -name <name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		option name.
Description	`Get option for the Flow Engine system`		
Example Usage	`flng::get_option -name <option name>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FLOW option '<*string*>' is invalid.

3.1.13.11. flng::get_property (::quartus::flng)

The following table displays information for the `flng::get_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::get_property [-h | -help] [-long_help] -name <name> -object <object>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		property name.
	`-object <object>`		The object to which the property belongs.
Description	`Get the value of a generic object's property.`		
Example Usage	`flng::get_property -object <object> -name <property name>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid object '%'s.
	TCL_ERROR	1	ERROR: Property '<*string*>' is invalid.

3.1.13.12. flng::get_task_command (::quartus::flng)

The following table displays information for the `flng::get_task_command` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::get_task_command [-h | -help] [-long_help] -task_instance` *<task_instance>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-task_instance` *<task_instance>*		The name of the task_instance
Description	`This command currently contains no help description.`		
Example Usage	`flng::get_task_command -task_instance task_instance`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.13.13. flng::get_task_status_property (::quartus::flng)

The following table displays information for the `flng::get_task_status_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::get_task_status_property [-h | -help] [-long_help] -property_name` *<property_name>* `-task_name` *<ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-property_name` *<property_name>*	The name of the property to lookup
	`-task_name` *<ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>*	The name of the task

continued...

Description	Returns the task status property of the specified task. The valid property_name values are: status: The status the task. The possible status are: no_status, scheduled, running and done. success: If task status is done, success indictes whether it completed successfully. errors: Returns the number of error messages. critical_warnings: Returns the number of critical warning messages. elapsed_time: Returns number seconds it the task has taken. id A internal id for the object run The name of the run name A user friendly name of the object process_id The process id that running the task hostname The name of the machine running the task percent The percent completed start_time The time the task started last_updated The time the task posted a progress update. end_time The time the task finished result The result of the execution. outdated Returns true if the task has been updated by either assignments or source file. dni Indicates if DNI database engine was active assignment_digest Internal hash of all the assignments. checkpoint Indicates if a checkpoint has been written out. imported_checkpoint Indicates if the checkpoint was imported
Example Usage	This command currently contains no example usage.

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Invalid status db property name
TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.
TCL_ERROR	1	ERROR: No task with the specified name exists.

3.1.13.14. flng::init_repository (::quartus::flng)

The following table displays information for the `flng::init_repository` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::init_repository [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	This command currently contains no help description.	

continued...

Example Usage	`flng::init_repository`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.13.15. flng::list_properties (::quartus::flng)

The following table displays information for the `flng::list_properties` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::list_properties [-h | -help] [-long_help] -object <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-object <object>`	The object whose properties are returned as a list of property names.
Description	`Get the list of property names from a generic object`	
Example Usage	`flng::list_properties -object <object>`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid object '%'s.

3.1.13.16. flng::monitor_flow (::quartus::flng)

The following table displays information for the `flng::monitor_flow` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::monitor_flow [-h | -help] [-long_help] -flow <flow>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-flow <flow>`	The name of the flow to run.
Description	`This command currently contains no help description.`	
Example Usage	`flng::monitor_flow -flow flowname`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.13.17. flng::run_flow (::quartus::flng)

The following table displays information for the `flng::run_flow` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::run_flow [-h | -help] [-long_help] [-enable_heartbeat <enable_heartbeat> ] [-end <end> ] -flow <flow> [-print_only] [-resume] [-start <start> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-enable_heartbeat <enable_heartbeat>`	Output a regular heartbeat for Quartus to monitor this flow using Monitor Mode.	
	`-end <end>`	The name of the last task to run in a flow.	
	`-flow <flow>`	The name of the flow to run.	
	`-print_only`	Print what will be run and do not execute.	
	`-resume`	Resume executing flow from where it left off.	
	`-start <start>`	The name of the first task to run in a flow	
Description	`This command currently contains no help description.`		
Example Usage	`flng::run_flow -flow flowname`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.13.18. flng::run_flow_command (::quartus::flng)

The following table displays information for the `flng::run_flow_command` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::run_flow_command [-h | -help] [-long_help] [-command <command> ] [-end <end> ] -flow <flow> [-print_only] [-resume] [-start <start> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-command <command>`	The command is used by "Custom" flow execution
	`-end <end>`	The name of the last task to run in a flow.
	`-flow <flow>`	The name of the flow to run.
	`-print_only`	Print what will be run and do not execute.
	`-resume`	Resume executing flow from where it left off.
	`-start <start>`	The name of the first task to run in a flow
Description	`This command currently contains no help description.`	

continued...

Example Usage	`flng::run_flow_command -flow flowname`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: run_flow_command with specified flow '<*string*>' completed successfully.
	TCL_OK	0	WARNING: Option -command should be used with Custom flow. The option is ignored.
	TCL_ERROR	1	ERROR: Option -command <command> is needed for Custom flow
	TCL_ERROR	1	ERROR: Failed to create a specified flow '<*string*>'.
	TCL_ERROR	1	ERROR: Failed to initialize the flow repository.
	TCL_ERROR	1	ERROR: Failed to run compilation flow <*string*>. The expected compilation flow name is <*string*> for current opened project.
	TCL_ERROR	1	ERROR: Cannot find the end task '<*string*>'. Specify an existing task name.
	TCL_ERROR	1	ERROR: Cannot find the flow definition for specified flow '<*string*>'.
	TCL_ERROR	1	ERROR: Cannot find the start task '<*string*>'. Specify an existing task name.
	TCL_ERROR	1	ERROR: Please open Quartus project before executing the command.
	TCL_ERROR	1	ERROR: run_flow_command with specified flow '<*string*>' failed: <*string*>.

3.1.13.19. flng::set_option (::quartus::flng)

The following table displays information for the `flng::set_option` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255		
Syntax	`flng::set_option [-h | -help] [-long_help] -name` <*name*> `-value` <*value*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name` <*name*>		option name.
	`-value` <*value*>		option value.
Description	`Set options for the Flow Engine system`		
Example Usage	`flng::set_option -name <option name> -value <option value>`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: FLOW option '<*string*>' is invalid.

3.1.13.20. flng::set_property (::quartus::flng)

The following table displays information for the `flng::set_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::set_property [-h | -help] [-long_help] -name <name> -object <object> -value <value>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	property name.
	`-object <object>`	The object to which the property belongs.
	`-value <value>`	property value.
Description	`Update property in term of name/value pair to a generic flow object`	
Example Usage	`flng::set_property -object <object> -name <property name> -value <property value>`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid object '%'s.
	TCL_ERROR	1	ERROR: Property '<*string*>' is invalid.

3.1.13.21. flng::write_task_assignment_digest (::quartus::flng)

The following table displays information for the `flng::write_task_assignment_digest` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::write_task_assignment_digest [-h | -help] [-long_help] [-digest <digest>] -task_name <ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-digest <digest>`	The assignment digest. If not specified, gets current digest.
	`-task_name <ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|`	The name of the task

continued...

	fitter_finalize|*sta_signoff*| *power_analysis*|*assembler*| *sasic_handoff_flow*| *countermeasures_flow*| *rdm_handoff_flow*|*ds_tool*|*dr_tool*| *simulation*| *analysis_and_elaboration_lint*>	
Description	`This command currently contains no help description.`	
Example Usage	`This command currently contains no example usage.`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.
	TCL_ERROR	1	ERROR: No task with the specified name exists.
	TCL_ERROR	1	ERROR: Unable to access status db

3.1.13.22. flng::write_task_checkpoint_written (::quartus::flng)

The following table displays information for the `flng::write_task_checkpoint_written` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::write_task_checkpoint_written [-h | -help] [-long_help] [-checkpoint` *<checkpoint>* `] -task_name` *<ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-checkpoint` *<checkpoint>*	name of checkpoint that has been written.
	`-task_name` *<ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>*	The name of the task

continued...

Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Failed to write to status db.
	TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.
	TCL_ERROR	1	ERROR: No task with the specified name exists.
	TCL_ERROR	1	ERROR: Unable to access status db

3.1.13.23. flng::write_task_finished (::quartus::flng)

The following table displays information for the `flng::write_task_finished` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::write_task_finished [-h | -help] [-long_help] [-critical_warnings` *<critical_warnings>* `] -dni` *<dni>* `[-errors` *<errors>* `] -success` *<success>* `-task` *<ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-critical_warnings` *<critical_warnings>*	Set critical warnings message count
	`-dni` *<dni>*	Set to 1 if task runs in dni
	`-errors` *<errors>*	Set error message count
	`-success` *<success>*	Set to 1 if task finished successfully
	`-task` *<ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>*	Specifies a task name

continued...

Description	`flng::write_task_finished -task analysis_and_synthesis -success 1 -errors 0 -critical_warnings 0 -dni 1`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Cannot process -critical_warnigns argument.
	TCL_ERROR	1	ERROR: Cannot process -dni argument. Value should be 0 or 1
	TCL_ERROR	1	ERROR: Cannot process -errors argument.
	TCL_ERROR	1	ERROR: Cannot process -success argument. Value should be 0 or 1
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.
	TCL_ERROR	1	ERROR: No task with the specified name exists.
	TCL_ERROR	1	ERROR: Unable to access status db

3.1.13.24. flng::write_task_started (::quartus::flng)

The following table displays information for the `flng::write_task_started` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flng on page 255	
Syntax	`flng::write_task_started [-h | -help] [-long_help] -dni <dni> -task <ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-dni <dni>`	Set to 1 if task runs in dni
	`-task <ip_generation|tile_ip_generation|design_analysis|logic_generation|pcc_generation|create_pcc_ip_projects|pcc_run_all_dynamic_tasks|analysis_and_synthesis|analysis_and_elaboration|synthesis|sta_early|eda_netlist_writer|fitter|fitter_implement|fitter_plan|fitter_place|fitter_route|fitter_fastforward_timing|fitter_retime|fitter_finalize|sta_signoff|power_analysis|assembler|sasic_handoff_flow|countermeasures_flow|`	Specifies a task name

continued...

	rdm_handoff_flow|ds_tool|dr_tool|simulation|analysis_and_elaboration_lint>		
Description	`flng::write_task_started -task analysis_and_synthesis -dni 1`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Cannot process -dni argument. Value should be 0 or 1
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.
	TCL_ERROR	1	ERROR: Unable to access status db

3.1.14. ::quartus::flow

The following table displays information for the **::quartus::flow** Tcl package:

Tcl Package and Version	::quartus::flow 1.1
Description	`This package contains the set of Tcl functions for running flows or command-line executables.`
Availability	`This package is loaded by default in the following executables:` `qpro` `qpro_sh` `quartus` `quartus_cdb` `quartus_sh` `This package is available for loading in the following executables:` `hdb_debug` `quartus_drc` `quartus_eda` `quartus_fit` `quartus_ipgenerate` `quartus_map` `quartus_si` `quartus_sim` `quartus_sta` `quartus_stp` `quartus_syn` `quartus_tlg`
Tcl Commands	execute_flow (::quartus::flow) on page 272 execute_module (::quartus::flow) on page 274 get_flow_templates (::quartus::flow) on page 275 get_status_db_property (::quartus::flow) on page 276 write_flow_assignment_digest (::quartus::flow) on page 276 write_flow_finished (::quartus::flow) on page 277 write_flow_started (::quartus::flow) on page 277 write_flow_template (::quartus::flow) on page 278

3.1.14.1. execute_flow (::quartus::flow)

The following table displays information for the `execute_flow` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flow on page 271	
Syntax	`execute_flow [-h | -help] [-long_help] [-analysis_and_elaboration] [-check_ios] [-check_netlist] [-compile] [-dni] [-dont_export_assignments] [-eco <value>] [-export_database] [-finalize] [-flow_args <Tcl list of name value pairs args to pass to the flow>] [-generate_functional_sim_netlist] [-implement] [-import_database] [-incremental_compilation_export] [-incremental_compilation_import] [-ip_upgrade] [-quick_elaboration] [-signalprobe] [-simulation]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-analysis_and_elaboration`	Option to run Analysis & Elaboration
	`-check_ios`	Option to run I/O assignment analysis
	`-check_netlist`	Option to run Check and Save Netlist
	`-compile`	Option to run a full compilation
	`-dni`	Option to request flow be executed in DNI mode.
	`-dont_export_assignments`	Option not to export assignments to file. By default, this command exports assignments before running command-line executables.
	`-eco <value>`	Option to run a Fitter ECO compilation
	`-export_database`	Option to export a version-compatible database
	`-finalize`	Option to run algorithms to prepare design for programming.
	`-flow_args <Tcl list of name value pairs args to pass to the flow>`	Option to specify flow args
	`-generate_functional_sim_netlist`	Option to generate a functional simulation netlist
	`-implement`	Option to run compilation up to route stage and skipping all time intensive algorithms after.
	`-import_database`	Option to import a version-compatible database
	`-incremental_compilation_export`	Option to export a design partition into a Quartus Prime Exported Partition (QXP) file
	`-incremental_compilation_import`	Option to import one or more Quartus Prime Exported Partition (QXP) files into the design partitions of the current project
	`-ip_upgrade`	Option to run a ip upgrade
	`-quick_elaboration`	Option to run Quick Elaboration
	`-signalprobe`	Option to run a Signal Probe compilation
	`-simulation`	Option to run simulation

continued...

Description

```
Runs one or more of the command-line executables using one of the
predefined flows, such as "-compile" or "-signalprobe". You can run
only one flow at a time, so you must use only one option.

Some flows have limited device support or other limitations based on
the features used. See documentation for the features in question for
details.

The "-export_database" and "-import_database" options use the value of
the VER_COMPATIBLE_DB_DIR assignment for the version-compatible
database files directory, defaulting to "export_db".

The "-incremental_compilation_export" option uses the value of the
INCREMENTAL_COMPILATION_EXPORT_FILE global assignment for the path of
the Quartus Prime Exported Partition (QXP) file to be created. The value
of the INCREMENTAL_COMPILATION_EXPORT_PARTITION_NAME global assignment
should specify the name of the partition to be exported. The value of
the INCREMENTAL_COMPILATION_EXPORT_NETLIST_TYPE global assignment
(which can either have value POST_SYNTH or POST_FIT) determines
whether post-synthesis or post-fitting results should be
exported. Finally, the value of the
INCREMENTAL_COMPILATION_EXPORT_ROUTING global assignment specifies
whether routing should be exported when a post-fit netlist is
generated.

The "-incremental_compilation_import" option uses the following
partition assignments to determine the location of the QXP files, and
how importation should be performed, on a per-partition basis:

PARTITION_IMPORT_FILE
PARTITION_IMPORT_PROMOTE_ASSIGNMENTS
PARTITION_IMPORT_NEW_ASSIGNMENTS
PARTITION_IMPORT_EXISTING_ASSIGNMENTS
PARTITION_IMPORT_EXISTING_LOGICLOCK_REGIONS

All assignments are exported first automatically, as if you
called the "export_assignments" command first, unless the
-dont_export_assignments option is specified.

You must use the Tcl command "catch" to determine whether the
predefined flow ran successfully or not, as in the following example:

if {[catch {execute_flow -compile} result]} {
    puts "\nResult: $result\n"
    puts "ERROR: Compilation failed. See report files.\n"
} else {
    puts "\nINFO: Compilation was successful.\n"
}
```

Example Usage

```
# To run quartus_map, quartus_fit, quartus_sta, quartus_asm
# or other executables based on options. (Refer to "Using
# Compilation Flows," "Compiling Designs," and "Specifying
# Compiler Settings" in Quartus Prime online Help for more
# information.)
execute_flow -compile

# To determine if compilation was successful or not
# and print out a personalized message.
if {[catch {execute_flow -compile} result]} {
    puts "\nResult: $result\n"
    puts "ERROR: Compilation failed. See report files.\n"
} else {
    puts "\nINFO: Compilation was successful.\n"
}

        # To perform a full compilation
            execute_flow -compile
        # To perform a ip_upgrade with flow_args to specify variation_files
            execute_flow -ip_upgrade -flow_args "variation_files=\"a.ip;b.ip\""
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't run multiple flows simultaneously. Wait for current flow to complete.
TCL_ERROR	1	ERROR: Flow doesn't exist: <*string*>. Make sure the specified flow exists.

continued...

	TCL_ERROR	1	ERROR: The flow args are invalid. It should be a legal Tcl list.
	TCL_ERROR	1	ERROR: Only one flow option is allowed. Only one flow can be run for a single command call. If multiple flows are required, use multiple commands.
	TCL_ERROR	1	ERROR: Can't find active revision. Make sure there is an open, active revision name. Use the -revision option of project_open, project_new, or use set_current_revision.
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: Error(s) found while running an executable. See report file(s) for error message(s). Message log indicates which executable was run last.
	TCL_ERROR	1	ERROR: Option -<*string*> is illegal in the Quartus Prime User Interface. Specify a different option or use a similar command from the Processing menu.
	TCL_ERROR	1	ERROR: At least one option is required. Specify at least one option.

3.1.14.2. execute_module (::quartus::flow)

The following table displays information for the `execute_module` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flow on page 271	
Syntax	`execute_module [-h | -help] [-long_help] [-args` <*arguments*> `] [-classic] [-dni] [-dont_export_assignments] [-tool` <*asm*|*cdb*|*drc*|*eda*|*fit*|*map*|*syn*|*pow*|*sta*|*stp*|*sim*|*si*|*cpf*|*ipg*|*pfg*|*qtlg*|*quick_elaboration*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-args` <*arguments*>	Option to specify arguments for the executable
	`-classic`	Option to request flow be executed in Classic mode.
	`-dni`	Option to request tool option be executed in DNI mode.
	`-dont_export_assignments`	Option not to export assignments to file. By default, this command exports assignments before running command-line executables.
	`-tool` <*asm*|*cdb*|*drc*|*eda*|*fit*|*map*|*syn*|*pow*|*sta*|*stp*|*sim*|*si*|*cpf*|*ipg*|*pfg*|*qtlg*|*quick_elaboration*>	Option to run the specified executable
Description		

```
Runs one of the command-line executables, such as quartus_map or
quartus_fit. If the -args option is specified, the arguments are
passed to the command-line executable.

All assignments are exported automatically first, as if the
"export_assignments" command was called first, unless
-dont_export_assignments option is specified.

You must use the Tcl command "catch" to determine whether the
command-line executable ran successfully or not, as in the following
example:

if {[catch {execute_module -tool map} result]} {
    puts "\nResult: $result\n"
    puts "ERROR: Analysis & Synthesis failed. See the report file.\n"
```

continued...

```
} else {
    puts "\nINFO: Analysis & Synthesis was successful.\n"
}
```

Example Usage

```
# Run quartus_map using device family Stratix and device part EP1S10B672C6.
execute_module -tool map -args "--family=Stratix --part=EP1S10B672C6"

# Compile using a set of executables
execute_module -tool map
execute_module -tool fit
execute_module -tool sta
execute_module -tool asm
execute_module -tool eda

# To determine if Analysis & Synthesis was successful or not
# and print out a personalized message.
if {[catch {execute_module -tool map} result]} {
    puts "\nResult: $result\n"
    puts "ERROR: Analysis & Synthesis failed. See the report file.\n"
} else {
    puts "\nINFO: Analysis & Synthesis was successful.\n"
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't run multiple flows simultaneously. Wait for current flow to complete.
TCL_ERROR	1	ERROR: Can't find active revision. Make sure there is an open, active revision name. Use the -revision option of project_open, project_new, or use set_current_revision.
TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
TCL_ERROR	1	ERROR: Error(s) found while running an executable. See report file(s) for error message(s). Message log indicates which executable was run last.
TCL_ERROR	1	ERROR: Option is required: -tool. Specify the -tool option.

3.1.14.3. get_flow_templates (::quartus::flow)

The following table displays information for the `get_flow_templates` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flow on page 271	
Syntax	`get_flow_templates [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`This command currently contains no help description.`	
Example Usage	`This command currently contains no example usage.`	

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.

3.1.14.4. get_status_db_property (::quartus::flow)

The following table displays information for the `get_status_db_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flow on page 271		
Syntax	`get_status_db_property [-h | -help] [-long_help] -property_name <property_name> -task_name <task_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-property_name <property_name>`		The name of the property to lookup
	`-task_name <task_name>`		The name of the task
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid status db property name
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.
	TCL_ERROR	1	ERROR: No task with the specified name exists.

3.1.14.5. write_flow_assignment_digest (::quartus::flow)

The following table displays information for the `write_flow_assignment_digest` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flow on page 271		
Syntax	`write_flow_assignment_digest [-h | -help] [-long_help] [-digest <digest> ] -flow_name <flow_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-digest <digest>`		The assignment digest. If not specified, gets current digest.
	`-flow_name <flow_name>`		The name of the flow that started
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.
	TCL_ERROR	1	ERROR: Unable to load status db

3.1.14.6. write_flow_finished (::quartus::flow)

The following table displays information for the `write_flow_finished` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flow on page 271		
Syntax	`write_flow_finished [-h | -help] [-long_help] [-critical_warnings` *<critical_warnings>* `] [-errors` *<errors>* `] -flow_name` *<flow_name>* `-success` *<success>*		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-critical_warnings` *<critical_warnings>*	Set critical warnings message count	
	`-errors` *<errors>*	Set error message count	
	`-flow_name` *<flow_name>*	The name of the flow that finished	
	`-success` *<success>*	Set to 1 if flow finished successfully	
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.

3.1.14.7. write_flow_started (::quartus::flow)

The following table displays information for the `write_flow_started` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flow on page 271	
Syntax	`write_flow_started [-h | -help] [-long_help] -flow_name` *<flow_name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-flow_name` *<flow_name>*	The name of the flow that started
Description	`This command currently contains no help description.`	
Example Usage	`This command currently contains no example usage.`	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: No revision is currently open. Open a revsion.

3.1.14.8. write_flow_template (::quartus::flow)

The following table displays information for the `write_flow_template` Tcl command:

Tcl Package and Version	Belongs to ::quartus::flow on page 271		
Syntax	`write_flow_template [-h | -help] [-long_help] [-directory` *<directory>* `] -flow_name` *<flow_name>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-directory` *<directory>*		Optional name to use as destination for flow template directory
	`-flow_name` *<flow_name>*		The name of the flow template to write
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.

3.1.15. ::quartus::insystem_memory_edit

The following table displays information for the **::quartus::insystem_memory_edit** Tcl package:

Tcl Package and Version	::quartus::insystem_memory_edit 1.0
Description	`This package contains the set of Tcl functions for reading and editing the contents of memory in an Intel device using the In-System Memory Content Editor.`
Availability	`This package is loaded by default in the following executables:` `quartus_stp` `quartus_stp_tcl`
Tcl Commands	begin_memory_edit (::quartus::insystem_memory_edit) on page 279 end_memory_edit (::quartus::insystem_memory_edit) on page 279 get_editable_mem_instances (::quartus::insystem_memory_edit) on page 280 read_content_from_memory (::quartus::insystem_memory_edit) on page 281 save_content_from_memory_to_file (::quartus::insystem_memory_edit) on page 282 update_content_to_memory_from_file (::quartus::insystem_memory_edit) on page 283 write_content_to_memory (::quartus::insystem_memory_edit) on page 284

3.1.15.1. begin_memory_edit (::quartus::insystem_memory_edit)

The following table displays information for the `begin_memory_edit` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_memory_edit on page 278	
Syntax	`begin_memory_edit [-h | -help] [-long_help] -device_name` *<device name>* `-hardware_name` *<hardware name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-device_name` *<device name>*	Name of the device that holds the editable memory instances
	`-hardware_name` *<hardware name>*	Name of the hardware that connects to the JTAG chain
Description	`Start the memory editing sequence. The editing sequence should be terminated with end_memory_edit. The sequence does not have to be terminated unless the device configuration is changed or a different device is edited.` `The hardware and device name can be obtained with the get_hardware_names and get_device_names commands from the jtag package.`	
Example Usage	`# Instance 0 is configured as {0 1024 8 RW ROM/RAM mem0}` `# Initiate a editing sequence` `begin_memory_edit -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `# Write memory content using binary string` `write_content_to_memory -instance_index 0 -start_address 575 -word_count 2 -content "0000001011011100"` `# Read back memory content in binary string written` `puts [read_content_from_memory -instance_index 0 -start_address 575 -word_count 2 ]` `# Write memory content using hexadecimal string` `write_content_to_memory -instance_index 0 -start_address 575 -word_count 2 -content "E2F1" -content_in_hex` `# Read back memory content in hexadecimal string written` `puts [read_content_from_memory -instance_index 0 -start_address 575 -word_count 2 -content_in_hex]` `# End the editing sequence` `end_memory_edit`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful
	TCL_ERROR	1 / ERROR: The specified device is not found.
	TCL_ERROR	1 / ERROR: A memory edit sequence has been started. End it first before starting a another one.
	TCL_ERROR	1 / ERROR: The specified hardware is not found.

3.1.15.2. end_memory_edit (::quartus::insystem_memory_edit)

The following table displays information for the `end_memory_edit` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_memory_edit on page 278
Syntax	`end_memory_edit [-h | -help] [-long_help]`

continued...

Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Terminate the memory editing sequence. The sequence does not have to be terminated unless the device configuration is changed or a different device is edited.`	
Example Usage	`# Instance 0 is configured as {0 1024 8 RW ROM/RAM mem0}` `# Initiate a editing sequence` `begin_memory_edit -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `# Write memory content using binary string` `write_content_to_memory -instance_index 0 -start_address 575 -word_count 2 -content "0000010011011100"` `# Read back memory content in binary string written` `puts [read_content_from_memory -instance_index 0 -start_address 575 -word_count 2 ]` `# Write memory content using hexadecimal string` `write_content_to_memory -instance_index 0 -start_address 575 -word_count 2 -content "E2F1" -content_in_hex` `# Read back memory content in hexadecimal string written` `puts [read_content_from_memory -instance_index 0 -start_address 575 -word_count 2 -content_in_hex]` `# End the editing sequence` `end_memory_edit`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful
	TCL_ERROR	1 / ERROR: A memory edit sequence has not been started.

3.1.15.3. get_editable_mem_instances (::quartus::insystem_memory_edit)

The following table displays information for the `get_editable_mem_instances` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_memory_edit on page 278	
Syntax	`get_editable_mem_instances [-h | -help] [-long_help] -device_name` *<device name>* `-hardware_name` *<hardware name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-device_name` *<device name>*	Name of the device that holds the editable memory instances
	`-hardware_name` *<hardware name>*	Name of the hardware that connects to the JTAG chain
Description	`Retrieve a list of editable memory, ROM, or lpm_constant instances.` `A list is returned, each element of which shows the configuration of each instance. This element is an another list that specifies the configuration in the following order: <instance index> <depth> <width> <read/write mode> <instance type> <instance name>. The <read/write mode> can be either "RW" or "W"; <instance type> can be either "ROM/RAM" or "CONSTANT". An example showing a list of two instances of different types is shown below:`  `{0 1024 8 RW ROM/RAM mem0} {1 1 32 RW CONSTANT con0}`	

continued...

	The hardware and device name can be obtained with the get_hardware_names and get_device_names commands from the jtag package. It is recommended that you call this command before the TCL command, begin_memory_edit. Within a memory edit sequence, this command can be applied only to the same device, on which the memory edit sequence has started.		
Example Usage	`# List information of all editable memories` `puts "Information on all editable memories:"` `puts "index,depth,width,mode,type,name"` `foreach instance [get_editable_mem_instances -hardware_name "USB-Blaster \[USB-0\]" -` `device_name "@1: EP1S25/_HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"] {` `  puts "[lindex $instance 0],[lindex $instance 1],[lindex $instance 2],[lindex $instance 3],` `[lindex $instance 4],[lindex $instance 5]"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The specified hardware is not found.
	TCL_ERROR	1	ERROR: The TCL command get_editable_mem_instances is called within a memory edit sequence for a different device. End the memory edit first.
	TCL_ERROR	1	ERROR: An internal TCL interpreter error occurred.

3.1.15.4. read_content_from_memory (::quartus::insystem_memory_edit)

The following table displays information for the `read_content_from_memory` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_memory_edit on page 278	
Syntax	`read_content_from_memory [-h | -help] [-long_help] [-content_in_hex] -instance_index` *<instance index>* `-start_address` *<starting address>* `[-timeout` *<timeout>* `] -word_count` *<word count>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-content_in_hex`	The memory content string is represented in hexadecimal format
	`-instance_index` *<instance index>*	Index of the editable memory instance to read
	`-start_address` *<starting address>*	The lowest memory address to be read
	`-timeout` *<timeout>*	amount of time in milliseconds allocated before read times out. Defaults to 10 seconds
	`-word_count` *<word count>*	The number of contiguous memory words to be read
Description	Retrieves the memory content represented in the bit stream from the specified editable memory instance starting from the specified address. The memory content string is in the same format as the input content string in the TCL command write_content_to_memory.	

continued...

Example Usage	<pre># Instance 0 is configured as {0 1024 8 RW ROM/RAM mem0} # Initiate a editing sequence begin_memory_edit -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)" # Write memory content using binary string write_content_to_memory -instance_index 0 -start_address 575 -word_count 2 -content "0000010011011100" # Read back memory content in binary string written puts [read_content_from_memory -instance_index 0 -start_address 575 -word_count 2 -timeout 30000] # Write memory content using hexadecimal string write_content_to_memory -instance_index 0 -start_address 575 -word_count 2 -content "E2F1" - content_in_hex # Read back memory content in hexadecimal string written puts [read_content_from_memory -instance_index 0 -start_address 575 -word_count 2 - content_in_hex] # End the editing sequence end_memory_edit</pre>

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: A memory edit sequence has not been started.
	TCL_ERROR	1	ERROR: The specified word count and the starting address exceeds the specified memory buffer size.
	TCL_ERROR	1	ERROR: The specified editable memory instance index is invalid.
	TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure or poor signal integrity in the JTAG chain.
	TCL_ERROR	1	ERROR: The device is locked by another application.

3.1.15.5. save_content_from_memory_to_file (::quartus::insystem_memory_edit)

The following table displays information for the `save_content_from_memory_to_file` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_memory_edit on page 278	
Syntax	`save_content_from_memory_to_file [-h | -help] [-long_help] -instance_index <instance index> -mem_file_path <path> -mem_file_type <type> [-timeout <timeout> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-instance_index <instance index>`	Index of the editable memory instance to read
	`-mem_file_path <path>`	Path to the memory file in which to save the memory content
	`-mem_file_type <type>`	Type of the memory file such as "mif" or "hex"
	`-timeout <timeout>`	amount of time in milliseconds allocated before read times out. Defaults to 10 seconds

continued...

Description	Retrieves the entire memory contents from the specified editable memory instance starting from address 0 and saves it into the specified memory file.		
Example Usage	`# Initiate a editing sequence` `begin_memory_edit -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `# Write memory content using the hex memory file` `update_content_to_memory_from_file -instance_index 0 -mem_file_path "image_8x1024.hex" - mem_file_type hex` `# Read memory content and save back to a hex memory file` `save_content_from_memory_to_file -instance_index 0 -mem_file_path "exported_image_8x1024.hex" - mem_file_type hex` `# Write memory content using the mif memory file` `update_content_to_memory_from_file -instance_index 0 -mem_file_path "exported_image_8x1024.mif" -mem_file_type mif` `# Read memory content and save back to a mif memory file` `save_content_from_memory_to_file -instance_index 0 -mem_file_path "image_8x1024.mif" - mem_file_type mif -timeout 30000` `# End the editing sequence` `end_memory_edit`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: A memory edit sequence has not been started.
	TCL_ERROR	1	ERROR: The specified memory file cannot be written to.
	TCL_ERROR	1	ERROR: The specified file type is either invalid or unsupported by this command.
	TCL_ERROR	1	ERROR: The specified editable memory instance index is invalid.
	TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure or poor signal integrity in the JTAG chain.
	TCL_ERROR	1	ERROR: The device is locked by another application.

3.1.15.6. update_content_to_memory_from_file (::quartus::insystem_memory_edit)

The following table displays information for the `update_content_to_memory_from_file` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_memory_edit on page 278	
Syntax	`update_content_to_memory_from_file [-h | -help] [-long_help] -instance_index` <*instance index*> `-mem_file_path` <*path*> `-mem_file_type` <*file type*> `[-timeout` <*timeout*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-instance_index` <*instance index*>	Index of the editable memory instance to modify
	`-mem_file_path` <*path*>	Path to the memory file to load the memory content
	`-mem_file_type` <*file type*>	Type of the memory file such as "mif" or "hex"

continued...

	`-timeout <timeout>`	amount of time in milliseconds allocated before write times out. Defaults to 10 seconds
Description	`Writes the data stored in the memory file into the specified memory instance starting from address 0.`	
Example Usage	`# Initiate a editing sequence` `begin_memory_edit -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `# Write memory content using the hex memory file` `update_content_to_memory_from_file -instance_index 0 -mem_file_path "image_8x1024.hex" -mem_file_type hex` `# Read memory content and save back to a hex memory file` `save_content_from_memory_to_file -instance_index 0 -mem_file_path "exported_image_8x1024.hex" -mem_file_type hex` `# Write memory content using the mif memory file` `update_content_to_memory_from_file -instance_index 0 -mem_file_path "exported_image_8x1024.mif" -mem_file_type mif -timeout 30000` `# Read memory content and save back to a mif memory file` `save_content_from_memory_to_file -instance_index 0 -mem_file_path "image_8x1024.mif" -mem_file_type mif` `# End the editing sequence` `end_memory_edit`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: A memory edit sequence has not been started.
	TCL_ERROR	1	ERROR: The specified memory file cannot be read because the content is corrupt or the configuration does not match the memory to be updated.
	TCL_ERROR	1	ERROR: The specified memory file cannot be opened.
	TCL_ERROR	1	ERROR: The specified file type is either invalid or unsupported by this command.
	TCL_ERROR	1	ERROR: The specified editable memory instance index is invalid.
	TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure or poor signal integrity in the JTAG chain.
	TCL_ERROR	1	ERROR: The device is locked by another application.

3.1.15.7. write_content_to_memory (::quartus::insystem_memory_edit)

The following table displays information for the `write_content_to_memory` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_memory_edit on page 278	
Syntax	`write_content_to_memory [-h | -help] [-long_help] -content <content string> [-content_in_hex] -instance_index <instance index> -start_address <starting address> [-timeout <timeout> ] -word_count <word count>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-content <content string>`	A string that represents all word values concatenated together in order in either binary or hexadecimal format

continued...

	`-content_in_hex`	The memory content string is represented in hexadecimal format
	`-instance_index` *<instance index>*	Index of the editable memory instance to modify
	`-start_address` *<starting address>*	The lowest memory address to be modified
	`-timeout` *<timeout>*	amount of time in milliseconds allocated before write times out. Defaults to 10 seconds
	`-word_count` *<word count>*	The number of contiguous memory words to be modified

Description

```
       Writes the data represented in the bit stream into the
specified editable memory instance starting from the
specified address. It returns the number of successful writes.

The bit stream should be ordered by word from high
address to low address, contiguously without gaps or
delimiters. If the starting address is ADDR, and word
count is N, the order is <word @ ADDR + N - 1>
... <word @ ADDR + 1><word @ ADDR> In each word, the
MSB is on the left, LSB is on the right. The bit
stream can be in either binary or hexadecimal. For
example, if the word width is 8, and two words, 1 and
128, are written to address 0 and 1 respectively, the
bitstream should be "1000000000000001" in binary or
"8001" in hexadecimal. The TCL command is
write_content_to_memory -instance_index 0
-start_address 0 -word_count 2 -content
"1000000000000001" or write_content_to_memory
-instance_index 0 -start_address 0 -word_count 2
-content "8001" -content_in_hex
```

Example Usage

```
# Instance 0 is configured as {0 1024 8 RW ROM/RAM mem0}

# Initiate a editing sequence
begin_memory_edit -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/
_HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"

# Write memory content using binary string
write_content_to_memory -instance_index 0 -start_address 575 -word_count 2 -content
"0000001011011100" -timeout 30000

# Read back memory content in binary string written
puts [read_content_from_memory -instance_index 0 -start_address 575 -word_count 2 ]

# Write memory content using hexadecimal string
write_content_to_memory -instance_index 0 -start_address 575 -word_count 2 -content "E2F1" -
content_in_hex

# Read back memory content in hexadecimal string written
puts [read_content_from_memory -instance_index 0 -start_address 575 -word_count 2 -
content_in_hex]

# End the editing sequence
end_memory_edit
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Data specified in the string does not match the number of bits to update the memory of the specified number of words.
TCL_ERROR	1	ERROR: A memory edit sequence has not been started.
TCL_ERROR	1	ERROR: The specified word count and the starting address exceeds the specified memory buffer size.
TCL_ERROR	1	ERROR: The specified editable memory instance index is invalid.

continued...

	TCL_ERROR	1	ERROR: The specified editable memory instance index is invalid.
	TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure or poor signal integrity in the JTAG chain.
	TCL_ERROR	1	ERROR: The device is locked by another application.

3.1.16. ::quartus::insystem_source_probe

The following table displays information for the **::quartus::insystem_source_probe** Tcl package:

Tcl Package and Version	::quartus::insystem_source_probe 1.0
Description	This package contains the set of Tcl functions for using the In-System Sources and Probes feature to interact with your design in an Intel device.
Availability	This package is loaded by default in the following executables: quartus_stp quartus_stp_tcl
Tcl Commands	end_insystem_source_probe (::quartus::insystem_source_probe) on page 286 get_insystem_source_probe_instance_info (::quartus::insystem_source_probe) on page 287 read_probe_data (::quartus::insystem_source_probe) on page 288 read_source_data (::quartus::insystem_source_probe) on page 288 start_insystem_source_probe (::quartus::insystem_source_probe) on page 289 write_source_data (::quartus::insystem_source_probe) on page 290

3.1.16.1. end_insystem_source_probe (::quartus::insystem_source_probe)

The following table displays information for the `end_insystem_source_probe` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_source_probe on page 286		
Syntax	`end_insystem_source_probe [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	This command releases the JTAG chain. Use when finished performing In-System Sources and Probes transactions.		
Example Usage	`#List probe data of instance 0` `start_insystem_source_probe -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `puts "probe data of instance 0"` `puts [read_probe_data -instance_index 0]` `end_insystem_source_probe`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The specified hardware is not found.

continued...

	TCL_ERROR	1	ERROR: An internal Tcl interpreter error occurred.
	TCL_ERROR	1	ERROR: No In-System Sources and Probes instance was found.
	TCL_ERROR	1	ERROR: The In-System Sources and Probes instance was not started. This command cannot be used unless the In-System Sources and Probes trasaction is started.

3.1.16.2. get_insystem_source_probe_instance_info (::quartus::insystem_source_probe)

The following table displays information for the `get_insystem_source_probe_instance_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_source_probe on page 286		
Syntax	`get_insystem_source_probe_instance_info [-h | -help] [-long_help] -device_name` *<device name>* `-hardware_name` *<hardware name>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-device_name` *<device name>*		Name of the device programmed with the design that includes In-System Sources and Probes instances
	`-hardware_name` *<hardware name>*		Name of the hardware that connects to the JTAG chain
Description	`Returns a list of the available In-System Sources and Probes instances and their configuration. {instance_index source_width probe_width instance_name}` `Example:` `{0 4 3 src1} {1 5 5 src2} {2 3 6 none}`		
Example Usage	`# List information of all In-System Sources and Probes instances` `puts "Information on all In-System Sources and Probes instances:"` `puts "index,source_width,probe_width,name"` `foreach instance [get_insystem_source_probe_instance_info -hardware_name "USB-Blaster \` `[USB-0\]" -device_name "@1: EP1S25/_HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"] {` `  puts "[lindex $instance 0],[lindex $instance 1],[lindex $instance 2],[lindex $instance 3]"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The specified hardware is not found.
	TCL_ERROR	1	ERROR: An internal Tcl interpreter error occurred.
	TCL_ERROR	1	ERROR: JTAG communication error detected. Errors can be caused by hardware failure or poor signal integrity in the JTAG chain.
	TCL_ERROR	1	ERROR: No In-System Sources and Probes instance was found.
	TCL_ERROR	1	ERROR: There is already an active In-System Sources and Probes session started. Unable to start another session.
	TCL_ERROR	1	ERROR: The device is locked by another application.

3.1.16.3. read_probe_data (::quartus::insystem_source_probe)

The following table displays information for the `read_probe_data` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_source_probe on page 286		
Syntax	`read_probe_data [-h | -help] [-long_help] -instance_index <instance_index> [-value_in_hex]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-instance_index <instance_index>`	Index of the In-System Sources and Probes instance to communicate with	
	`-value_in_hex`	Specifies that the value string is represented in hexadecimal format	
Description	`Retrieves the current value of the probes.` `A string is returned specifying the status of each probe, with the MSB on the left and LSB on the right. By default, the value is represented as a binary string. Optionally, the option -value_in_hex makes the value a hex string.`		
Example Usage	`#List probe data of instance 0` `start_insystem_sourc_probe -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `puts "probe data of instance 0"` `puts [read_probe_data -instance_index 0]` `end_insystem_source_probe`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The specified hardware is not found.
	TCL_ERROR	1	ERROR: An internal Tcl interpreter error occurred.
	TCL_ERROR	1	ERROR: The specified In-System Sources and Probes instance index is invalid.
	TCL_ERROR	1	ERROR: No In-System Sources and Probes instance was found.

3.1.16.4. read_source_data (::quartus::insystem_source_probe)

The following table displays information for the `read_source_data` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_source_probe on page 286	
Syntax	`read_source_data [-h | -help] [-long_help] -instance_index <instance_index> [-value_in_hex]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-instance_index <instance_index>`	Index of the In-System Sources and Probes instance to communicate with

continued...

	`-value_in_hex`	Specifies that the value string is represented in hexadecimal format
Description	`Retrieves the current value of the sources.` `A string is returned specifying the status of each source, with the MSB on the left and LSB on the right. By default, the value is represented as a binary string. Optionally, the option -value_in_hex makes the value a hex string.`	
Example Usage	`#List source data of instance 0` `start_insystem_source_probe -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `puts "source data of instance 0"` `puts [read_source_data -instance_index 0]` `end_insystem_source_probe`	

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The specified hardware is not found.
	TCL_ERROR	1	ERROR: An internal Tcl interpreter error occurred.
	TCL_ERROR	1	ERROR: The specified In-System Sources and Probes instance index is invalid.
	TCL_ERROR	1	ERROR: No In-System Sources and Probes instance was found.

3.1.16.5. start_insystem_source_probe (::quartus::insystem_source_probe)

The following table displays information for the `start_insystem_source_probe` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_source_probe on page 286	
Syntax	`start_insystem_source_probe [-h | -help] [-long_help] -device_name` *<device name>* `-hardware_name` *<hardware name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-device_name` *<device name>*	Name of the device that holds the In-System Sources and Probes instances
	`-hardware_name` *<hardware name>*	Name of programming hardware connected to the JTAG chain
Description	`Use this command before beginning any In-System Sources and Probes transactions`	
Example Usage	`#List probe data of instance 0` `start_insystem_source_probe -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `puts "probe data of instance 0"` `puts [read_probe_data -instance_index 0]` `end_insystem_source_probe`	

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The specified hardware is not found.
	TCL_ERROR	1	ERROR: An internal Tcl interpreter error occurred.
	TCL_ERROR	1	ERROR: JTAG communication error detected. Errors can be caused by hardware failure or poor signal integrity in the JTAG chain.
	TCL_ERROR	1	ERROR: No In-System Sources and Probes instance was found.
	TCL_ERROR	1	ERROR: There is already an active In-System Sources and Probes session started. Unable to start another session.
	TCL_ERROR	1	ERROR: The device is locked by another application.

3.1.16.6. write_source_data (::quartus::insystem_source_probe)

The following table displays information for the `write_source_data` Tcl command:

Tcl Package and Version	Belongs to ::quartus::insystem_source_probe on page 286		
Syntax	`write_source_data [-h | -help] [-long_help] -instance_index <instance_index> -value <value> [-value_in_hex]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-instance_index <instance_index>`		Index of the In-System Sources and Probes instance to communicate with
	`-value <value>`		Value for the source
	`-value_in_hex`		Specify that the value string is represented in hexadecimal format
Description	`Sets values for the sources.` `A value string is sent to the source values. MSB is on the left and LSB is on the right. The value string is is truncated on the left (MSB) side if necessary.` `By default, the values are represented as a binary string. Optionally, the option -value_in_hex makes the values hex strings.`		
Example Usage	`#List probe data of instance 0` `start_insystem_source_probe -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1S25/ _HARDCOPY_FPGA_PROTOTYPE (0x020030DD)"` `puts "write source data 10010"` `write_source_data -instance_index 0 -value "10010"` `puts "source data of instance 0"` `puts [read_source_data -instance_index 0]` `end_insystem_source_probe`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The specified hardware is not found.

continued...

	TCL_ERROR	1	ERROR: An internal Tcl interpreter error occurred.
	TCL_ERROR	1	ERROR: The specified In-System Sources and Probes instance index is invalid.
	TCL_ERROR	1	ERROR: No In-System Sources and Probes instance was found.

3.1.17. ::quartus::interactive_synthesis

The following table displays information for the **::quartus::interactive_synthesis** Tcl package:

Tcl Package and Version	::quartus::interactive_synthesis 1.0
Description	This package contains no general description.
Availability	This package is loaded by default in the following executables: quartus quartus_syn
Tcl Commands	analyze_files (::quartus::interactive_synthesis) on page 291 check_rtl_connections (::quartus::interactive_synthesis) on page 292 dissolve_rtl_partition (::quartus::interactive_synthesis) on page 292 dynamic_report (::quartus::interactive_synthesis) on page 293 elaborate (::quartus::interactive_synthesis) on page 294 get_entities (::quartus::interactive_synthesis) on page 295 get_rtl_partition_name (::quartus::interactive_synthesis) on page 295 get_rtl_partitions (::quartus::interactive_synthesis) on page 296 init_synthesis_constraints_propagation_reporter (::quartus::interactive_synthesis) on page 296 link_rtl_design (::quartus::interactive_synthesis) on page 296 print_ipxact (::quartus::interactive_synthesis) on page 297 report_rtl_assignments (::quartus::interactive_synthesis) on page 297 report_rtl_parameters (::quartus::interactive_synthesis) on page 298 report_rtl_stats (::quartus::interactive_synthesis) on page 298 reset_rtl_design (::quartus::interactive_synthesis) on page 299 sasic (::quartus::interactive_synthesis) on page 299 save_rtl_design (::quartus::interactive_synthesis) on page 300 set_sasic_handoff_flow (::quartus::interactive_synthesis) on page 300 synthesize (::quartus::interactive_synthesis) on page 301 uniquify (::quartus::interactive_synthesis) on page 301 write_rtl_report (::quartus::interactive_synthesis) on page 302

3.1.17.1. analyze_files (::quartus::interactive_synthesis)

The following table displays information for the `analyze_files` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291	
Syntax	`analyze_files [-h | -help] [-long_help] [-files <files_value> ] [-library <libray_value> ] [-lint]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-files <files_value>`	Specifies the list of files to analyze
	`-library <libray_value>`	Specifies the target library for design units
	`-lint`	Enables lint
Description	This command currently contains no help description.	

continued...

Example Usage	`# Compile foo.sv into the default library` `analyze_files -files foo.sv`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.2. check_rtl_connections (::quartus::interactive_synthesis)

The following table displays information for the `check_rtl_connections` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`check_rtl_connections [-h | -help] [-long_help] [-panel_name` *<panel_name_value>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name` *<panel_name_value>*		Specifies the name of the Check Connections report panel
Description	`This command currently contains no help description.`		
Example Usage	`# Report RTL connection issues to "Check RTL Connections"` `check_rtl_connections -panel_name "Check RTL Connections"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.17.3. dissolve_rtl_partition (::quartus::interactive_synthesis)

The following table displays information for the `dissolve_rtl_partition` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`dissolve_rtl_partition [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command
	TCL_ERROR	1	ERROR: Partition *<string>* not found

3.1.17.4. dynamic_report (::quartus::interactive_synthesis)

The following table displays information for the `dynamic_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291	
Syntax	`dynamic_report [-h | -help] [-long_help] [-dump_all] [-dump_lines` *<dumplines_value>* `] [-filename` *<filename_value>* `] [-html] [-query` *<query_value>* `] -report` *<report_value>* `[-summary] [-xml]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-dump_all`	dump the entire target report
	`-dump_lines` *<dumplines_value>*	dump first # of lines of the target report
	`-filename` *<filename_value>*	Name of output file to be generated
	`-html`	Option to generate output file in HTML format
	`-query` *<query_value>*	Specifies the SQLite query
	`-report` *<report_value>*	Specifies a target report to begin action with
	`-summary`	give a summary of the target report
	`-xml`	Option to generate output file in XML format
Description		

```
API that allows user to act with a full report which is limited on showing items in GUI.
The report is a dynamic reporting infrastructure from SQLite report database, independent of
synthesis compilation.
Use "-report" to target a report. This option is mandatory as further action depends on a
certain report.
Current available "report" values are:
         Value              Report name in compilation                      Database stored
    -----------------   ----------------------------------
-------------------------------------------
    "registers removed"    Registers Removed During Synthesis     ./dynamic_report/
registers_removed.sqlite
Use "-summary" to get a summary of the target report such as table name, column name that can be
used in query.
The format of the summary would be shown as:
    ================================================================
    Database name: <name of the database that the target report is saved>
    Table name: <table in the database that target report content locates>
    Total size: <total size of the report>
    Column: <1st column name of the report>
    Column: <2nd column name of the report>
    ...
    ================================================================
Use "-query" to take a whole SQLite query as an argument.
For example -query "SELECT * FROM <table> WHERE <column> = <...>" to get SELECT result from the
database.
If query command is incorrect, an error message with details will be issued.
Use "-dump_all" to dump all content in the report.
Use "-dump_lines" with a non-negative integer to dump first <non-negative integer> of lines in
the report.
The result of "-query", "-dump_all" and "-dump_lines" will be sent to an output file stored in
local design directory.
Use "-filename" to specify output filename. If no filename entered, output file will be named as
report value.
If the "-html" option is specified, the output file is generated in HTML format.
If the "-xml" option is specified, the output file is generated in XML format.
Otherwise, the output file is generated in ASCII format.
"-html" option and "-xml" option are mutually exclusive. Please only specify one option.
```

Example Usage

```
dynamic_report -report <report name> -query <SQLite query> -filename <output file name> -xml
dynamic_report -report <report name> -dump_all -filename <output file name>
dynamic_report -report <report name> -dump_lines <number of lines> -filename <output file
name> -html
dynamic_report -report <report name> -summary
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: No specified filename provided. Use <*string*> as the default output file name
	TCL_OK	0	INFO: Successfully generated output file: <*string*>. <*string*> lines reported.
	TCL_OK	0	INFO: The report's database summary is shown as the following:\n<*string*>
	TCL_OK	0	WARNING: Summary content directly returns from terminal, further options(filename, html, and xml) are ignored.
	TCL_ERROR	1	ERROR: Can't create or overwrite file: <*string*>.
	TCL_ERROR	1	ERROR: Failed to open the SQLite database. Make sure you are running tcl command in the top level design directory not dynamic_report directory.\n If the SQLite database is not generated during compilation, please turn ON the qsf: synth_rpt_enable_dynamic_report and rerun synthesis compilation
	TCL_ERROR	1	ERROR: Invalid filename input. Make sure your input is not empty and does not contain any special characters or spaces.
	TCL_ERROR	1	ERROR: Illegal value: <*string*>. Specify a non-negative integer ranging from 1 to 999999999 for the option -dump_lines.
	TCL_ERROR	1	ERROR: The referred sqlite database does not exist. There's no Registers Removed During Synthesis report table generated in your design.
	TCL_ERROR	1	ERROR: This report name: <*string*> is currently not supported for dynamic reporting.\nPlease use dynamic_report -help to see current supported report.
	TCL_ERROR	1	ERROR: SQLite error: <*string*>

3.1.17.5. elaborate (::quartus::interactive_synthesis)

The following table displays information for the `elaborate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291	
Syntax	`elaborate [-h | -help] [-long_help] [-lint] [-recompile]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-lint`	Enables linting
	`-recompile`	Enables recompile
Description	`This command currently contains no help description.`	
Example Usage	`# Elaborate from the top-level hierarchy` `elaborate`	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.6. get_entities (::quartus::interactive_synthesis)

The following table displays information for the `get_entities` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`get_entities [-h | -help] [-long_help] [-library` *<library_value>* `] [-name` *<name_value>* `]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-library` *<library_value>*	Specifies the library filter	
	`-name` *<name_value>*	Specifies the name filter	
Description	`This command currently contains no help description.`		
Example Usage	`# Get all "cpu" entities defined in all libraries` `set cpus [get_entities -entity cpu]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.7. get_rtl_partition_name (::quartus::interactive_synthesis)

The following table displays information for the `get_rtl_partition_name` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`get_rtl_partition_name [-h | -help] [-long_help] [-name` *<name_value>* `]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name` *<name_value>*	Specifies the name filter	
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.8. get_rtl_partitions (::quartus::interactive_synthesis)

The following table displays information for the `get_rtl_partitions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`get_rtl_partitions [-h | -help] [-long_help] [-name <name_value> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <name_value>`	Specifies the name filter	
Description	`This command currently contains no help description.`		
Example Usage	`# Get partitions matching filter pr_region*` `set pr_regions [get_rtl_partitions -name pr_region*]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.9. init_synthesis_constraints_propagation_reporter (::quartus::interactive_synthesis)

The following table displays information for the `init_synthesis_constraints_propagation_reporter` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`init_synthesis_constraints_propagation_reporter [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`This command currently contains no help description.`		
Example Usage	`# Initialize constraints propagation reporter for synthesis` `init_synthesis_constraints_propagation_reporter`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.10. link_rtl_design (::quartus::interactive_synthesis)

The following table displays information for the `link_rtl_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291
Syntax	`link_rtl_design [-h | -help] [-long_help] [-snapshot <snapshot_value> ]`

continued...

Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-snapshot` *<snapshot_value>*		Specifies input snapshot
Description	`This command currently contains no help description.`		
Example Usage	`# Link the RTL design for the current revision` `link_rtl_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.11. print_ipxact (::quartus::interactive_synthesis)

The following table displays information for the `print_ipxact` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`print_ipxact [-h | -help] [-long_help] -print_ipxact_file` *<file_name>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-print_ipxact_file` *<file_name>*		Specifies the source file to print
Description	`This command currently contains no help description.`		
Example Usage	`# Generate the IP-XACT for foo.v` `print_ipxact -print_ipxact_file foo.v`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.12. report_rtl_assignments (::quartus::interactive_synthesis)

The following table displays information for the `report_rtl_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`report_rtl_assignments [-h | -help] [-long_help] [-instance` *<instance_name>* `] [-panel_name` *<panel_name_value>* `] [-partition` *<partition_value>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-instance` *<instance_name>*		Specifies the hierarchy path of the instance to report (supports wildcards)

continued...

	`-panel_name` *<panel_name_value>*		Specifies the name of the report panel
	`-partition` *<partition_value>*		Specifies the hierarchy path of the partition(s) to report (supports wildcards)
Description	`This command currently contains no help description.`		
Example Usage	`report_rtl_assignments -panel_name "Source Assignments for Top partition" -partition |`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.17.13. report_rtl_parameters (::quartus::interactive_synthesis)

The following table displays information for the `report_rtl_parameters` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`report_rtl_parameters [-h | -help] [-long_help] [-instance` *<instance_name>* `] [-panel_name` *<panel_name_value>* `] [-partition` *<partition_value>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-instance` *<instance_name>*		Specifies the hierarchy path of the instance to report (supports wildcards)
	`-panel_name` *<panel_name_value>*		Specifies the name of the report panel
	`-partition` *<partition_value>*		Specifies the hierarchy path of the partition(s) to report (supports wildcards)
Description	`This command currently contains no help description.`		
Example Usage	`report_rtl_parameters -panel_name "RTL Parameters for Top partition" -partition |`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.17.14. report_rtl_stats (::quartus::interactive_synthesis)

The following table displays information for the `report_rtl_stats` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291	
Syntax	`report_rtl_stats [-h | -help] [-long_help] [-filename` *<filename_value>* `] -partition` *<partition_value>* `[-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-filename` *<filename_value>*	Specifies the output filename for the RTL report
	`-partition` *<partition_value>*	Specifies the hierarchy path of the partition to synthesize

continued...

	-stdout		Report to standard output
Description	This command currently contains no help description.		
Example Usage	This command currently contains no example usage.		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.17.15. reset_rtl_design (::quartus::interactive_synthesis)

The following table displays information for the reset_rtl_design Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	reset_rtl_design [-h | -help] [-long_help]		
Arguments	-h | -help		Short help
	-long_help		Long help with examples and possible return values
Description	This command currently contains no help description.		
Example Usage	# Resets the current RTL design. This must be # called prior to calling link_rtl_design again reset_rtl_design		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.16. sasic (::quartus::interactive_synthesis)

The following table displays information for the sasic Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	sasic [-h | -help] [-long_help] [-handoff]		
Arguments	-h | -help		Short help
	-long_help		Long help with examples and possible return values
	-handoff		Runs SASIC handoff flow
Description	This command currently contains no help description.		
Example Usage	# Run SASIC handoff sasic -handoff		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.17. save_rtl_design (::quartus::interactive_synthesis)

The following table displays information for the `save_rtl_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`save_rtl_design [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`This command currently contains no help description.`		
Example Usage	`# Saves the current RTL design for partition Top` `save_rtl_design -partition |` `# Saves all partitions in the current RTL design at a particular snapshot` `# Currently only allow partitioned and synthesized snapshot` `save_rtl_design -snapshot synthesized`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Snapshot <*string*> is invalid for save
	TCL_ERROR	1	ERROR: You must open a project before you can use this command
	TCL_ERROR	1	ERROR: Partition <*string*> not found

3.1.17.18. set_sasic_handoff_flow (::quartus::interactive_synthesis)

The following table displays information for the `set_sasic_handoff_flow` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`set_sasic_handoff_flow [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`This command currently contains no help description.`		
Example Usage	`# Set SASIC handoff flow` `set_sasic_handoff_flow`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.17.19. synthesize (::quartus::interactive_synthesis)

The following table displays information for the `synthesize` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`synthesize [-h | -help] [-long_help] -partition <partition_value> [-recompile]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-partition <partition_value>`		Specifies the hierarchy path of the partition to synthesize
	`-recompile`		Enables incremental resynthesis algorithm
Description	`This command currently contains no help description.`		
Example Usage	`# Run synthesis on Top partition` `synthesize -partition |` `# Run a re-synthesis on Top partition` `synthesize -partition | -recompile` `# Run a synthesis on ocs partition` `synthesize -partition | -techmap` `# Run high-level synthesis on Top partition` `synthesize -partition | -flow hls` `# Run post-high-level synthesis on Top partition` `synthesize -partition | -flow post_hls` `# This is the default and is equivalent to "synthesize -partition |"` `synthesize -partition | -flow full`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command
	TCL_ERROR	1	ERROR: Partition <*string*> not found

3.1.17.20. uniquify (::quartus::interactive_synthesis)

The following table displays information for the `uniquify` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`uniquify [-h | -help] [-long_help] [-analysis_and_elab_report] [-recompile]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-analysis_and_elab_report`		Print synthesis reports after analysis and elaboration
	`-recompile`		Enables recompile
Description	`This command currently contains no help description.`		
Example Usage	`# Run uniquify on all partitions as required` `uniquify` `# Run uniquify on one specific partition` `uniquify -partition cpu_left`		
Return Value	**Code Name**	**Code**	**String Return**

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command
	TCL_ERROR	1	ERROR: Current design not found
	TCL_ERROR	1	ERROR: Partition <*string*> not found

3.1.17.21. write_rtl_report (::quartus::interactive_synthesis)

The following table displays information for the `write_rtl_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::interactive_synthesis on page 291		
Syntax	`write_rtl_report [-h | -help] [-long_help] [-ascii] [-filename` <*filename_value*> `] [-qdb] [-xml]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-ascii`	Spcefies ASCII as the output format	
	`-filename` <*filename_value*>	Specifies the output filename for the RTL report	
	`-qdb`	Specifies binary as the output format	
	`-xml`	Specifies XML as the output format	
Description	`This command currently contains no help description.`		
Example Usage	`# Write the RTL report to the QDB database in binary forat` `write_rtl_report -qdb` `# Write the RTL report to the default ASCII report file` `write_rtl_report -ascii` `# Write the RTL report to the XML file` `write_rtl_report -xml -filename report.xml`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command

3.1.18. ::quartus::ipdrc

The following table displays information for the **::quartus::ipdrc** Tcl package:

Tcl Package and Version	::quartus::ipdrc 1.0
Description	`This package contains no general description.`
Availability	`This package is available for loading in the following executables:` `quartus_cdb` `quartus_syn`

continued...

Tcl Commands	ipdrc::get_device_speed (::quartus::ipdrc) on page 303 ipdrc::get_ip_hpaths (::quartus::ipdrc) on page 303 ipdrc::get_ip_name (::quartus::ipdrc) on page 304 ipdrc::get_ip_pma_modulation (::quartus::ipdrc) on page 304 ipdrc::get_ip_speed (::quartus::ipdrc) on page 305 ipdrc::get_ip_type (::quartus::ipdrc) on page 305 ipdrc::get_ip_xcvr_type (::quartus::ipdrc) on page 305 ipdrc::set_ip_info (::quartus::ipdrc) on page 306

3.1.18.1. ipdrc::get_device_speed (::quartus::ipdrc)

The following table displays information for the `ipdrc::get_device_speed` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipdrc on page 302		
Syntax	`ipdrc::get_device_speed [-h | -help] [-long_help] -core <core> -hpath <hpath> -xcvr <xcvr>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-core <core>`		Core speed grade of the target device
	`-hpath <hpath>`		Hierarchical path of an instance
	`-xcvr <xcvr>`		Transceiver speed grade of the target device
Description	`IPDRC get device speed --This command returns the max speed of a device given core grade (1~4) and transceiver grade (1~3)`		
Example Usage	`ipdrc::get_device_speed -hpath $hpath -core $core -xcvr $xcvr`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.18.2. ipdrc::get_ip_hpaths (::quartus::ipdrc)

The following table displays information for the `ipdrc::get_ip_hpaths` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipdrc on page 302	
Syntax	`ipdrc::get_ip_hpaths [-h | -help] [-long_help] [-filter <filter> ] [-ip_name <ip_name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-filter <filter>`	Filter searches against parameters stored in the JSON file. Format: {param_id value} 2 strings connected by 1 space
	`-ip_name <ip_name>`	IP name to search for
Description	`IPDRC get IP hpaths --This command returns a list of IP hpaths based on prescribed filterings; by default returns every existing hpath`	

continued...

Example Usage	`ipdrc::get_ip_hpaths -ip_name $ip_name -filter $filter`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.18.3. ipdrc::get_ip_name (::quartus::ipdrc)

The following table displays information for the `ipdrc::get_ip_name` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipdrc on page 302		
Syntax	`ipdrc::get_ip_name [-h | -help] [-long_help] -type <type>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-type <type>`		Identifier representing an IP type
Description	`IPDRC get IP name --This command returns the actual IP name corresponding to a given IP type`		
Example Usage	`ipdrc::get_ip_name -type $type`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.18.4. ipdrc::get_ip_pma_modulation (::quartus::ipdrc)

The following table displays information for the `ipdrc::get_ip_pma_modulation` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipdrc on page 302		
Syntax	`ipdrc::get_ip_pma_modulation [-h | -help] [-long_help] -hpath <hpath>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-hpath <hpath>`		Hierarchical path of an instance
Description	`IPDRC get IP PMA modulation --This command returns the PMA modulation mode of an IP instance`		
Example Usage	`ipdrc::get_ip_pma_modulation -hpath $hpath`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.18.5. ipdrc::get_ip_speed (::quartus::ipdrc)

The following table displays information for the `ipdrc::get_ip_speed` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipdrc on page 302		
Syntax	`ipdrc::get_ip_speed [-h | -help] [-long_help] -hpath <hpath>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-hpath <hpath>`		Hierarchical path of an instance
Description	`IPDRC get IP speed --This command returns the transceiver speed of an IP instance`		
Example Usage	`ipdrc::get_ip_speed -hpath $hpath`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.18.6. ipdrc::get_ip_type (::quartus::ipdrc)

The following table displays information for the `ipdrc::get_ip_type` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipdrc on page 302		
Syntax	`ipdrc::get_ip_type [-h | -help] [-long_help] -hpath <hpath>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-hpath <hpath>`		Hierarchical path of an instance
Description	`IPDRC get IP type --This command returns the IP type of an instance`		
Example Usage	`ipdrc::get_ip_type -hpath $hpath`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.18.7. ipdrc::get_ip_xcvr_type (::quartus::ipdrc)

The following table displays information for the `ipdrc::get_ip_xcvr_type` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipdrc on page 302	
Syntax	`ipdrc::get_ip_xcvr_type [-h | -help] [-long_help] -hpath <hpath>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-hpath <hpath>`		Hierarchical path of an instance
Description	`IPDRC get IP XCVR type --This command returns the XCVR type of an IP instance`		
Example Usage	`ipdrc::get_ip_xcvr_type -hpath $hpath`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.18.8. ipdrc::set_ip_info (::quartus::ipdrc)

The following table displays information for the `ipdrc::set_ip_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipdrc on page 302		
Syntax	`ipdrc::set_ip_info [-h | -help] [-long_help] -file <file>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-file <file>`		JSON file containing instances information of the design
Description	`IPDRC set IP info --This command initializes the content of the static IPDRC_CHECK_UTILS object with the given JSON file`		
Example Usage	`ipdrc::set_ip_info -file $file`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.19. ::quartus::ipgen

The following table displays information for the **::quartus::ipgen** Tcl package:

Tcl Package and Version	::quartus::ipgen 1.0
Description	`This package contains no general description.`
Availability	`This package is loaded by default in the following executable:` `quartus_ipgenerate`
Tcl Commands	clear_ip_generation_dirs (::quartus::ipgen) on page 307 generate_ip_file (::quartus::ipgen) on page 307 generate_project_ip_files (::quartus::ipgen) on page 308 get_project_ip_files (::quartus::ipgen) on page 310

3.1.19.1. clear_ip_generation_dirs (::quartus::ipgen)

The following table displays information for the `clear_ip_generation_dirs` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipgen on page 306		
Syntax	`clear_ip_generation_dirs [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`This command removes all the generation directories for the Platform Designer IP files in the opened project.` `All the content in the generation directories will be removed.`		
Example Usage	`# Remove all the generation directories for the Platform Designer IP files in the opened project` `project_open my_project` `clear_ip_generation_dirs`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: The command failed with an unknown error.

3.1.19.2. generate_ip_file (::quartus::ipgen)

The following table displays information for the `generate_ip_file` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipgen on page 306	
Syntax	`generate_ip_file [-h | -help] [-long_help] [-clean] [-clear_ip_generation_dirs] [-modelsim_flow` *<qrun|traditional>* `] [-simulation` *<verilog|vhdl>* `] [-simulator` *<modelsim|vcs|vcsmx|riviera|xcelium>* `] [-synthesis` *<verilog|vhdl>* `] <file>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clean`	Specify whether pre-existing generation directories should be cleared before generation.
	`-clear_ip_generation_dirs`	Specify whether pre-existing generation directories should be cleared before generation.
	`-modelsim_flow` *<qrun|traditional>*	Specifies the Modelsim flow used for simulation script generation. Valid values are qrun or traditional.
	`-simulation` *<verilog|vhdl>*	Set the simulation target type. Valid values are verilog or vhdl.
	`-simulator` *<modelsim|vcs|vcsmx|riviera|xcelium>*	Set the simulator target type. Valid values are modelsim, vcs, vcsmx, riviera, and/or xcelium.
	`-synthesis` *<verilog|vhdl>*	Set the synthesis target type. Valid values are verilog or vhdl.
	<file>	A Platform Designer IP file path. -file="path1;path2"

continued...

Description	This command generates the files for a specified Platform Designer IP in the opened project. --synthesis <value>: Specify the synthesis target type. Valid values are verilog or vhdl. This is not a required option. When not specified, it defaults to verilog. --simulation <value>: Specify the simulation target type. Valid values are verilog or vhdl. This is not a required option. When not specified, no simulation files are generated. --simulator <value>: Specify the simulator target type. Valid values are modelsim, vcs, vcsmx, riviera, xcelium. This is not a required option. When not specified, simulation files for all simulators are generated. --modelsim_flow <value>: Specify the Modelsim flow used for simulation script generation. Valid values are qrun or traditional. This is not a required option. When not specified, it defaults to qrun. --clear_ip_generation_dirs: Specify whether pre-existing generation directories should be cleared before generation. This is not a required option. When not specified, the generation directories will not be cleared. --clean: Specify whether pre-existing generation directories should be cleared before generation. This option is a short version of the clear_ip_generation_dirs option. This is not a required option. When not specified, the generation directories will not be cleared.
Example Usage	`# generate the specified Platform Designer IP in the project with the specified targets. Clear any pre-existing # generation directories before performing the generation.` `project_open my_project` `generate_ip_file my_ip_file.qsys -synthesis verilog -simulation verilog -simulator modelsim -clear_ip_generation_dirs`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The file <*string*> does not exist in project.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: The command failed with an unknown error.

3.1.19.3. generate_project_ip_files (::quartus::ipgen)

The following table displays information for the `generate_project_ip_files` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipgen on page 306	
Syntax	`generate_project_ip_files [-h | -help] [-long_help] [-clean] [-clear_ip_generation_dirs] [-modelsim_flow <qrun|traditional> ] [-parallel <on|off> ] [-simulation <verilog|vhdl> ] [-simulator <modelsim|vcs|vcsmx|riviera|xcelium> ] [-synthesis <verilog|vhdl> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clean`	Specify whether pre-existing generation directories should be cleared before generation.

continued...

	`-clear_ip_generation_dirs`	Specify whether pre-existing generation directories should be cleared before generation.
	`-modelsim_flow <qrun|traditional>`	Specifies the Modelsim flow used for simulation script generation. Valid values are qrun or traditional.
	`-parallel <on|off>`	Specify whether to allow parallel IP generation. Specify off to disable parallel
	`-simulation <verilog|vhdl>`	Set the simulation target type. Valid values are verilog or vhdl.
	`-simulator <modelsim|vcs|vcsmx|riviera|xcelium>`	Set the simulator target type. Valid values are modelsim, vcs, vcsmx, riviera, and/or xcelium.
	`-synthesis <verilog|vhdl>`	Set the synthesis target type. Valid values are verilog or vhdl.

Description

```
This command generates the files for all Platform Designer IP in the opened project.
       If no option is specified, this command generates the verilog synthesis target only.

       --synthesis <value>:         Specify the synthesis target type. Valid values are verilog
or vhdl.
                                    This is not a required option. When not specified, it
defaults to verilog.

       --simulation <value>:        Specify the simulation target type. Valid values are
verilog or vhdl.
                                    This is not a required option. When not specified, no
simulation files are generated.

       --simulator <value>:        Specify the simulator target type. Valid values are
modelsim, vcs, vcsmx, riviera, and/or xcelium.
                                    This is not a required option. When not specified,
simulation files for all simulators are generated.

       --modelsim_flow <value>:     Specify the Modelsim flow used for simulation script
generation. Valid values are qrun or traditional.
                                    This is not a required option. When not specified, it
defaults to qrun.

       --clear_ip_generation_dirs:  Specify whether pre-existing generation directories should
be cleared before generation.
                                    This is not a required option. When not specified, the
generation directories will not be cleared.

       --clean:                    Specify whether pre-existing generation directories should
be cleared before generation.
                                    This option is a short version of the
clear_ip_generation_dirs option.
                                    This is not a required option. When not specified, the
generation directories will not be cleared.

       --parallel <on,off>:         Specify whether to allow parallel IP generation. When this
is on, parallel
                                     processing will be enable if project or global Quartus
setting is enabled.
                                     When this setting is off, parallel is disabled regardless
of project or global
                                     Quartus settings.
```

Example Usage

```
  # generate all the Platform Designer IP in the project with the specified targets. Clear any
pre-existing
         # generation directories before performing the generation.
project_open my_project
generate_project_ip_files -synthesis verilog -simulation verilog -simulator modelsim -
clear_ip_generation_dirs
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: The file <*string*> does not exist in project.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: The command failed with an unknown error.

3.1.19.4. get_project_ip_files (::quartus::ipgen)

The following table displays information for the `get_project_ip_files` Tcl command:

Tcl Package and Version	Belongs to ::quartus::ipgen on page 306		
Syntax	`get_project_ip_files [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`This command returns a Tcl list containing the full path of the Platform Designer IP files (.qsys or .ip) found in the opened project.`		
Example Usage	`project_open my_project` `get_project_ip_files`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: The command failed with an unknown error.

3.1.20. ::quartus::iptclgen

The following table displays information for the **::quartus::iptclgen** Tcl package:

Tcl Package and Version	::quartus::iptclgen 1.0
Description	`This package contains the set of Tcl functions for generating Memory IP.`
Availability	`This package is available for loading in the following executables:` `qpro` `qpro_sh` `quartus` `quartus_cdb` `quartus_sh`
Tcl Commands	compute_pll (::quartus::iptclgen) on page 310 generate_vhdl_simgen_model (::quartus::iptclgen) on page 311 parse_hdl (::quartus::iptclgen) on page 312 parse_tcl (::quartus::iptclgen) on page 313

3.1.20.1. compute_pll (::quartus::iptclgen)

The following table displays information for the `compute_pll` Tcl command:

Tcl Package and Version	Belongs to ::quartus::iptclgen on page 310
Syntax	`compute_pll [-h | -help] [-long_help] -family` *<family>* `-input_freq` *<input_freq>* `-output_freqs` *<output_freqs>* `-output_phases` *<output_phases>* `-pll_type` *<pll_type>* `-speed_grade` *<speed_grade>* `-temp_dir` *<temp_dir>*

continued...

Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-family` *<family>*	Device family
	`-input_freq` *<input_freq>*	input frequency in Mhz
	`-output_freqs` *<output_freqs>*	desired output frequencys
	`-output_phases` *<output_phases>*	desired output phase shifts
	`-pll_type` *<pll_type>*	pll type
	`-speed_grade` *<speed_grade>*	Device speed grade
	`-temp_dir` *<temp_dir>*	temporary directory
Description	`Parses the HDL file specified and saves output to the file name specified.`	
Example Usage	`compute_pll -family "stratix iii" -pll_type "fast_pll" -input_freq 100 -output_freqs "100, 200, 333"`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid Intel FPGA IP Name: *<string>*. Specify a legal port type.
	TCL_ERROR	1	ERROR: Illegal number of arguments. Specify %u argument(s) for option *<string>*.
	TCL_ERROR	1	ERROR: No open project. Open an existing project or create a new project.

3.1.20.2. generate_vhdl_simgen_model (::quartus::iptclgen)

The following table displays information for the `generate_vhdl_simgen_model` Tcl command:

Tcl Package and Version	Belongs to ::quartus::iptclgen on page 310	
Syntax	`generate_vhdl_simgen_model [-h | -help] [-long_help] -blackbox` *<blackbox>* `-family` *<family>* `-files` *<files>* `-result_dir` *<result_dir>* `-temp_dir` *<temp_dir>* `-top_level_name` *<top_level_name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-blackbox` *<blackbox>*	comma-separated list of modules that should be blackboxed
	`-family` *<family>*	family name
	`-files` *<files>*	comma seperated list of files
	`-result_dir` *<result_dir>*	result directory for .vho file. Must already exist
	`-temp_dir` *<temp_dir>*	temp_dir
	`-top_level_name` *<top_level_name>*	top level entity name

continued...

Description	Creates a temporary project in the temporary directory, creates the simgen model and copies the model to result_dir.		
Example Usage	`generate_vhdl_simgen_model -family "stratix iii" -files "mycore.v,subcore.v" -top_level_name "mycore.v" -temp_dir "c:/temp" -result_dir "c:/outdir" -blackbox "blackboxme"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No open project. Open an existing project or create a new project.

3.1.20.3. parse_hdl (::quartus::iptclgen)

The following table displays information for the `parse_hdl` Tcl command:

Tcl Package and Version	Belongs to ::quartus::iptclgen on page 310		
Syntax	`parse_hdl [-h | -help] [-long_help] -core_params` *<core_params>* `-indir_name` *<indir_name>* `-inhdl_files` *<inhdl_files>* `[-module_list` *<module_list>* `] -module_name` *<module_name>* `-outdir_name` *<outdir_name>* `-supported_params` *<supported_params>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-core_params` *<core_params>*		Comma-delimited list of core parameters
	`-indir_name` *<indir_name>*		input directory of hdl files
	`-inhdl_files` *<inhdl_files>*		input hdl name
	`-module_list` *<module_list>*		list of modules to uniquify name
	`-module_name` *<module_name>*		toplevel module name
	`-outdir_name` *<outdir_name>*		output directory of hdl files
	`-supported_params` *<supported_params>*		Comma-delimited list of supported core parameters
Description	Parses the HDL file specified and saves output to the file name specified.		
Example Usage	`parse_hdl -inhdl_files rldram_dev.v -core_params "USE_CLK, STRATIXIII" -outdir_name "/project/hdl"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid Intel FPGA IP Name: *<string>*. Specify a legal port type.
	TCL_ERROR	1	ERROR: Illegal number of arguments. Specify %u argument(s) for option *<string>*.
	TCL_ERROR	1	ERROR: No open project. Open an existing project or create a new project.

3.1.20.4. parse_tcl (::quartus::iptclgen)

The following table displays information for the `parse_tcl` Tcl command:

Tcl Package and Version	Belongs to ::quartus::iptclgen on page 310		
Syntax	`parse_tcl [-h | -help] [-long_help] [-core_params` *<core_params>* `] [-core_sub_params` *<core_sub_params>* `] -indir_name` *<indir_name>* `-intcl_files` *<intcl_files>* `-module_name` *<module_name>* `-outdir_name` *<outdir_name>*		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-core_params` *<core_params>*	Comma-delimited list of core parameters	
	`-core_sub_params` *<core_sub_params>*	map of parameter-to-value pairs to replace	
	`-indir_name` *<indir_name>*	input directory of tcl files	
	`-intcl_files` *<intcl_files>*	input tcl name	
	`-module_name` *<module_name>*	toplevel module name	
	`-outdir_name` *<outdir_name>*	output directory of tcl files	
Description	`Parses the HDL file specified and saves output to the file name specified.`		
Example Usage	`parse_tcl -intcl_files rldram_dev.v -core_params "USE_CLK, STRATIXIII" -outdir_name "/project/tcl"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid Intel FPGA IP Name: *<string>*. Specify a legal port type.
	TCL_ERROR	1	ERROR: Illegal number of arguments. Specify %u argument(s) for option *<string>*.
	TCL_ERROR	1	ERROR: No open project. Open an existing project or create a new project.

3.1.21. ::quartus::jtag

The following table displays information for the **::quartus::jtag** Tcl package:

Tcl Package and Version	::quartus::jtag 1.0
Description	`This package contains the set of Tcl functions for controlling the JTAG chain using Intel programming hardware.`
Availability	`This package is loaded by default in the following executables:` `quartus_stp` `quartus_stp_tcl`
Tcl Commands	close_device (::quartus::jtag) on page 314 device_dr_shift (::quartus::jtag) on page 315 device_ir_shift (::quartus::jtag) on page 316 device_lock (::quartus::jtag) on page 317

continued...

	device_run_test_idle (::quartus::jtag) on page 318 device_unlock (::quartus::jtag) on page 319 device_virtual_dr_shift (::quartus::jtag) on page 320 device_virtual_ir_shift (::quartus::jtag) on page 322 get_device_names (::quartus::jtag) on page 323 get_hardware_names (::quartus::jtag) on page 324 open_device (::quartus::jtag) on page 325

3.1.21.1. close_device (::quartus::jtag)

The following table displays information for the `close_device` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313		
Syntax	`close_device [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`End the shared communication with the opened device.`		
Example Usage	`# List all available programming hardwares, and select the USBBlaster.` `# (Note: this example assumes only one USBBlaster connected.)` `puts "Programming Hardwares:"` `foreach hardware_name [get_hardware_names] {` `  puts $hardware_name` `  if { [string match "USB-Blaster*" $hardware_name] } {` `  set usbblaster_name $hardware_name` `  }` `}` `puts "\nSelect JTAG chain connected to $usbblaster_name.\n";` `# List all devices on the chain, and select the first device on the chain.` `puts "\nDevices on the JTAG chain:"` `foreach device_name [get_device_names -hardware_name $usbblaster_name] {` `  puts $device_name` `  if { [string match "@1*" $device_name] } {` `  set test_device $device_name` `  }` `}` `puts "\nSelect device: $test_device.\n";` `# Open device` `open_device -hardware_name $usbblaster_name -device_name $test_device` `# Retrieve device id code.` `# IDCODE instruction value is 6; The ID code is 32 bits long.` `# IR and DR shift should be locked together to ensure that other applications` `# will not change the instruction register before the id code value is shifted` `# out while the instruction register is still holding the IDCODE instruction.` `device_lock -timeout 10000` `device_ir_shift -ir_value 6 -no_captured_ir_value` `puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]"` `device_unlock` `# Close device` `close_device`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No device has been opened.
	TCL_ERROR	1	ERROR: A device was locked.

3.1.21.2. device_dr_shift (::quartus::jtag)

The following table displays information for the `device_dr_shift` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`device_dr_shift [-h | -help] [-long_help] [-dr_value` *<data register value>* `] -length` *<data register length>* `[-no_captured_dr_value] [-value_in_hex]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-dr_value` *<data register value>*	Value of string operand type in either default binary or hexadecimal format to be written into the data register in the JTAG tap controller of the open device
	`-length` *<data register length>*	Length of the data register in the JTAG tap controller in the open device
	`-no_captured_dr_value`	Option not to return the data instruction register value. If this is specified, this DR scan may be packed together with the subsequent IR or DR scan until the device is unlocked or a captured value is requested
	`-value_in_hex`	Option to specify that the value string is represented in hexadecimal format
Description		

```
Writes the specified value into the data register of
the JTAG tap controller of the open device. Returns
the captured data register value. The captured value
return can be disabled to improve the JTAG
communication speed by packing multiple IR or DR scans
together.

The value is specified using either a binary string or
a hexadecimal string. The bit on the left most side is
the first bit shifted in.  For example, using binary
string "010001", the first bit shifted into the dr
register is 1; the last bit is 0.  The same string can
be represented in hexadecimal as "11".

The device must be locked before you can perform this
operation.
```

Example Usage

```
# List all available programming hardware, and select the USB-Blaster.
# (Note: this example assumes only one USB-Blaster is connected.)
puts "Programming Hardware:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# Retrieve device id code.
# IDCODE instruction value is 6; The ID code is 32 bits long.

# IR and DR shift should be locked together to ensure that other applications
# will not change the instruction register before the id code value is shifted
# out while the instruction register is still holding the IDCODE instruction.
device_lock -timeout 10000
```

continued...

```
device_ir_shift -ir_value 6 -no_captured_ir_value
puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]"
device_unlock

# Close device
close_device
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Captured value cannot be disabled at the time when no value is shifted into data register.
	TCL_ERROR	1	ERROR: A device has not been locked; exclusive communication must be obtained first.
	TCL_ERROR	1	ERROR: A device has been locked by another application; exclusive communication cannot be granted within the specified timeout period.
	TCL_ERROR	1	ERROR: The length of the value string specified does not match the length parameter specified.

3.1.21.3. device_ir_shift (::quartus::jtag)

The following table displays information for the `device_ir_shift` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`device_ir_shift [-h | -help] [-long_help] -ir_value` *<instruction register value>* `[-no_captured_ir_value]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-ir_value` *<instruction register value>*	Value to be written into the instruction register in the JTAG tap controller of the open device. Operand must be of a TCL numerical type such as decimal (10), hexadecimal (0xa), or octal number (012)
	`-no_captured_ir_value`	Option to not return the captured instruction register value. If this is specified, this IR scan may be packed together with the subsequent IR or DR scan until the device is unlocked or a captured value is requested
Description		

```
       Writes the specified value into the instruction
register of the JTAG tap controller of the open
device. Returns the captured instruction register
value. The captured value return can be disabled to
improve the JTAG communication speed by packing
multiple IR or DR scans together.

The instruction register length is determined
automatically by the Quartus Prime JTAG server.

The device must be locked first before this operation.
```

Example Usage

```
# List all available programming hardware, and select the USB-Blaster.
# (Note: this example assumes only one USB-Blaster is connected.)
puts "Programming Hardware:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
```

continued...

```
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# Retrieve device id code.
# IDCODE instruction value is 6; The ID code is 32 bits long.

# IR and DR shift should be locked together to ensure that other applications
# will not change the instruction register before the id code value is shifted
# out while the instruction register is still holding the IDCODE instruction.
device_lock -timeout 10000
device_ir_shift -ir_value 6 -no_captured_ir_value
puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]"
device_unlock

# Close device
close_device
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: A device has not been locked; exclusive communication must be obtained first.
TCL_ERROR	1	ERROR: A device has been locked by another application; exclusive communication cannot be granted within the specified timeout period.

3.1.21.4. device_lock (::quartus::jtag)

The following table displays information for the `device_lock` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`device_lock [-h | -help] [-long_help] -timeout <timeout>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-timeout <timeout>`	The amount of time in millisecond to wait for the access to the device.

Description

```
Obtain an exclusive JTAG communication to the device
       for the subsequent IR and DR shift operations. The
       device must be locked before any instruction and/or
       data register shift operation.

This should be used as little time as possible as it
       denies the access of other applications to this
       chain. The command, unlock, should be called as soon
       as possible to allow other applications to access the
       device.
```

Example Usage

```
# List all available programming hardwares, and select the USBBlaster.
# (Note: this example assumes only one USBBlaster connected.)
puts "Programming Hardwares:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
```

continued...

```
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# Retrieve device id code.
# IDCODE instruction value is 6; The ID code is 32 bits long.

# IR and DR shift should be locked together to ensure that other applications
# will not change the instruction register before the id code value is shifted
# out while the instruction register is still holding the IDCODE instruction.
device_lock -timeout 10000
device_ir_shift -ir_value 6 -no_captured_ir_value
puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]"
device_unlock

# Close device
close_device
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: No device has been opened.
TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure and signal integrity in the JTAG chain. Try to restart.
TCL_ERROR	1	ERROR: A device was locked.
TCL_ERROR	1	ERROR: A device has been locked by another application; exclusive communication cannot be granted within the specified timeout period.

3.1.21.5. device_run_test_idle (::quartus::jtag)

The following table displays information for the `device_run_test_idle` Tcl command:

Tcl Package and Version

Belongs to ::quartus::jtag on page 313

Syntax

`device_run_test_idle [-h | -help] [-long_help] [-num_clocks <state cycle count> ]`

Arguments

`-h | -help`	Short help
`-long_help`	Long help with examples and possible return values
`-num_clocks <state cycle count>`	The number of times the Run_Test_Idle state is cycled through. If not specified, this value is 1

Description

```
Drive the JTAG controller into the Run_Test_Idle state
for a number cycles specified with the -num_clocks
option.

The device must be locked before you can perform this
operation.
```

Example Usage

```
# List all available programming hardware, and select the USB-Blaster.
# (Note: this example assumes only one USB-Blaster is connected.)
puts "Programming Hardware:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
```

continued...

```
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# Retrieve device id code.
# IDCODE instruction value is 6; The ID code is 32 bits long.

# IR and DR shift should be locked together to ensure that other applications
# will not change the instruction register before the id code value is shifted
# out while the instruction register is still holding the IDCODE instruction.
device_lock -timeout 10000
device_ir_shift -ir_value 6 -no_captured_ir_value
puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]"

# Goto the Run_Test_Idle state and stay there for 8 cycles.
device_run_test_idle -num_clocks 8
device_unlock

# Close device
close_device
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: A device has not been locked; exclusive communication must be obtained first.
TCL_ERROR	1	ERROR: A device has been locked by another application; exclusive communication cannot be granted within the specified timeout period.

3.1.21.6. device_unlock (::quartus::jtag)

The following table displays information for the `device_unlock` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`device_unlock [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Release the exclusive JTAG communication lock on the device.`	
Example Usage		

```
# List all available programming hardwares, and select the USBBlaster.
# (Note: this example assumes only one USBBlaster connected.)
puts "Programming Hardwares:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
```

continued...

```
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# Retrieve device id code.
# IDCODE instruction value is 6; The ID code is 32 bits long.

# IR and DR shift should be locked together to ensure that other applications
# will not change the instruction register before the id code value is shifted
# out while the instruction register is still holding the IDCODE instruction.
device_lock -timeout 10000
device_ir_shift -ir_value 6 -no_captured_ir_value
puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]"
device_unlock

# Close device
close_device
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No device has been opened.
	TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure and signal integrity in the JTAG chain. Try to restart.
	TCL_ERROR	1	ERROR: A device has not been locked; exclusive communication must be obtained first.

3.1.21.7. device_virtual_dr_shift (::quartus::jtag)

The following table displays information for the `device_virtual_dr_shift` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`device_virtual_dr_shift [-h | -help] [-long_help] [-dr_value` *<data register value>* `] -instance_index` *<instance index>* `-length` *<data register length>* `[-no_captured_dr_value] [-show_equivalent_device_ir_dr_shift] [-value_in_hex]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-dr_value` *<data register value>*	Value of string operand type in either default binary or hexadecimal format to be written into the data register in this instance
	`-instance_index` *<instance index>*	The index of the virtual JTAG instance
	`-length` *<data register length>*	Length of the data register in this instance
	`-no_captured_dr_value`	Option to not return the data instruction register value
	`-show_equivalent_device_ir_dr_shift`	Option to show equivalent device ir dr shifts performed by this command
	`-value_in_hex`	Option to specify that the value string is represented in hexadecimal format
Description		

```
       Writes the specified value into the data register of
the JTAG tap controller of the open device. Returns
the captured data register value. The captured value
return can be disabled to improve the JTAG
communication speed by packing multiple IR or DR scans
```

continued...

```
together.

The value is specified using either a binary string or
a hexadecimal string. The bit on the left most side is
the first bit shifted in.  For example, using the
binary string "010001", the first bit shifted into the
dr register is 1; the last bit is 0.  The same string
can be represented in hexadecimal as "11".

The device must be locked first, and the target
instance must be activated using the
device_virtual_ir_shift command before this
operation. Moreover, the device should be locked
before the virtual IR shift operation to prevent
another application from activating another instance.
```

Example Usage

```
# List all available programming hardware, and select the USB-Blaster.
# (Note: this example assumes only one USB-Blaster is connected.)
puts "Programming Hardware:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# The follow virtual JTAG IR and DR shift sequence engage with
# the example virtual JTAG instance.
#
# Two instructions: SAMPLE (1) FEED (2)
# SAMPLE instruction samples a 8-bit bus; the captured value shows the number of sample
performed.
# FEED instruction supplies a 8-bit value to the logic connected to this instance.
# Both data registers corresponding to the IR are 8 bit wide.

# Send SAMPLE instruction to IR, read captured IR for the sampling number.
# Capture the DR register for the current sampled value.
device_lock -timeout 10000
puts "Current LED Value (sample #[device_virtual_ir_shift -instance_index 0 -ir_value 1]):
[device_virtual_dr_shift -instance_index 0  -length 8 -value_in_hex]"
device_unlock

# Send FEED instruction to IR, read a two-digit hex string from the console,
# then send the new value to the DR register.
puts "\nType in 2 digits in hexadecimal to update the LED:"
gets stdin update_value

device_lock -timeout 10000
device_virtual_ir_shift -instance_index 0 -ir_value 2 -no_captured_ir_value
device_virtual_dr_shift -instance_index 0  -length 8 -dr_value $update_value -value_in_hex -
no_captured_dr_value
device_unlock

# Close device
close_device
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Captured value cannot be disabled at the time when no value is shifted into data register.
TCL_ERROR	1	ERROR: The specified virtual JTAG instance cannot be found.
TCL_ERROR	1	ERROR: One of the options, mfg_id, type_id and version is specified, but not all. All of them are required.

continued...

	TCL_ERROR	1	ERROR: A device has not been locked; exclusive communication must be obtained first.
	TCL_ERROR	1	ERROR: A device has been locked by another application; exclusive communication cannot be granted within the specified timeout period.
	TCL_ERROR	1	ERROR: The length of the value string specified does not match the length parameter specified.

3.1.21.8. device_virtual_ir_shift (::quartus::jtag)

The following table displays information for the `device_virtual_ir_shift` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`device_virtual_ir_shift [-h | -help] [-long_help] -instance_index` *<instance index>* `-ir_value` *<instruction register value>* `[-no_captured_ir_value] [-show_equivalent_device_ir_dr_shift]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-instance_index` *<instance index>*	The index of the virtual JTAG instance
	`-ir_value` *<instruction register value>*	Value to be written into the instruction register in this instance. Operand must be of a TCL numerical type such as decimal (10), hexadecimal (0xa), or octal number (012)
	`-no_captured_ir_value`	Option to not return the captured instruction register value. If this is specified, this IR scan may be packed together with the subsequent IR or DR scan until the device is unlocked or a captured value is requested
	`-show_equivalent_device_ir_dr_shift`	Option to show equivalent device ir and dr shifts performed by this command
Description		

```
       Writes the specified value into the instruction
register of the specified virtual JTAG instance in the
open device. Returns the captured instruction register
value. You can disable the captured value return to
improve the JTAG communication speed by packing
multiple IR or DR scans together.

The command also activates the target instance such
that the consequent virtual DR shift operations are
applied to this instance before the device is
unlocked. Before any virtual DR shift operation, this
command must be executed first to activate the
instance.

The device must be locked first before this operation.
```

Example Usage

```
# List all available programming hardwares, and select the USBBlaster.
# (Note: this example assumes only one USBBlaster connected.)
puts "Programming Hardwares:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
```

continued...

```
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# The follow virtual JTAG IR and DR shift sequence engage with
# the example virtual JTAG instance.
#
# Two instructions: SAMPLE (1) FEED (2)
# SAMPLE instruction samples a 8-bit bus; the captured value shows the number of sample
performed.
# FEED instruction supplies a 8-bit value to the logic connected to this instance.
# Both data registers corresponding to the IR are 8 bit wide.

# Send SAMPLE instruction to IR, read captured IR for the sampling number.
# Capture the DR register for the current sampled value.
device_lock -timeout 10000
puts "Current LED Value (sample #[device_virtual_ir_shift -instance_index 0 -ir_value 1]):
[device_virtual_dr_shift -instance_index 0  -length 8 -value_in_hex]"
device_unlock

# Send FEED instruction to IR, read a two-digit hex string from the console,
# then send the new value to the DR register.
puts "\nType in 2 digits in hexadecimal to update the LED:"
gets stdin update_value
device_lock -timeout 10000
device_virtual_ir_shift -instance_index 0 -ir_value 2 -no_captured_ir_value
device_virtual_dr_shift -instance_index 0  -length 8 -dr_value $update_value -value_in_hex -
no_captured_dr_value
device_unlock

# Close device
close_device
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The specified virtual JTAG instance cannot be found.
	TCL_ERROR	1	ERROR: One of the options, mfg_id, type_id and version is specified, but not all. All of them are required.
	TCL_ERROR	1	ERROR: A device has not been locked; exclusive communication must be obtained first.

3.1.21.9. get_device_names (::quartus::jtag)

The following table displays information for the `get_device_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`get_device_names [-h | -help] [-long_help] -hardware_name` *<hardware name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-hardware_name` *<hardware name>*	The name of the programming hardware that connects to the JTAG chain. The name can be obtained from command: get_hardware_names.
Description		

```
       Retrieves a list of names of the devices on the JTAG
chain to which the specified programming hardware is
connected.

The name of the device is in the following format:
<number on circuit board>: <JTAG ID code>: <device
name>.  For example, in the device name @1:
0x082000DD: EP20K200C, @1 indicates that it is the
```

continued...

```
first device on the circuit board, 0x082000DD is the
JTAG ID code for the device, and EP20K200C is the
device name.
```

Example Usage

```
# List all available programming hardwares, and select the USBBlaster.
# (Note: this example assumes only one USBBlaster connected.)
puts "Programming Hardwares:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# Retrieve device id code.
# IDCODE instruction value is 6; The ID code is 32 bits long.

# IR and DR shift should be locked together to ensure that other applications
# will not change the instruction register before the id code value is shifted
# out while the instruction register is still holding the IDCODE instruction.
device_lock -timeout 10000
device_ir_shift -ir_value 6 -no_captured_ir_value
puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]"
device_unlock

# Close device
close_device
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: No device is detected in the specified JTAG chain.
TCL_ERROR	1	ERROR: The specified hardware is not found.

3.1.21.10. get_hardware_names (::quartus::jtag)

The following table displays information for the `get_hardware_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`get_hardware_names [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description		

```
       Retrieves a list of the names of the programming
hardware attached to and configured for the JTAG
server.

The hardware name is in the following format:
<hardware type> [<port>].

       For example, in the hardware name USB-Blaster [USB-0],
USB-Blaster is the name of the programming hardware,
and USB-0 is the name of the port to which the
hardware is connected.
```

continued...

Example Usage	<pre># List all available programming hardware, and select the USB-Blaster. # (Note: this example assumes only one USB-Blaster is connected.) puts "Programming Hardware:" foreach hardware_name [get_hardware_names] { puts $hardware_name if { [string match "USB-Blaster*" $hardware_name] } { set usbblaster_name $hardware_name } } puts "\nSelect JTAG chain connected to $usbblaster_name.\n"; # List all devices on the chain, and select the first device on the chain. puts "\nDevices on the JTAG chain:" foreach device_name [get_device_names -hardware_name $usbblaster_name] { puts $device_name if { [string match "@1*" $device_name] } { set test_device $device_name } } puts "\nSelect device: $test_device.\n"; # Open device open_device -hardware_name $usbblaster_name -device_name $test_device # Retrieve device id code. # IDCODE instruction value is 6; The ID code is 32 bits long. # IR and DR shift should be locked together to ensure that other applications # will not change the instruction register before the id code value is shifted # out while the instruction register is still holding the IDCODE instruction. device_lock -timeout 10000 device_ir_shift -ir_value 6 -no_captured_ir_value puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]" device_unlock # Close device close_device</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No programming hardware is attached to the JTAG server or it is not configured properly.

3.1.21.11. open_device (::quartus::jtag)

The following table displays information for the open_device Tcl command:

Tcl Package and Version	Belongs to ::quartus::jtag on page 313	
Syntax	`open_device [-h | -help] [-long_help] -device_name` *<device name>* `-hardware_name` *<hardware name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-device_name` *<device name>*	The name of the device on the JTAG chain. The name can be obtained from command: get_device_names
	`-hardware_name` *<hardware name>*	The name of the programming hardware that connects to the JTAG chain. The name can be obtained from command: get_hardware_names
Description	<pre>Initiate a shared JTAG communication with the device. The command, close_device, is called to end the communication with the device. Only one device can be opened per process. Multiple devices can be opened in multiple processes.</pre>	

continued...

Example Usage

```
# List all available programming hardwares, and select the USBBlaster.
# (Note: this example assumes only one USBBlaster connected.)
puts "Programming Hardwares:"
foreach hardware_name [get_hardware_names] {
    puts $hardware_name
    if { [string match "USB-Blaster*" $hardware_name] } {
        set usbblaster_name $hardware_name
    }
}
puts "\nSelect JTAG chain connected to $usbblaster_name.\n";

# List all devices on the chain, and select the first device on the chain.
puts "\nDevices on the JTAG chain:"
foreach device_name [get_device_names -hardware_name $usbblaster_name] {
    puts $device_name
    if { [string match "@1*" $device_name] } {
        set test_device $device_name
    }
}
puts "\nSelect device: $test_device.\n";

# Open device
open_device -hardware_name $usbblaster_name -device_name $test_device

# Retrieve device id code.
# IDCODE instruction value is 6; The ID code is 32 bits long.

# IR and DR shift should be locked together to ensure that other applications
# will not change the instruction register before the id code value is shifted
# out while the instruction register is still holding the IDCODE instruction.
device_lock -timeout 10000
device_ir_shift -ir_value 6 -no_captured_ir_value
puts "IDCODE: 0x[device_dr_shift -length 32 -value_in_hex]"
device_unlock

# Close device
close_device
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: A device was opened. Only one device can be open at a time within this process. Close previously opened device first.
TCL_ERROR	1	ERROR: The specified device is not found.
TCL_ERROR	1	ERROR: The specified hardware is not found.

3.1.22. ::quartus::logic_analyzer_interface

The following table displays information for the **::quartus::logic_analyzer_interface** Tcl package:

Tcl Package and Version	::quartus::logic_analyzer_interface 1.0
Description	This package contains the set of Tcl functions for querying and modifying the Logic Analyzer Interface output pin state in an Intel device.
Availability	This package is loaded by default in the following executables: quartus_stp quartus_stp_tcl
Tcl Commands	begin_logic_analyzer_interface_control (::quartus::logic_analyzer_interface) on page 327 change_bank_to_output_pin (::quartus::logic_analyzer_interface) on page 328 end_logic_analyzer_interface_control (::quartus::logic_analyzer_interface) on page 329 get_current_state_of_output_pin (::quartus::logic_analyzer_interface) on page 330 tristate_output_pin (::quartus::logic_analyzer_interface) on page 331

3.1.22.1. begin_logic_analyzer_interface_control (::quartus::logic_analyzer_interface)

The following table displays information for the `begin_logic_analyzer_interface_control` Tcl command:

Tcl Package and Version	Belongs to ::quartus::logic_analyzer_interface on page 326	
Syntax	`begin_logic_analyzer_interface_control [-h | -help] [-long_help] -device_name` *<device name>* `-file_path` *<file path>* `-hardware_name` *<hardware name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-device_name` *<device name>*	Name of the device to be controlled
	`-file_path` *<file path>*	Path of the Logic Analyzer Interface (.lai) file
	`-hardware_name` *<hardware name>*	Name of the hardware that connects to the JTAG chain
Description	`Starts the Logic Analyzer Interface control sequence to query the Logic Analyzer Interface output pin state and change output pins state. The control sequence should be terminated with end_logic_analyzer_interface_control.` `The hardware and device name can be obtained by using get_hardware_names and get_device_names respectively from the jtag package.`	
Example Usage	`# Start a new control sequence.` `begin_logic_analyzer_interface_control -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1C20 (0x020840DD)" -file_path "lai_demo.lai"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Change input bank source to the output pins` `change_bank_to_output_pin -instance_name "auto_lai_0" -bank_name "Bank 1"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Change input bank source to the output pins` `change_bank_to_output_pin -instance_name "auto_lai_0" -bank_index 0` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Tristate the output pins` `tristate_output_pin -instance_name "auto_lai_0"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# End the control sequence.` `end_logic_analyzer_interface_control`	
Return Value	**Code Name** / **Code**	**String Return**
	TCL_OK / 0	INFO: Operation successful
	TCL_ERROR / 1	ERROR: A Logic Analyzer Interface control sequence has been started already.

continued...

	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The Logic Analyzer Interface file (.lai) cannot be opened, or it is an invalid file.
	TCL_ERROR	1	ERROR: The specified hardware is not found.

3.1.22.2. change_bank_to_output_pin (::quartus::logic_analyzer_interface)

The following table displays information for the `change_bank_to_output_pin` Tcl command:

Tcl Package and Version	Belongs to ::quartus::logic_analyzer_interface on page 326	
Syntax	`change_bank_to_output_pin [-h | -help] [-long_help] [-bank_index` *<bank index>* `] [-bank_name` *<bank name>* `] -instance_name` *<instance name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-bank_index` *<bank index>*	Index of the bank on the mux to be used as the source of the output pins
	`-bank_name` *<bank name>*	Name of the bank to be used as the source of the output pins
	`-instance_name` *<instance name>*	Name of the Logic Analyzer Interface instance to change

Description

```
Change the Logic Analyzer Interface output pin's
source on the specified instance to use the specified
bank.
```

Example Usage

```
# Start a new control sequence.
begin_logic_analyzer_interface_control -hardware_name "USB-Blaster \[USB-0\]" -device_name
"@1: EP1C20 (0x020840DD)" -file_path "lai_demo.lai"

# Query the output pin state.
puts "Current output pin state of instance auto_lai_0:"
puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]

# Change input bank source to the output pins
change_bank_to_output_pin -instance_name "auto_lai_0" -bank_name "Bank 1"

# Query the output pin state.
puts "Current output pin state of instance auto_lai_0:"
puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]

# Change input bank source to the output pins
change_bank_to_output_pin -instance_name "auto_lai_0" -bank_index 0

# Query the output pin state.
puts "Current output pin state of instance auto_lai_0:"
puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]

# Tristate the output pins
tristate_output_pin -instance_name "auto_lai_0"

# Query the output pin state.
puts "Current output pin state of instance auto_lai_0:"
puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]

# End the control sequence.
end_logic_analyzer_interface_control
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: No Logic Analyzer Interface control sequence has been started.

continued...

TCL_ERROR	1	ERROR: The specified device is not found.
TCL_ERROR	1	ERROR: The specified hardware is not found.
TCL_ERROR	1	ERROR: The specified Logic Analyzer Interface instance in the file is not compatible with the instance in the device.
TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure or poor signal integrity in the JTAG chain.
TCL_ERROR	1	ERROR: The specified bank cannot be found in the Logic Analyzer Interface instance.
TCL_ERROR	1	ERROR: The specified Logic Analyzer Interface instance cannot be found.
TCL_ERROR	1	ERROR: The version of the specified Logic Analyzer Interface instance is not supported in this release of software.

3.1.22.3. end_logic_analyzer_interface_control (::quartus::logic_analyzer_interface)

The following table displays information for the `end_logic_analyzer_interface_control` Tcl command:

Tcl Package and Version	Belongs to ::quartus::logic_analyzer_interface on page 326	
Syntax	`end_logic_analyzer_interface_control [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Terminate the Logic Analyzer Interface control sequence.`	
Example Usage	`# Start a new control sequence.` `begin_logic_analyzer_interface_control -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1C20 (0x020840DD)" -file_path "lai_demo.lai"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Change input bank source to the output pins` `change_bank_to_output_pin -instance_name "auto_lai_0" -bank_name "Bank 1"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Change input bank source to the output pins` `change_bank_to_output_pin -instance_name "auto_lai_0" -bank_index 0` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Tristate the output pins` `tristate_output_pin -instance_name "auto_lai_0"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# End the control sequence.` `end_logic_analyzer_interface_control`	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No Logic Analyzer Interface control sequence has been started.

3.1.22.4. get_current_state_of_output_pin (::quartus::logic_analyzer_interface)

The following table displays information for the `get_current_state_of_output_pin` Tcl command:

Tcl Package and Version	Belongs to ::quartus::logic_analyzer_interface on page 326		
Syntax	`get_current_state_of_output_pin [-h | -help] [-long_help] -instance_name` *<instance name>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-instance_name` *<instance name>*		Name of the Logic Analyzer Interface instance to change
Description	`Query the device to get the current state of the output pins of the specified instance.` `The result is either the bank name or "tristated".`		
Example Usage	`# Start a new control sequence.` `begin_logic_analyzer_interface_control -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1C20 (0x020840DD)" -file_path "lai_demo.lai"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Change input bank source to the output pins` `change_bank_to_output_pin -instance_name "auto_lai_0" -bank_name "Bank 1"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Change input bank source to the output pins` `change_bank_to_output_pin -instance_name "auto_lai_0" -bank_index 0` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Tristate the output pins` `tristate_output_pin -instance_name "auto_lai_0"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# End the control sequence.` `end_logic_analyzer_interface_control`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No Logic Analyzer Interface control sequence has been started.
	TCL_ERROR	1	ERROR: The specified device is not found.
	TCL_ERROR	1	ERROR: The specified hardware is not found.

continued...

	TCL_ERROR	1	ERROR: The specified Logic Analyzer Interface instance in the file is not compatible with the instance in the device.
	TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure or poor signal integrity in the JTAG chain.
	TCL_ERROR	1	ERROR: The specified Logic Analyzer Interface instance cannot be found.
	TCL_ERROR	1	ERROR: The version of the specified Logic Analyzer Interface instance is not supported in this release of software.

3.1.22.5. tristate_output_pin (::quartus::logic_analyzer_interface)

The following table displays information for the `tristate_output_pin` Tcl command:

Tcl Package and Version	Belongs to ::quartus::logic_analyzer_interface on page 326		
Syntax	`tristate_output_pin [-h | -help] [-long_help] -instance_name` *<instance name>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-instance_name` *<instance name>*		Name of the Logic Analyzer Interface instance to change
Description	`Tristate the output pins of the specified Logic Analyzer Interface instance.`		
Example Usage	`# Start a new control sequence.` `begin_logic_analyzer_interface_control -hardware_name "USB-Blaster \[USB-0\]" -device_name "@1: EP1C20 (0x020840DD)" -file_path "lai_demo.lai"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Change input bank source to the output pins` `change_bank_to_output_pin -instance_name "auto_lai_0" -bank_name "Bank 1"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Change input bank source to the output pins` `change_bank_to_output_pin -instance_name "auto_lai_0" -bank_index 0` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# Tristate the output pins` `tristate_output_pin -instance_name "auto_lai_0"` `# Query the output pin state.` `puts "Current output pin state of instance auto_lai_0:"` `puts [get_current_state_of_output_pin -instance_name "auto_lai_0"]` `# End the control sequence.` `end_logic_analyzer_interface_control`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No Logic Analyzer Interface control sequence has been started.
	TCL_ERROR	1	ERROR: The specified device is not found.

continued...

	TCL_ERROR	1	ERROR: The specified hardware is not found.
	TCL_ERROR	1	ERROR: The specified Logic Analyzer Interface instance in the file is not compatible with the instance in the device.
	TCL_ERROR	1	ERROR: JTAG communication error is detected. It can be caused by the hardware failure or poor signal integrity in the JTAG chain.
	TCL_ERROR	1	ERROR: The specified Logic Analyzer Interface instance cannot be found.
	TCL_ERROR	1	ERROR: The version of the specified Logic Analyzer Interface instance is not supported in this release of software.

3.1.23. ::quartus::misc

The following table displays information for the **::quartus::misc** Tcl package:

Tcl Package and Version	::quartus::misc 1.0
Description	This package contains a set of Tcl functions for performing miscellaneous tasks.
Availability	This package is loaded by default in the following executables: hdb_debug qacv qpro qpro_sh quartus quartus_cdb quartus_drc quartus_eda quartus_fit quartus_ipc quartus_ipd quartus_ipgenerate quartus_map quartus_sh quartus_si quartus_sim quartus_sta quartus_stp
Tcl Commands	checksum (::quartus::misc) on page 333 disable_natural_bus_naming (::quartus::misc) on page 333 enable_natural_bus_naming (::quartus::misc) on page 334 escape_brackets (::quartus::misc) on page 335 foreach_in_collection (::quartus::misc) on page 336 get_collection_size (::quartus::misc) on page 338 get_environment_info (::quartus::misc) on page 339 get_message_count (::quartus::misc) on page 339 init_tk (::quartus::misc) on page 340 load (::quartus::misc) on page 340 load_package (::quartus::misc) on page 341 post_message (::quartus::misc) on page 342 qerror (::quartus::misc) on page 343 qexec (::quartus::misc) on page 343 qexit (::quartus::misc) on page 344 record_tcl_cmd (::quartus::misc) on page 344 stopwatch (::quartus::misc) on page 345

3.1.23.1. checksum (::quartus::misc)

The following table displays information for the `checksum` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`checksum [-h | -help] [-long_help] [-algorithm <crc32|adler32> ] <input_file>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-algorithm <crc32|adler32>`	Option to specify the checksum algorithm. Uses the CRC-32 algorithm by default.
	<input_file>	Option to specify the input file
Description	`Returns the checksum value in hexadecimal format based on the specified algorithm which defaults to crc32.`	
Example Usage	`set file "one_wire.sof"` `  # Use CRC-32` `puts "$file -> [checksum $file]"` `puts "$file -> [checksum $file -algorithm crc32]"` `  # Use ADLER-32` `puts "$file -> [checksum $file -algorithm adler32]"`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't read file: *<string>*. Make sure the file exists and is not a directory, and you have permission to read the file.

3.1.23.2. disable_natural_bus_naming (::quartus::misc)

The following table displays information for the `disable_natural_bus_naming` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`disable_natural_bus_naming [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Disables natural bus naming.` `You may choose to disable natural bus naming to string match patterns such as "in\[024\]". In this example, you are string matching the names "in0", "in2", and "in4".` `Note that if you disable natural bus naming, then square brackets must be escaped twice (\\\[ or \\\]) so that the strings are interpreted as bus names during a string match, such as:` `set bus_name "address\[0\]" string match [escape_brackets $bus_name] $bus_name` `The "escape_brackets" command escapes "address\[0\]" into "address\\\[0\\\]".` `To enable natural bus naming again, type "enable_natural_bus_naming".`	

continued...

	For more information about natural bus naming, type "enable_natural_bus_naming -h".		
Example Usage	# Disables natural bus naming disable_natural_bus_naming		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.23.3. enable_natural_bus_naming (::quartus::misc)

The following table displays information for the `enable_natural_bus_naming` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332		
Syntax	`enable_natural_bus_naming [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	Enables natural bus naming so that square brackets for bus names do not have to be escaped to prevent Tcl from interpreting them as sub-commands. Bus names have the following format: <bus name>[<bus index>] or <bus name>[*] The <bus name> portion is a string of alphanumeric characters. The <bus index> portion is an integer greater than or equal to zero or it can be the character "*" used for string matching. Notice that the <bus index> is enclosed by the square brackets "[" and "]". For example, "a[0]" and "a[*]" are supported bus names. Many Quartus Prime Tcl commands allow bus names in their arguments, such as: set_location_assignment -to address[10] Pin_M20 If natural bus naming is enabled, you can just use address[10] instead of having to excape the square brackets into address\[10\]. There are also Quartus Prime Tcl commands that take Tcl string match patterns in their arguments, such as: get_all_instance_assignments -name location -to address[10] Since Tcl string matching take string patterns containing special characters from the set "*?\[]" as values, address[10] would be interpreted incorrectly. By enabling natural bus naming, these Tcl commands will automatically detect address[10] as a bus name so that you don't have to doubly escape the brackets into address\\\[10\\\]. To disable natural bus naming, type "disable_natural_bus_naming". For more information on the effects of disabling natural bus naming, type "disable_natural_bus_naming -h".		
Example Usage	# Enables natural bus naming enable_natural_bus_naming		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.23.4. escape_brackets (::quartus::misc)

The following table displays information for the `escape_brackets` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`escape_brackets [-h | -help] [-long_help] <str>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<str>	String to escape
Description		

```
Escapes square brackets in bus name patterns for use in string
matching. Also escapes the escape character; for example, the string
"\" is escaped into "\\".

Note that natural bus naming is supported by default. This means that
bus names, not bus name patterns, are automatically detected and do
not need to be escaped by using this command.  Bus names have the
following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.  The
<bus index> portion is an integer greater than or equal to zero or it
can be the character "*" used for string matching.  Notice that the
<bus index> is enclosed by the square brackets "[" and "]". For
example, "a[0]" and "a[*]" are supported bus names.

All other uses of square brackets must be escaped if you do not intend
to use the brackets as part of a string pattern in a string match. For
example, the bus name pattern "a\[0-2\]" must be escaped using this
"escape_brackets" command since the "0-2" does not satisfy the <bus
index> requirement to be a bus name.

Many Quartus Prime Tcl commands allow string matching in option
arguments. A common error is using bus name patterns in these
arguments, such as:

address\[0-2\]

Square brackets for bus name patterns must already be preceded by an
escape character (\[ or \]) to prevent Tcl from interpreting them as
sub-commands. String matching, however, also uses square brackets to
match any character between the brackets.  The previous example, when
used as a string match pattern, searches for the string patterns
address0, address1, and address2. It does not search for address[0],
address[1], and address[2].

Therefore, for arguments that support string matching, square brackets
must be escaped twice (\\\[ or \\\]) so that the strings are
interpreted as bus name patterns. For example, to search for
address[0], address[0], and address[2], type the following string
match pattern:

address\\\[\[0-2\]\\\]

or, equivalently,

"address[escape_brackets \[]\[0-2\][escape_brackets \]]"

Quartus Prime Tcl commands do not convert bus name patterns
automatically, since they cannot determine if the string is intended
as a bus name pattern or a regular string match pattern. Therefore,
"escape_brackets" is provided for your convenience.

You may choose to disable natural bus naming in order to string match
patterns such as "in\[024\]". In this example, you are string matching
the names "in0", "in2", and "in4".

To disable natural bus naming, type
"disable_natural_bus_naming".

Note that if you disable natural bus naming, then square brackets must
be escaped twice (\\\[ or \\\]) so that the strings are interpreted as
bus names during a string match, such as:

set bus_name "address\[0\]"
string match [escape_brackets $bus_name] $bus_name
```

continued...

	The "escape_brackets" command escapes "address\[0\]" into "address\\\[0\\\]". To enable natural bus naming again, type "enable_natural_bus_naming". For more information about natural bus naming, type "enable_natural_bus_naming -h".
Example Usage	# Get all location assignments for bus address[] set address_names address[*] set address_locations [get_all_instance_assignments -to $address_names] -name LOCATION # Get location assignment for bus address[0] set address_names address[0] set address_locations [get_all_instance_assignments -to $address_names] -name LOCATION # Get location assignments for bus address[0], # address[1], and address[2] set address_names address[0-2] set address_locations [get_all_instance_assignments -to [escape_brackets $address_names]] - name LOCATION

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.23.5. foreach_in_collection (::quartus::misc)

The following table displays information for the `foreach_in_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`foreach_in_collection [-h | -help] [-long_help]` *<variable_name>* *<collection>* *<body>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<variable_name>	Variable name
	<collection>	Collection
	<body>	Body

Description

```
Accesses each element of a collection.

Some Tcl commands return a collection. The following table shows
examples of commands that return a collection:

Tcl package                Tcl commands (returning a collection)
-------------------------  ------------------------------------
::quartus::project         get_all_quartus_defaults
                           get_all_global_assignments
                           get_all_instance_assignments
                           get_all_parameters
                           get_names
                           assignment_group (only for the "-get_members"
                                             and "-get_exceptions" options)

::quartus::chip_editor     get_nodes
                           get_iports
                           get_oports

The command is used in the following format:

foreach_in_collection <variable name> <collection> {

      # This is the body of "foreach_in_collection"
      ...
}
```

continued...

```
Unlike a Tcl list, a collection is a container specific to the Quartus
II software, whose elements can be accessed by using the
"foreach_in_collection" command.
```

Example Usage

```
## Get a collection of global assignments
set collection_of_global_assignments [get_all_global_assignments -name *]
## Display the collection string representation
puts $collection_of_global_assignments
## Iterate through the collection and display
## the information for each global assignment
foreach_in_collection global $collection_of_global_assignments {

    set sect_id [lindex $global 0]
    set name [lindex $global 1]
    set value [lindex $global 2]

    ## Now, display the content of the global assignment
    puts "Section ID ($sect_id)"
    puts "Assignment Name ($name)"
    puts "Assignment Value ($value)"
}

## Get a collection of instance assignments
set collection_of_instance_assignments [get_all_instance_assignments -name *]
## Display the collection string representation
puts $collection_of_instance_assignments
## Iterate through the collection and display
## the information for each instance assignment
foreach_in_collection instance $collection_of_instance_assignments {

    set sect_id [lindex $instance 0]
    set src [lindex $instance 1]
    set dest [lindex $instance 2]
    set name [lindex $instance 3]
    set value [lindex $instance 4]

    ## Now, display the content of the instance assignment
    puts "Section ID ($sect_id)"
    puts "Source ($src)"
    puts "Destination ($dest)"
    puts "Assignment Name ($name)"
    puts "Assignment Value ($value)"
}

## Get a collection of parameters
set collection_of_parameters [get_all_parameters -name *]
## Display the collection string representation
puts $collection_of_parameters
## Iterate through the collection and display
## the information for each parameter
foreach_in_collection parameter $collection_of_parameters {

    set dest [lindex $parameter 0]
    set name [lindex $parameter 1]
    set value [lindex $parameter 2]

    ## Now, display the content of the parameter
    puts "Destination ($dest)"
    puts "Parameter Name ($name)"
    puts "Parameter Value ($value)"
}

## Get a collection of all node name ids from a successful
## compilation
set collection_of_name_ids [get_names -filter *]
## Display the collection string representation
puts $collection_of_name_ids
## Iterate through the collection and display
## the information for each name id
foreach_in_collection name_id $collection_of_name_ids {

    set parent_name_id [get_name_info -info parent_name_id $name_id]
    set base_name [get_name_info -info base_name $name_id]
    set entity_name [get_name_info -info entity_name $name_id]
    set instance_name [get_name_info -info instance_name $name_id]
    set full_path [get_name_info -info full_path $name_id]
    set short_full_path [get_name_info -info short_full_path $name_id]
    set node_type [get_name_info -info node_type $name_id]
    set creator [get_name_info -info creator $name_id]
    set signaltapii [get_name_info -info signaltapii $name_id]
    set file_location [get_name_info -info file_location $name_id]

    ## Now, display information about the name
    puts "Parent Name Id ($parent_name_id)"
    puts "Base Name ($base_name)"
    puts "Entity Name ($entity_name)"
```

continued...

```
    puts "Instance Name ($instance_name)"
    puts "Full Path ($full_path)"
    puts "Short Full Path ($short_full_path)"
    puts "Node Type ($node_type)"
    puts "Creator ($creator)"
    puts "Signaltapii ($signaltapii)"
    puts "File location ($file_location)"
}

# Display the members of a particular assignment group named "tg1"
foreach_in_collection member [assignment_group "tg1" -get_members] {

    # Print the name of the member
    puts $member
}

# Display the exception to a particular assignment group named "tg1"
foreach_in_collection exception [assignment_group "tg1" -get_exceptions] {

    # Print the name of the exception
    puts $exception
}
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.23.6. get_collection_size (::quartus::misc)

The following table displays information for the `get_collection_size` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332		
Syntax	`get_collection_size [-h | -help] [-long_help] [ <collection> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<collection>	Collection	
Description			

```
Returns the size of a collection.

Unlike a Tcl list, a collection is a container specific to the
Quartus Prime software, whose elements can be accessed by using the
"foreach_in_collection" command.
```

Example Usage

```
## Displays the number of global assignments
project_open chiptrip
set num_global_asgns [get_collection_size [get_all_global_assignments -name {*}]]
puts $num_global_asgns
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Collection does not exist with name: *<string>*. Specify the collection name returned by a Tcl command that supports foreach_in_collection. Note a valid collection name can become invalid if the variable holding the collection goes out of scope, as well as a result of some built-in TCL commands, for example 'string length'.
	TCL_ERROR	1	ERROR: Nested calls to foreach_in_collection with the same collection name *<string>* are not allowed. Specify a different collection name for the other collection.

3.1.23.7. get_environment_info (::quartus::misc)

The following table displays information for the `get_environment_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332		
Syntax	`get_environment_info [-h | -help] [-long_help] [-num_logical_processors] [-num_physical_processors] [-operating_system]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-num_logical_processors`	Option to return the number of available logical processors (cores including hyper-threading)	
	`-num_physical_processors`	Option to return the number of available physical processors (sockets)	
	`-operating_system`	Option to return the operating system name	
Description	`Returns information about the system environment depending on the options specified.`		
Example Usage	`# Get the number of physical processors available on my computer.` `get_environment_info -num_physical_processors`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: At least one option is required. Specify at least one option.
	TCL_ERROR	1	ERROR: Multiple options used. Choose only one option for this command.

3.1.23.8. get_message_count (::quartus::misc)

The following table displays information for the `get_message_count` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`get_message_count [-h | -help] [-long_help] -type` *<info|extra_info|warning|critical_warning|error>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-type` *<info|extra_info|warning|critical_warning|error>*	Type of message
Description	`Get a count of messages of a specific message type.` `The message type can be "info", "warning", "critical_warning", or "error".`	
Example Usage	`# Get count of error messages` `get_message_count -type error` `# Get count of warning messages` `get_message_count -type warning`	

continued...

	`` # Get count of critical warning messages get_message_count -type critical_warning ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal message type: <*string*>. Specify info, warning, critical_warning, or error.
	TCL_ERROR	1	ERROR: Missing required positional argument: <*string*>. Specify the <*string*> argument.
	TCL_ERROR	1	ERROR: You specified <*string*> arguments to the -args option. However, you can pass a maximum of <*string*> arguments.

3.1.23.9. init_tk (::quartus::misc)

The following table displays information for the `init_tk` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332		
Syntax	`init_tk [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	``` Initializes a Tk window. If you are using Tk functionality in Tcl, you must run this command first before running any Tcl scripts. ```		
Example Usage	``` # Initialize the Tk library init_tk # Create a top level and add a title toplevel .top wm title .top "Hello World" # Add widgets button .top.hello -text Hello -command {puts stdout "Hello, World!"} pack .top.hello -padx 20 -pady 10 # Without "tkwait", the script finishes at this point and the # window is destroyed. The "tkwait" command prevents the # script from finishing until the you close the window. tkwait window .top ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.23.10. load (::quartus::misc)

The following table displays information for the `load` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`load [-h | -help] [-long_help]` <*load_args*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	<load_args>		Arguments to load
Description	Loads machine code and initializes new commands. This command works exactly like Tcl's native "load" command.		
Example Usage	Refer to documentation for Tcl's native "load" command at the Tcl/Tk web site at www.tcl.tk.		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't load library: *<string>*. The operating system reports the following error: *<string>*

3.1.23.11. load_package (::quartus::misc)

The following table displays information for the `load_package` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332		
Syntax	`load_package [-h | -help] [-long_help] [-version` *<version number>* `]` *<package name>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-version` *<version number>*		Option to specify the Quartus Prime Tcl package version to load
	<package name>		Name of Quartus Prime Tcl package to load
Description	Loads the specified Quartus Prime Tcl package with the specified version number. If you do not specify the "-version" option, the latest version is loaded by default. The Quartus Prime Tcl package names have the "::quartus::" prefix, such as "::quartus::project". For convenience, you can omit the "::quartus::" prefix when you use the <package name> argument. This command is similar to the "package require" command. The advantage of using "load_package" is that you can alternate freely between different versions of the same package. For example, if you loaded version 2.0, and now want to load version 1.0, you can type: "load_package -version 1.0 <package name>".		
Example Usage	# Load version 1.0 of the ::quartus::project package load_package project -version 1.0 # Load version 2.0 of the ::quartus::project package load_package project -version 2.0		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Tcl package *<string>* does not exist. Specify an available Quartus Prime Tcl package. Type help for a list of available Quartus Prime Tcl packages.

continued...

	TCL_ERROR	1	ERROR: Tcl package <*string*> version <*string*> does not exist. Specify an available Quartus Prime Tcl package. Type help for a list of available Quartus Prime Tcl packages.
	TCL_ERROR	1	ERROR: Tcl package <*string*> is only available in Quartus Pro Edition
	TCL_ERROR	1	ERROR: Tcl package <*string*> is only available in Quartus Standard Edition

3.1.23.12. post_message (::quartus::misc)

The following table displays information for the `post_message` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332		
Syntax	`post_message [-h | -help] [-long_help] [-type` <*info*|*extra_info*|*warning*|*critical_warning*|*error*> `] [` <*string*> `]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-type` <*info*|*extra_info*|*warning*|*critical_warning*|*error*>	Type of message to display	
	<*string*>	Message to be displayed	
Description	`Displays messages and sub-messages of the specified type.` `The message type can be "info", "warning", "critical_warning", or "error".` `If you do not use the -type option, the default message type is "info".` `The "extra_info" type is deprecated. Messages with this type will not be displayed. Use the "info" type instead.` `Use the -submsgs option to group messages indented under a message. The -submsgs option posts each string in a Tcl list of strings as a sub-message. The sub-messages have the same message type as the main message.`		
Example Usage	`# Display an error message` `post_message -type error "Can't open file test.tcl"` `# Display an info message` `post_message "Generated output file: test.out"` `# OR` `post_message -type info "Generated output file: test.out"` `# Display an info message with a sub-message` `post_message "Generated output file: test.out" -submsgs [list "Ouput file saved in project directory"]` `# Display a warning message` `post_message -type warning "Defaulting fmax to 155.55mhz"` `# Display a critical warning message` `post_message -type critical_warning "Invalid fmax was specified - defaulting to 155.55mhz"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Error(s) found while processing handle: <*string*>. Make sure you specified a valid Quartus Prime message handle for the -<*string*> option and passed the correct

continued...

			number of arguments to the -<*string*> option. Also check that you passed the correct message type to the -<*string*> option.
	TCL_ERROR	1	ERROR: Illegal message type: <*string*>. Specify info, warning, critical_warning, or error.
	TCL_ERROR	1	ERROR: Missing required positional argument: <*string*>. Specify the <*string*> argument.
	TCL_ERROR	1	ERROR: You specified <*string*> arguments to the -args option. However, you can pass a maximum of <*string*> arguments.

3.1.23.13. qerror (::quartus::misc)

The following table displays information for the `qerror` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332		
Syntax	`qerror [-h | -help] [-long_help] [-error_null] [-over_malloc] [-qt_segfault] [-std_segfault]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-error_null`		Option to error by dereferencing null
	`-over_malloc`		Option to error by malloc'ing INT_MAX
	`-qt_segfault`		Accesses an element at an invalid index in a QT container
	`-std_segfault`		Accesses an element at an invalid index in a std container
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.23.14. qexec (::quartus::misc)

The following table displays information for the `qexec` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`qexec [-h | -help] [-long_help] <command>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*command*>	Command
Description	`Runs the specified shell command from within a Tcl shell or script.` `Usage for this command is as follows:` `  qexec "<command>"`	

continued...

Example Usage	`qexec ls`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.23.15. qexit (::quartus::misc)

The following table displays information for the `qexit` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`qexit [-h | -help] [-long_help] [-error] [-success]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-error`	Option to exit with an equivalent Quartus Prime error code
	`-success`	Option to exit with an equivalent Quartus Prime success code
Description	`Exits the Quartus Prime software.` `The Quartus Prime Tcl command "qexit" is equivalent to the Tcl command "exit".` `When used with a particular option, this command exits the Quartus Prime software with the corresponding Quartus Prime exit code. For example, typing "qexit -success" is equivalent to typing "exit 0". When the "-success" option is specified, the corresponding Quartus Prime exit code is "0".` `If no option is specified, the default exit code is the same as for the "-success" option.`	
Example Usage	`1) To exit the Quartus Prime software with an equivalent Quartus Prime success code, type:` `qexit -success` `2) To exit the Quartus Prime software with an equivalent Quartus Prime error code, type:` `qexit -error`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful

3.1.23.16. record_tcl_cmd (::quartus::misc)

The following table displays information for the `record_tcl_cmd` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332	
Syntax	`record_tcl_cmd [-h | -help] [-long_help] [-filename` *<file name>* `] <start_end>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-filename` *<file name>*	Option to specify an user filename
	<start_end>	To start or end recording

continued...

Description	`Start or end to record Tcl commands.`		
Example Usage	`1) Type "record_tcl_cmd" start -filename user.rec` `To start record Tcl commands with filename "user.rec"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.23.17. stopwatch (::quartus::misc)

The following table displays information for the `stopwatch` Tcl command:

Tcl Package and Version	Belongs to ::quartus::misc on page 332		
Syntax	`stopwatch [-h | -help] [-long_help] [-lap_time] [-number_format] [-reset] [-start]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-lap_time`	Option to get the lap time	
	`-number_format`	Option to show the lap time in seconds without appending the "s", i.e. seconds, character	
	`-reset`	Option to reset the stopwatch	
	`-start`	Option to start the stopwatch	
Description	`Provides a stopwatch interface.`		
Example Usage	`    # Begin the stopwatch` `stopwatch -start` `exec sleep 1` `    # Get the lap time` `puts [stopwatch -lap_time]` `exec sleep 1` `    # Get the lap time` `puts [stopwatch -lap_time]` `    # Reset the stopwatch` `stopwatch -reset`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.24. ::quartus::names

The following table displays information for the **::quartus::names** Tcl package:

Tcl Package and Version	::quartus::names 1.0
Description	`This package contains no general description.`
Availability	`This package is available for loading in the following executables:` `    qpro_sh` `    quartus_asm` `    quartus_cdb` `    quartus_eda`

continued...

	quartus_fit quartus_map quartus_pow quartus_sh quartus_sta quartus_syn
Tcl Commands	get_assignment (::quartus::names) on page 346 set_assignment (::quartus::names) on page 346

3.1.24.1. get_assignment (::quartus::names)

The following table displays information for the `get_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::names on page 345		
Syntax	`get_assignment [-h | -help] [-long_help] -dict -name <ASSIGNMENT_NAME> [-to <ASSIGNMENT_TO> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-dict`	Indicates this command should return it's result as a Tcl dict object.	
	`-name <ASSIGNMENT_NAME>`	The assignment name to get the value of.	
	`-to <ASSIGNMENT_TO>`	The node to retrieve the assignment value from (omitting this will retrieve the global assignment value, if any).	
Description	Gets the assignment's value. If -to is specified, tries to get the assignment value on the given node, as long as the assignment targeted the node using it's -to field. If -to is not specified, the global value is returned (if any). If the assignment is not found, a Tcl error is produced. If the assignment is found and the -dict option is given, a Tcl dict object is returned with the assignment 'name' (the given input), and the 'value' retrieved from the found assignment. In addition, if the assignment has a 'to' or 'from' field, those fields will also exist in the returned dict.		
Example Usage	`# Returns fitter seed` `get_assignment SEED`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No assignment matching the name <*string*> was found.

3.1.24.2. set_assignment (::quartus::names)

The following table displays information for the `set_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::names on page 345	
Syntax	`set_assignment [-h | -help] [-long_help] -name <ASSIGNMENT_NAME> [-to <INSTANCE_NAME> ] -value <VALUE>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <ASSIGNMENT_NAME>`	The assignment name to set.

continued...

	`-to <INSTANCE_NAME>`	The node to which the assignment should be applied.
	`-value <VALUE>`	The value to set the assignment to.
Description	`Sets the assignment to the given value. If -to is specified, tries to set the assignment value to the given node. If -to is not specified, this applies to global assignments.`	
Example Usage	`set_assignment SEED 9`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No assignment matching the name <*string*> was found.

3.1.25. ::quartus::periph

The following table displays information for the **::quartus::periph** Tcl package:

Tcl Package and Version	::quartus::periph 1.0
Description	`This package contains the set of Tcl functions for interacting with the Interface Planner plans.`
Availability	`This package is available for loading in the following executable:` `quartus_fit`
Tcl Commands	blueprint::initialize (::quartus::periph) on page 352 blueprint::shutdown (::quartus::periph) on page 362 periph::check_plan (::quartus::periph) on page 348 periph::get_cell_info (::quartus::periph) on page 348 periph::get_cells (::quartus::periph) on page 349 periph::get_location_info (::quartus::periph) on page 350 periph::get_placement_info (::quartus::periph) on page 351 periph::get_placements (::quartus::periph) on page 352 periph::load_floorplan (::quartus::periph) on page 353 periph::place_cells (::quartus::periph) on page 353 periph::remove_invalid_reports (::quartus::periph) on page 354 periph::report_all (::quartus::periph) on page 355 periph::report_cell_connectivity (::quartus::periph) on page 355 periph::report_cell_placement_reasons (::quartus::periph) on page 356 periph::report_cells (::quartus::periph) on page 356 periph::report_clocks (::quartus::periph) on page 357 periph::report_legal_cell_locations (::quartus::periph) on page 357 periph::report_location_types (::quartus::periph) on page 358 periph::report_locations (::quartus::periph) on page 358 periph::report_noc_performance (::quartus::periph) on page 359 periph::report_regions (::quartus::periph) on page 359 periph::report_summary (::quartus::periph) on page 360 periph::reset_plan (::quartus::periph) on page 360 periph::save_floorplan (::quartus::periph) on page 361 periph::set_clock_type (::quartus::periph) on page 361 periph::undo_last_placement (::quartus::periph) on page 362 periph::unplace_cells (::quartus::periph) on page 363 periph::update_pdpw (::quartus::periph) on page 363 periph::update_plan (::quartus::periph) on page 364 periph::write_plan (::quartus::periph) on page 364

3.1.25.1. periph::check_plan (::quartus::periph)

The following table displays information for the `periph::check_plan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::check_plan [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Checks the legality of the current plan`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `periph::check_plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.2. periph::get_cell_info (::quartus::periph)

The following table displays information for the `periph::get_cell_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::get_cell_info [-h | -help] [-long_help] [-children] [-guide_cell_id] [-ip_type] [-links] [-location] [-name] [-parent] [-type] <cell_id>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-children`	Return the the cell id of the children cells
	`-guide_cell_id`	Returns the guide cell's elem_id
	`-ip_type`	Returns the IP type if the cell is an IP cell or an empty string otherwise
	`-links`	Return the given design element's connections to other cells
	`-location`	Returns the location ID if the cell is placed or an empty string otherwise
	`-name`	Return the cell name of the cell id
	`-parent`	Return the the cell id of the parent cells
	`-type`	Return the the type of the cell
	<cell_id>	Single cell id
Description	`Gets information about the specified cell (referenced by cell ID).` `You can obtain cell using the periph::get_cells Tcl command.`	

continued...

Example Usage	<pre>project_open onewire_nf blueprint::initialize periph::update_plan foreach cell [periph::get_cells -type IO_CLUSTER] { puts "Found cell ID $cell named [periph::get_cell_info -name $cell]" } blueprint::shutdown project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied cell id <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one cell ID must be supplied, but no cell IDs were supplied
	TCL_ERROR	1	ERROR: <*string*> cell IDs were expected but <*string*> were supplied

3.1.25.3. periph::get_cells (::quartus::periph)

The following table displays information for the `periph::get_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::get_cells [-h | -help] [-long_help] [-atom_only] [-instance_only] [-ip_only] [-num_location` <*num_location*> `] [-physical_only] [-placed] [-toplevel_only] [-type` <*type*> `] [-unplaced] [` <*filter*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-atom_only`	Return only atom cells
	`-instance_only`	Return only instance cells
	`-ip_only`	Return only instance cells that are IP instances
	`-num_location` <*num_location*>	Specify a specific number of available locations the cell should have
	`-physical_only`	Return only physical cells
	`-placed`	Return only placed cells
	`-toplevel_only`	Return only toplevel cells
	`-type` <*type*>	Return only cells of the given types
	`-unplaced`	Return only unplaced cells
	<*filter*>	Object filter
Description	<pre>Returns a list of cells IDs in the design. All cell names in the collection match the specified pattern. Wildcards can be used to select multiple cells at once. When you use the wildcard matching, use pipe characters to separate one hierarchy level from the next. They are treated as special characters and are taken into account when string matching with wildcards is performed. When this matching scheme is enabled, the specified pattern is matched against absolute cell names: the names that include the entire hierarchical path. A full cell name can contain multiple pipe characters in it to reflect the hierarchy. All</pre>	

continued...

	hierarchy levels in the pattern are matched level by level. Any included wildcards refer to only one hierarchical level. For example, "*" and "*|*" produce different collections since they refer to the highest hierarchical level and second highest hierarchical level respectively.		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `foreach cell [periph::get_cells -type IO_CLUSTER] {` `  puts "Found cell ID $cell named [periph::get_cell_info -name $cell]"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied number of locations <*string*> is invalid.
	TCL_ERROR	1	ERROR: <*string*> number of locations were expected but <*string*> were supplied

3.1.25.4. periph::get_location_info (::quartus::periph)

The following table displays information for the `periph::get_location_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::get_location_info [-h | -help] [-long_help] [-children] [-gid] [-name] [-parents] [-placed_cells] [-properties] [-type]` <*location_id*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-children`	Query the children location IDs
	`-gid`	Query the gid of the location IDs
	`-name`	Return the location name of the location id
	`-parents`	Query the parent location IDs
	`-placed_cells`	Return the placed cells at the location id
	`-properties`	Return the device location properties in json
	`-type`	Return the location type of the location id
	<*location_id*>	location id
Description	Gets information about the specified location (referenced by location ID). You can obtain location using the periph::get_locations Tcl command or using the periph::get_cell_info -location Tcl command	
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `foreach cell [periph::get_cells -placed] {` `  puts "Found cell ID $cell named [periph::get_cell_info -name $cell] placed in location`	

continued...

	`[periph::get_cell_info -location $cell] named [periph::get_location_info -name [periph::get_cell_info -location $cell]]" }`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied location id <*string*> is invalid.
	TCL_ERROR	1	ERROR: The supplied type <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one device location ID must be supplied, but no location IDs were supplied
	TCL_ERROR	1	ERROR: At least one type must be supplied, but no types were supplied
	TCL_ERROR	1	ERROR: <*string*> location IDs were expected but <*string*> were supplied
	TCL_ERROR	1	ERROR: <*string*> types were expected but <*string*> were supplied

3.1.25.5. periph::get_placement_info (::quartus::periph)

The following table displays information for the `periph::get_placement_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::get_placement_info [-h | -help] [-long_help] [-placement] <placement_id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-placement`		Return the placement as a list of cell/id pairs
	<*placement_id*>		Single placement id
Description	`Return information about a given placement`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied placement id <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one placement ID must be supplied, but no placement IDs were supplied
	TCL_ERROR	1	ERROR: <*string*> placement IDs were expected but <*string*> were supplied

3.1.25.6. periph::get_placements (::quartus::periph)

The following table displays information for the `periph::get_placements` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::get_placements [-h | -help] [-long_help] <cell_id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<cell_id>		Single cell id
Description	`Returns a vector of placements for the supplied cell`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied cell id *<string>* is not a placeable cell.
	TCL_ERROR	1	ERROR: The supplied cell id *<string>* is invalid.
	TCL_ERROR	1	ERROR: At least one cell ID must be supplied, but no cell IDs were supplied
	TCL_ERROR	1	ERROR: *<string>* cell IDs were expected but *<string>* were supplied

3.1.25.7. blueprint::initialize (::quartus::periph)

The following table displays information for the `blueprint::initialize` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`blueprint::initialize [-h | -help] [-long_help] [-load_compiler_snapshot] [-read_settings_files <on|off> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-load_compiler_snapshot`	Initialize the Interface Planner placement state with the latest compiler snapshot. Use periph::compiler_snapshot_exists to check if a compiler snapshot can be loaded
	`-read_settings_files <on|off>`	Option to identify that settings files should be read from disk
Description	`Initialize the Interface Planner planning engine. Initialization consists of loading all required databases from disk for the design and device. Once initialization is complete, the constraints have not yet been applied and must be done before placement can begin.` `After the assignments have been created, they can be applied to the project by running the plan::update_platform Tcl command.`	

continued...

Example Usage	<pre>project_open onewire_nf blueprint::initialize periph::update_plan blueprint::shutdown project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.8. periph::load_floorplan (::quartus::periph)

The following table displays information for the `periph::load_floorplan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::load_floorplan [-h | -help] [-long_help] -filename` *<filename>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-filename` *<filename>*	Filename to load
Description	<pre>Load the floorplan from a Interface Planner floorplan file</pre>	
Example Usage	<pre>project_open onewire_nf blueprint::initialize periph::update_plan periph::place_cells -unplaced_cells periph::save_floorplan -filename onewire_blueprint_floorplan.plan periph::load_floorplan -filename onewire_blueprint_floorplan.plan project_close</pre>	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.9. periph::place_cells (::quartus::periph)

The following table displays information for the `periph::place_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::place_cells [-h | -help] [-long_help] [-cell_location` *<cell_location>* `] [-cells` *<cells>* `] [-dont_revert_on_fail] [-fixed_cells] [-placement` *<placement>* `] [-unplaced_cells]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cell_location` *<cell_location>*	Cell location id pair to place cells into
	`-cells` *<cells>*	One or more cell ids

continued...

	`-dont_revert_on_fail`	Option to specify that the best partial placement should be saved on the undo stack upon a placement failure
	`-fixed_cells`	Place all unplaced cells
	`-placement` *<placement>*	Place cells according to a placement. A placement is a special object that comes from the periph::get_placements Tcl command
	`-unplaced_cells`	Place all unplaced cells
Description		
	`Performs a placement on the supplied cells`	
Example Usage		
	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `periph::place_cells -unplaced_cells` `periph::check_plan` `project_close`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied cell id <*string*> is invalid.
	TCL_ERROR	1	ERROR: The supplied cell id / location id pair <*string*> is invalid.
	TCL_ERROR	1	ERROR: The supplied location id <*string*> is invalid.
	TCL_ERROR	1	ERROR: The supplied placement id <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one cell ID must be supplied, but no cell IDs were supplied
	TCL_ERROR	1	ERROR: At least one device location ID must be supplied, but no location IDs were supplied
	TCL_ERROR	1	ERROR: At least one placement ID must be supplied, but no placement IDs were supplied
	TCL_ERROR	1	ERROR: <*string*> cell IDs were expected but <*string*> were supplied
	TCL_ERROR	1	ERROR: <*string*> location IDs were expected but <*string*> were supplied
	TCL_ERROR	1	ERROR: <*string*> placement IDs were expected but <*string*> were supplied

3.1.25.10. periph::remove_invalid_reports (::quartus::periph)

The following table displays information for the `periph::remove_invalid_reports` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::remove_invalid_reports [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

Description	Remove all invalid report		
Example Usage	This command currently contains no example usage.		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.11. periph::report_all (::quartus::periph)

The following table displays information for the `periph::report_all` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::report_all [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Create all default summary reports	
Example Usage	This command currently contains no example usage.	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.12. periph::report_cell_connectivity (::quartus::periph)

The following table displays information for the `periph::report_cell_connectivity` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::report_cell_connectivity [-h | -help] [-long_help] [-fanins] [-fanouts] [-panel_name` *<name>* `] <cell_id>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fanins`	Report only the fanins of the cell
	`-fanouts`	Report only the fanouts of the cell
	`-panel_name` *<name>*	Sends the results to the panel and specifies the name of the new panel
	<cell_id>	Single cell id
Description	Creates a report of the connectivity for a cell.	
Example Usage	This command currently contains no example usage.	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.13. periph::report_cell_placement_reasons (::quartus::periph)

The following table displays information for the `periph::report_cell_placement_reasons` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_cell_placement_reasons [-h | -help] [-long_help] [-panel_name <name> ] <cell_id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	<cell_id>		Single cell id
Description	`Creates a report of all the locations a particular cell can be placed and the reasons it cannot be placed there`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.14. periph::report_cells (::quartus::periph)

The following table displays information for the `periph::report_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_cells [-h | -help] [-long_help] [-name <name> ] [-panel_name <name> ] [-placed] [-type <type> ] [-unplaced]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		Filter the list of placed cells specifying a name. Wildcards are supported.
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-placed`		Report the list of placed cells
	`-type <type>`		Filter the list of placed cells specifying a list of types
	`-unplaced`		Report the list of unplaced cells
Description	`Returns a list of periphery cells based on the specified criteria.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.15. periph::report_clocks (::quartus::periph)

The following table displays information for the `periph::report_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_clocks [-h | -help] [-long_help] [-panel_name <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
Description	`Show the signals that are using low-skew routing networks (clock networks) in the device. If applicable, also show any signals that were considered for automatic clock network promotion, but were not promoted.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.16. periph::report_legal_cell_locations (::quartus::periph)

The following table displays information for the `periph::report_legal_cell_locations` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_legal_cell_locations [-h | -help] [-long_help] [-panel_name <name> ] <cell_id>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	<cell_id>		Single cell id
Description	`Creates a report of the legal periphery cell locations of a cell`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.17. periph::report_location_types (::quartus::periph)

The following table displays information for the `periph::report_location_types` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_location_types [-h | -help] [-long_help] [-panel_name <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
Description	`Creates a report of the location types in the periphery`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.18. periph::report_locations (::quartus::periph)

The following table displays information for the `periph::report_locations` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_locations [-h | -help] [-long_help] [-panel_name <name> ] <type>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	<type>		location type to query
Description	`Creates a report of the locations for the requested type in the periphery`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.19. periph::report_noc_performance (::quartus::periph)

The following table displays information for the `periph::report_noc_performance` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_noc_performance [-h | -help] [-long_help] [-iniu_freq <iniu_freq> ] [-panel_name <name> ] [-tniu_freq <tniu_freq> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-iniu_freq <iniu_freq>`		Specify the INIU frequency used in the report
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-tniu_freq <tniu_freq>`		Specify the TNIU frequency used in the report
Description	`Show the performance of the NoC given the current placement of the NoC elements.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.20. periph::report_regions (::quartus::periph)

The following table displays information for the `periph::report_regions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_regions [-h | -help] [-long_help] [-panel_name <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.21. periph::report_summary (::quartus::periph)

The following table displays information for the `periph::report_summary` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::report_summary [-h | -help] [-long_help] [-panel_name <name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.22. periph::reset_plan (::quartus::periph)

The following table displays information for the `periph::reset_plan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::reset_plan [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Reverts the current design to be unplaced and without assignments applied`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `periph::reset_plan` `blueprint::shutdown` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.23. periph::save_floorplan (::quartus::periph)

The following table displays information for the `periph::save_floorplan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::save_floorplan [-h | -help] [-long_help] -filename <filename>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-filename <filename>`	Filename to write to	
Description	`Write the Interface Planner floorplan that can be reloaded in Interface Planner`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `set io_cells [periph::get_cells -unplaced -type IO_CLUSTER]` `periph::place_cells -cells $io_cells` `periph::save_floorplan -filename onewire_blueprint_floorplan.plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.24. periph::set_clock_type (::quartus::periph)

The following table displays information for the `periph::set_clock_type` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::set_clock_type [-h | -help] [-long_help] [-cell_id <cell_id> ] [-cell_name <cell_name> ] -type <type>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-cell_id <cell_id>`	Single cell id	
	`-cell_name <cell_name>`	Cell name, returned from periph::get_cell_info -name	
	`-type <type>`	The type of the clock type	
Description	`This command currently contains no help description.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The supplied clock cell id <*string*> is invalid.

continued...

	TCL_ERROR	1	ERROR: The supplied clock type <*string*> is invalid.
	TCL_ERROR	1	ERROR: At least one clock type must be supplied, but no clock types were supplied
	TCL_ERROR	1	ERROR: <*string*> clock types were expected but <*string*> were supplied

3.1.25.25. blueprint::shutdown (::quartus::periph)

The following table displays information for the `blueprint::shutdown` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`blueprint::shutdown [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Shutdown Interface Planner.`		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `blueprint::shutdown` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.26. periph::undo_last_placement (::quartus::periph)

The following table displays information for the `periph::undo_last_placement` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::undo_last_placement [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Undo the last placement or unplacement operation`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.27. periph::unplace_cells (::quartus::periph)

The following table displays information for the `periph::unplace_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::unplace_cells [-h | -help] [-long_help] [-cells <cells> ] [-placed_cells]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-cells <cells>`	One or more cell ids	
	`-placed_cells`	Unplace all placed cells	
Description	Removes the placement from the specified cells. Any constraints for the cells remain, but the cell no longer has a placement.		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `periph::unplace_cells -placed_cells` `periph::check_plan` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.28. periph::update_pdpw (::quartus::periph)

The following table displays information for the `periph::update_pdpw` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347		
Syntax	`periph::update_pdpw [-h | -help] [-long_help] [-assignments] [-placement]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-assignments`	Indicates assignment model needs updating	
	`-placement`	Indicates placement needs updating	
Description	Update everything that needs updating in pdpw. This essentially sends a single TCL command to pdpw to update everything as needed. Used in the TCL proc source wrapper only.		
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_pdpw -pdp_state [blueprint_internal::get_pdp_state]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.29. periph::update_plan (::quartus::periph)

The following table displays information for the `periph::update_plan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::update_plan [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Applies all changes to constraints and reloads them into Interface Planner. After the platform has been updated with the constraints, placement operations can be performed.	
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `blueprint::shutdown` `project_close`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.25.30. periph::write_plan (::quartus::periph)

The following table displays information for the `periph::write_plan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::periph on page 347	
Syntax	`periph::write_plan [-h | -help] [-long_help] [-clocks] [-disabled] -filename <filename> [-force] [-other_locations] [-pin_locations]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clocks`	Write out clock assignments
	`-disabled`	Write out disabled assignments
	`-filename <filename>`	Filename to write to
	`-force`	Force the creation of the plan
	`-other_locations`	Write out other location assignments
	`-pin_locations`	Write out pin location assignments
Description	Export the floorplan constraints Tcl script	
Example Usage	`project_open onewire_nf` `blueprint::initialize` `periph::update_plan` `periph::place_cells -unplaced_cells` `periph::check_plan`	

continued...

	`` periph::write_plan -filename onewire_blueprint_assignments.tcl project_close ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.26. ::quartus::pfg

The following table displays information for the **::quartus::pfg** Tcl package:

Tcl Package and Version	::quartus::pfg 1.0
Description	``` This package contains the set of Tcl functions for using the Programming File Generator (PFG). ```
Availability	``` This package is loaded by default in the following executable: quartus_pfg ```
Tcl Commands	test (::quartus::pfg) on page 365

3.1.26.1. test (::quartus::pfg)

The following table displays information for the `test` Tcl command:

Tcl Package and Version	Belongs to ::quartus::pfg on page 365		
Syntax	`test [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	``` Dispaly general information ```		
Example Usage	``` test ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.27. ::quartus::proj_asgn

The following table displays information for the **::quartus::proj_asgn** Tcl package:

Tcl Package and Version	::quartus::proj_asgn 1.0
Description	``` This package contains the set of Tcl functions for making project-wide assignments. In versions before 4.0 of this package, the full path of the source file assignment was returned when you accessed the assignment through the "get_global_assignment" or "get_all_global_assignments" command. In version 4.0 of this package, the actual value of the source file assignment stored in the Quartus Prime Settings File (.qsf) is returned. To get the resolved full path of the file, use ```

continued...

	the "resolve_file_path" command. For more information about resolving file names and view an example, type "resolve_file_path -long_help". In version 5.0 of this package, two new Tcl commands "get_all_assignments" and "get_assignment_info" have been introduced to replace the following commands: get_all_quartus_defaults get_all_global_assignments get_all_instance_assignments get_all_parameters These two new commands simplify the interface to retrieve information about Quartus Prime Settings File (.qsf) and Quartus Prime Default Settings File (.qdf) assignments. In addition, the new "assignment_group" command replaces the deprecated "timegroup" command. In version 6.0, all Tcl commands designed to process Timing Analyzer assignments have been moved to the ::quartus::timing_assignment package.
Availability	This package is loaded by default in the following executables: qpro quartus
Tcl Commands	create_revision (::quartus::proj_asgn) on page 366 generate_project_tcl (::quartus::proj_asgn) on page 367 get_name_info (::quartus::proj_asgn) on page 368 get_names (::quartus::proj_asgn) on page 370 get_top_level_entity (::quartus::proj_asgn) on page 372 is_fitter_in_qhd_mode (::quartus::proj_asgn) on page 373

3.1.27.1. create_revision (::quartus::proj_asgn)

The following table displays information for the `create_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::proj_asgn on page 365	
Syntax	`create_revision [-h | -help] [-long_help] [-based_on` *<revision_name>* `] [-copy_results] [-new_rev_type` *<revision_type>* `] [-root_partition_qdb_file` *<qdb_file>* `] [-set_current]` *<revision_name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-based_on` *<revision_name>*	Revision name on which new revision bases its settings
	`-copy_results`	Option to copy results from "based_on" revision
	`-new_rev_type` *<revision_type>*	The type of the newly created revision
	`-root_partition_qdb_file` *<qdb_file>*	The Partition Database (.qdb) file for the root partition
	`-set_current`	Option to set new revision as current revision
	<revision_name>	Revision name
Description	Creates the specified revision. If the revision is not included in the current project, a new revision is created in the project with default settings. If you specify the "-set_current" option, this command sets the newly created revision as the current revision.	

continued...

	If you specify the "-based_on" option, the command creates a new revision in the project based on the settings of the based-on revision specified by the option.
Example Usage	```## Create a new revision called "tmp" create_revision tmp ## Create a new revision called "tmp" ## and set it as the current revision create_revision tmp -set_current ## This method is the same as create_revision tmp set_current_revision tmp ## Create a new revision called "speed_ch" ## with settings based on "chiptrip" ## and set it as the current revision create_revision speed_ch -based_on chiptrip -set_current```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	WARNING: Revision is already the current revision: <*string*>. No action is required.
TCL_ERROR	1	ERROR: Based-on revision is not included in the current project: <*string*>. Make sure the based-on revision name is spelled correctly and included in the current project.
TCL_ERROR	1	ERROR: Can't create revision because the current project uses the device family: <*string*>. Change the device family or create the revision in another project that uses a different device family.
TCL_ERROR	1	ERROR: Can't create file: <*string*>. Make sure you have permission to write to the specified file.
TCL_ERROR	1	ERROR: Unable to create a new 'Persona Implementation' revision based on an existing 'Persona Implementation' revision. Specify another revision type or change the base revision.
TCL_ERROR	1	ERROR: Can't create revision: <*string*>. Specify a legal revision name.
TCL_ERROR	1	ERROR: Can't remove file: <*string*>. Make sure the file is not read-only and you have permission to write to the specified file.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Didn't create revision because it is already included in current project: <*string*>. If you want a new revision, specify a different revision name.

3.1.27.2. generate_project_tcl (::quartus::proj_asgn)

The following table displays information for the `generate_project_tcl` Tcl command:

Tcl Package and Version	Belongs to ::quartus::proj_asgn on page 365	
Syntax	`generate_project_tcl [-h | -help] [-long_help] -filename <filename> [-include_default_assignments] [-overwrite]`	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`		Long help with examples and possible return values
	`-filename <filename>`		Tcl Filename
	`-include_default_assignments`		Option to include default assignments in the tcl file.
	`-overwrite`		Option to overwrite an existing tcl file
Description			
	``With currently opened, write a tcl file that can create the project and populate it current set of assignments in the qsf.```		
Example Usage			
	``## Generate a tcl file to create the current project. generate_project_tcl -filename create_project.tcl ## Generate a tcl file to create the current project and ## also include the default assignments. generate_project_tcl -filename create_project.tcl -include_default_assignments```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Failed when attempting to read qsf file.
	TCL_ERROR	1	ERROR: Failed when attempting to write tcl file.
	TCL_ERROR	1	ERROR: Project is not open, and path and project name was not specified. Open an existing project, or specify the path and project name in the command.
	TCL_ERROR	1	ERROR: Tcl file already exist. Please use -overwrite option to overwrite

3.1.27.3. get_name_info (::quartus::proj_asgn)

The following table displays information for the `get_name_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::proj_asgn on page 365	
Syntax	`get_name_info [-h | -help] [-long_help] [-info <parent_name_id|base_name|entity_name|entity_definition|instance_name|full_path|short_full_path|node_type|creator|signaltapii|file_location|library|children|parameters>] [-observable_type <all|pre_synthesis|post_synthesis|post_fitter|post_asm|stp_pre_synthesis>] <name_id>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-info <parent_name_id|base_name|entity_name|entity_definition|instance_name|full_path|short_full_path|node_type|creator|signaltapii|file_location|library|children|parameters>`	Option to specify the type of information to display.
	`-observable_type <all|pre_synthesis|post_synthesis|post_fitter|post_asm|stp_pre_synthesis>`	Option to specify the observable type of the name ID
	`<name_id>`	Option to specify the node name ID
Description		
	``Displays the specified type of information for the specified node name id. Type "get_names -long_help" to view how to get a collection of node name IDs.```	

continued...

```
If the "-observable_type" option is not specified, the default
value is "all". The specified observable type must have the same
observable type as specified in the "get_names" Tcl command which
returned the currently specified node name id.

The value for "-observable_type" option can be one of
the following:

Observable Type      Description
---------------      ---------------------------------
all                  Use post-Fitter information. If it is
                     not available, post-synthesis
                     information is used. Otherwise,
                     pre-synthesis information is used if it
                     exists.
pre_synthesis        Use pre-synthesis information.
post_synthesis       Use post-synthesis information.
post_fitter          Use post-Fitter information.
post_asm             Use post-Assembler information. The
                     post-Assembler information is only supported
                     for designs using the HardCopy II device family.
stp_pre_synthesis    Use Signal Tap pre-synthesis information.

The info type for the "-info" option can be one of
the following:

Info Type            Description
--------------       ---------------------------------
parent_name_id       The name id for the node's parent.
base_name            The node name, which consists of an
                     entity name and/or an instance name
                     separated by a colon if necessary.
entity_name          The entity name.
entity_definition    The entity definition.
instance_name        The instance name.
full_path            The full hierarchy path name, which
                     consists of entity name(s) and/or the
                     instance name(s). This path name
                     excludes the current focus entity.
                     If there is nothing shown, the
                     name id is the current focus entity's
                     name id.
short_full_path      The short full hierarchy path name,
                     which consists of the instance name(s).
                     This path name excludes the current
                     focus entity. If nothing is
                     shown, the name id is the
                     current focus entity's name id.
node_type            The node type, which can be one of
                     the types supported by "get_names",
                     namely, "input", "output", "bidirectional",
                     "register", "combinational",
                     "hierarchy", "memory", or "bus".
                     If "pin" type was specified for
                     "get_names" command, the node type shown
                     here is expanded to be "input", "output", or
                     "bidirectional".
                     Node type value of "qsf" indicates name
                     originates from qsf settings file.
creator              The creator of the node, which is either
                     "user_entered" or "compiler_generated".
signaltapii          If this node can be connected to a
                     Signal Tap embedded logic analyzer,
                     1 is shown. Otherwise, 0 is shown.
file_location        The source file location. For example,
                     the source file location for the entity
                     chiptrip is "chiptrip.v". To get the full
                     path to the source file, use the command
                     "resolve_file_path" which exists only in
                     version 4.0 or later of ::quartus::project
                     package.
library              Library associated with the instance name.
children             Collection of all the children names of the
             specified name. The children will include all
             the names in the specified hierarchy.
parameters           Collection of parameters associated with the name.
             Each element of the collection is a triplet that contains
             the name, value and the type of the parameter.
```

Example Usage

```
# Get the name id of the current focus entity
set current_focus_entity_id [get_top_level_entity]

# The full path name of the current focus entity
# is empty because the full path excludes the
# current focus entity
set msg "Full path of the current focus entity => ("
```

continued...

```
append msg [get_name_info -info full_path $current_focus_entity_id]
append msg ")"
puts $msg
puts ""

# Get the node type of the current focus entity
# The node type should be a hierarchy type
set msg "Node type of the current focus entity => ("
append msg [get_name_info -info node_type $current_focus_entity_id]
append msg ")"
puts $msg
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <*string*>. At the minimum, run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
	TCL_ERROR	1	ERROR: Illegal info type: <*string*>. Specify parent_name_id, base_name, entity_name, instance_name, full_path, short_full_path, node_type, creator, or signaltapii.
	TCL_ERROR	1	ERROR: Illegal name id: <*string*>. Specify a name id that exists in a compiled Quartus Prime project.
	TCL_ERROR	1	ERROR: Invalid name id for top level entity: <*string*>. Specify a valid top level entity id value. In Quartus Prime Pro, get_name_info query through integer value is only supported for get_top_level_entity TCL command. For other names query, please refer to get_names TCL command.
	TCL_ERROR	1	ERROR: Invalid name id: <*string*>. Specify an integer greater than or equal to zero.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Post-Assembler compiler database does not exist for revision name: <*string*>. Run Assembler (quartus_asm) with the specified revision name before using this Tcl command.

3.1.27.4. get_names (::quartus::proj_asgn)

The following table displays information for the `get_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::proj_asgn on page 365	
Syntax	`get_names [-h | -help] [-long_help] [-entity` <*wildcard*> `] -filter` <*wildcard*> `[-library` <*wildcard*> `] [-node_type` <*all*|*comb*|*reg*|*pin*|*input*|*output*|*bidir*|*hierarchy*|*mem*|*bus*|*qsf*|*state_machine*|*assigned*|*unassigned*|*all_reg*|*partition*|*virtual*> `] [-observable_type` <*all*|*pre_synthesis*|*post_synthesis*|*post_fitter*|*post_asm*|*stp_pre_synthesis*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity` <*wildcard*>	Option to specify the entity to get names from hierarchies instantiated by the entity
	`-filter` <*wildcard*>	Option to specify the node's full path name and/or wildcard character(s)
	`-library` <*wildcard*>	Option to specify the containing library

continued...

	`-node_type` *<all|comb|reg|pin|input|output|bidir|hierarchy|mem|bus|qsf|state_machine|assigned|unassigned|all_reg|partition|virtual>*	Option to filter based on the specified node type.
	`-observable_type` *<all|pre_synthesis|post_synthesis|post_fitter|post_asm|stp_pre_synthesis>*	Option to filter based on the specified observable type

Description

```
Returns a filtered output collection of all matching
node name IDs found in a compiled Quartus Prime project.

To access each element of the output collection, use the Tcl
command "foreach_in_collection". To see example usage, type
"get_names -long_help" or "foreach_in_collection -long_help".

If the "-node_type" option is not specified, the default
value is "all". Similarly, if the "-observable_type" option
is not specified, the default value is "all".

The node type "pin" includes "input", "output", "bidir", "assinged" and "unassigned".
The node type "qsf" include names from qsf settings file.
The node type "all" includes all node types.
The node type "all_reg" includes all node types and register post-fitting

The value for "-observable_type" option can be one of
the following:

Observable Type      Description
---------------      -----------------------------------
all                  Use post-Fitter information. If it is
                     not available, post-Synthesis
                     information is used. Otherwise,
                     pre-synthesis information is used if it
                     exists.
pre_synthesis        Use pre-synthesis information.
post_synthesis       Use post-synthesis information.
post_fitter          Use post-Fitter information.
post_asm             Use post-Assembler information. The
                     post-Assembler information is only supported
                     for designs using the HardCopy II device family.
stp_pre_synthesis    Use Signal Tap pre-synthesis information.
```

Example Usage

```
# Search for a single post-Fitter pin with the name accel and
# make assignments
set accel_name_id [get_names -filter accel -node_type pin -observable_type post_fitter]
foreach_in_collection name_id $accel_name_id {

    # Get the full path name of the node
    set target [get_name_info -info full_path $name_id]

    # Set multicycle assignment
    set_multicycle_assignment -to $target 2

    # Set location assignment
    set_location_assignment -to $target Pin_E22
}
# Search for nodes of any post-Fitter node type with name length <= 5
# The default node type is "all"
set name_ids [get_names -filter ????? -observable_type post_fitter]
foreach_in_collection name_id $name_ids {

    # Print the name id
    puts $name_id

    # Print the node type
    puts [get_name_info -info node_type $name_id]

    # Print the full path (which excludes the current
    # focus entity from the path)
    puts [get_name_info -info full_path $name_id]
}
# Search for nodes of any post-Fitter node type that end in "eed".
# The default node type is "all"
set name_ids [get_names -filter *eed -observable_type post_fitter]
foreach_in_collection name_id $name_ids {

    # Print the name id
    puts $name_id

    # Print the node type
    puts [get_name_info -info node_type $name_id]
```

continued...

```
    # Print the full path (which excludes the current
    # focus entity from the path)
    puts [get_name_info -info full_path $name_id]
}
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Get names cannot return <*string*> because the name was found in a partition that's not the root partition. Refine your get_names search pattern to exclude child partitions
	TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <*string*>. At the minimum, run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
	TCL_ERROR	1	ERROR: Illegal node type: <*string*>. Specify all, comb, reg, pin, hierarchy, or bus.
	TCL_ERROR	1	ERROR: Illegal observable type: <*string*>. Specify all, pre_synthesis, post_synthesis, or post_fitter.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Post-Assembler compiler database does not exist for revision name: <*string*>. Run Assembler (quartus_asm) with the specified revision name before using this Tcl command.

3.1.27.5. get_top_level_entity (::quartus::proj_asgn)

The following table displays information for the `get_top_level_entity` Tcl command:

Tcl Package and Version	Belongs to ::quartus::proj_asgn on page 365	
Syntax	`get_top_level_entity [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Returns the name id for the current focus entity.`	
Example Usage	`# Get the name id of the current focus entity` `set current_focus_entity_id [get_top_level_entity]` `# Print out the entity name of the current focus entity` `set msg "Entity name of the current focus entity => ("` `append msg [get_name_info -info entity_name $current_focus_entity_id]` `append msg ")"` `puts ""` `puts $msg`	

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <*string*>. At the minimum, run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.27.6. is_fitter_in_qhd_mode (::quartus::proj_asgn)

The following table displays information for the `is_fitter_in_qhd_mode` Tcl command:

Tcl Package and Version	Belongs to ::quartus::proj_asgn on page 365		
Syntax	`is_fitter_in_qhd_mode [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Returns true if we're running in QHD mode`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.28. ::quartus::project

The following table displays information for the **::quartus::project** Tcl package:

Tcl Package and Version	::quartus::project 7.0
Description	This package contains the set of Tcl functions for making project-wide assignments. In versions before 4.0 of this package, the full path of the source file assignment was returned when you accessed the assignment through the "get_global_assignment" or "get_all_global_assignments" command. In version 4.0 of this package, the actual value of the source file assignment stored in the Quartus Prime Settings File (.qsf) is returned. To get the resolved full path of the file, use the "resolve_file_path" command. For more information about resolving file names and view an example, type "resolve_file_path -long_help". In version 5.0 of this package, two new Tcl commands "get_all_assignments" and "get_assignment_info" have been introduced to replace the following commands: get_all_quartus_defaults get_all_global_assignments get_all_instance_assignments get_all_parameters These two new commands simplify the interface to retrieve information about Quartus Prime Settings File (.qsf) and Quartus Prime Default Settings File (.qdf) assignments. In addition, the new "assignment_group" command replaces the deprecated "timegroup" command.

continued...

	In version 6.0, all Tcl commands designed to process Timing Analyzer assignments have been moved to the ::quartus::timing_assignment package.
Availability	This package is loaded by default in the following executables: hdb_debug qpro_sh quartus_asm quartus_bpps quartus_cdb quartus_design quartus_eda quartus_fit quartus_idb quartus_ipd quartus_ipgenerate quartus_map quartus_sh quartus_si quartus_sim quartus_sta quartus_stp quartus_syn quartus_tlg
Tcl Commands	close_side_revision (::quartus::project) on page 375 create_revision (::quartus::project) on page 375 delete_revision (::quartus::project) on page 377 execute_assignment_batch (::quartus::project) on page 377 export_assignments (::quartus::project) on page 378 generate_project_tcl (::quartus::project) on page 379 get_all_assignment_names (::quartus::project) on page 380 get_all_assignments (::quartus::project) on page 381 get_all_global_assignments (::quartus::project) on page 384 get_all_instance_assignments (::quartus::project) on page 386 get_all_parameters (::quartus::project) on page 389 get_all_quartus_defaults (::quartus::project) on page 391 get_all_user_option_names (::quartus::project) on page 393 get_assignment_info (::quartus::project) on page 393 get_assignment_name_info (::quartus::project) on page 394 get_current_project (::quartus::project) on page 395 get_current_revision (::quartus::project) on page 395 get_database_version (::quartus::project) on page 396 get_global_assignment (::quartus::project) on page 396 get_instance_assignment (::quartus::project) on page 398 get_location_assignment (::quartus::project) on page 399 get_name_info (::quartus::project) on page 400 get_names (::quartus::project) on page 402 get_parameter (::quartus::project) on page 404 get_project_directory (::quartus::project) on page 405 get_project_revisions (::quartus::project) on page 405 get_revision_description (::quartus::project) on page 406 get_top_level_entity (::quartus::project) on page 407 get_user_option (::quartus::project) on page 407 is_database_version_compatible (::quartus::project) on page 408 is_fitter_in_qhd_mode (::quartus::project) on page 408 is_project_open (::quartus::project) on page 409 open_side_revision (::quartus::project) on page 409 project_archive (::quartus::project) on page 410 project_clean (::quartus::project) on page 411 project_close (::quartus::project) on page 412 project_exists (::quartus::project) on page 413 project_new (::quartus::project) on page 413 project_open (::quartus::project) on page 415 project_restore (::quartus::project) on page 416 remove_all_global_assignments (::quartus::project) on page 417 remove_all_instance_assignments (::quartus::project) on page 419 remove_all_parameters (::quartus::project) on page 421 resolve_file_path (::quartus::project) on page 423 revision_exists (::quartus::project) on page 424 set_current_revision (::quartus::project) on page 425 set_global_assignment (::quartus::project) on page 425 set_high_effort_fmax_optimization_assignments (::quartus::project) on page 428 set_instance_assignment (::quartus::project) on page 428 set_io_assignment (::quartus::project) on page 430 set_location_assignment (::quartus::project) on page 432 set_parameter (::quartus::project) on page 433 set_power_file_assignment (::quartus::project) on page 435 set_revision_description (::quartus::project) on page 436 set_user_option (::quartus::project) on page 437 test_assignment_trait (::quartus::project) on page 437

3.1.28.1. close_side_revision (::quartus::project)

The following table displays information for the `close_side_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`close_side_revision [-h | -help] [-long_help] <revision_name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<revision_name>	Revision name	
Description	`Closes the specified revision name, if it has already been opened as a side revision. Closing the revision will causes any changed assignments to be written to disk, and must be done before the revision can be set again as the current revision.`		
Example Usage	`## Create and open "new_rev" as a side revision. Apply an assignment and close the revision.` `create_revision new_rev -based_on my_rev -copy_results` `open_side_revision new_rev` `set_global_assignment -name OPTIMIZATION_TECHNIQUE "Area" -revision new_rev` `close_side_revision new_rev`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	WARNING: Revision is already the current revision: *<string>*. No action is required.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Revision file does not exist: *<string>*.qsf. Use delete_revision to delete the revision from the current project. Then use create_revision to create the revision and its .qsf before setting *<string>* as the current revision.
	TCL_ERROR	1	ERROR: Revision is not included in the current project: *<string>* . Use the create_revision command to create the revision.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.2. create_revision (::quartus::project)

The following table displays information for the `create_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`create_revision [-h | -help] [-long_help] [-based_on <revision_name> ] [-copy_results] [-new_rev_type <revision_type> ] [-root_partition_qdb_file <qdb_file> ] [-set_current] <revision_name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-based_on <revision_name>`	Revision name on which new revision bases its settings
	`-copy_results`	Option to copy results from "based_on" revision

continued...

	`-new_rev_type` <*revision_type*>	The type of the newly created revision
	`-root_partition_qdb_file` <*qdb_file*>	The Partition Database (.qdb) file for the root partition
	`-set_current`	Option to set new revision as current revision
	<*revision_name*>	Revision name

Description

```
Creates the specified revision. If the revision is not
included in the current project, a new revision is
created in the project with default settings.

If you specify the "-set_current" option, this command
sets the newly created revision as the current revision.

If you specify the "-based_on" option, the command creates
a new revision in the project based on the settings of the
based-on revision specified by the option.
```

Example Usage

```
## Create a new revision called "tmp"
create_revision tmp

## Create a new revision called "tmp"
## and set it as the current revision
create_revision tmp -set_current
## This method is the same as
create_revision tmp
set_current_revision tmp

## Create a new revision called "speed_ch"
## with settings based on "chiptrip"
## and set it as the current revision
create_revision speed_ch -based_on chiptrip -set_current
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	WARNING: Revision is already the current revision: <*string*>. No action is required.
TCL_ERROR	1	ERROR: Based-on revision is not included in the current project: <*string*>. Make sure the based-on revision name is spelled correctly and included in the current project.
TCL_ERROR	1	ERROR: Can't create revision because the current project uses the device family: <*string*>. Change the device family or create the revision in another project that uses a different device family.
TCL_ERROR	1	ERROR: Can't create file: <*string*>. Make sure you have permission to write to the specified file.
TCL_ERROR	1	ERROR: Unable to create a new 'Persona Implementation' revision based on an existing 'Persona Implementation' revision. Specify another revision type or change the base revision.
TCL_ERROR	1	ERROR: Can't create revision: <*string*>. Specify a legal revision name.
TCL_ERROR	1	ERROR: Can't remove file: <*string*>. Make sure the file is not read-only and you have permission to write to the specified file.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Didn't create revision because it is already included in current project: <*string*>. If you want a new revision, specify a different revision name.

3.1.28.3. delete_revision (::quartus::project)

The following table displays information for the `delete_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`delete_revision [-h | -help] [-long_help] [-remove_results] <revision_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-remove_results`		Option to delete the database files
	<revision_name>		Revision name
Description	`Deletes the specified revision from the current project. The corresponding <revision name>.qsf file is deleted as well.`		
Example Usage	`## Delete the revision called "tmp"` `delete_revision tmp`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't delete the current revision: *<string>*. Specify a different revision name.
	TCL_ERROR	1	ERROR: Can't delete revision because it is not included in the current project: *<string>* . Specify a revision name that is included in the project.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.4. execute_assignment_batch (::quartus::project)

The following table displays information for the `execute_assignment_batch` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`execute_assignment_batch [-h | -help] [-long_help] <tcl commands>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<tcl commands>	Tcl list of Tcl commands
Description	`Iterates through the specified Tcl list of Tcl commands and executes each command sequentially in batch mode.` `In batch mode, Tcl commands that set Quartus Prime Settings File (.qsf) assignments are optimized to prevent them from repeatedly write-locking and write-unlocking the QSF during consecutive calls, thereby slowing down the execution.` `Currently, only the following commands are supported:` `assignment_group` `remove_all_global_assignments` `remove_all_instance_assignments` `remove_all_parameters` `set_global_assignment`	

continued...

```
    set_instance_assignment
    set_io_assignment
    set_location_assignment
    set_parameter
    set_power_file_assignment
```

Example Usage

```
project_open one_wire
set tcl_cmds [list [list set_global_assignment -name FAMILY StratixII] \
                   [list set_global_assignment -name DEVICE AUTO] \
                   [list set_global_assignment -name TOP_LEVEL_ENTITY one_wire] \
                   [list set_global_assignment -name SAVE_DISK_SPACE OFF] \
                   [list set_location_assignment PIN_1 -to in1] \
                   [list set_instance_assignment -name MULTICYCLE 4 -from in1 -to out1] \
                   [list set_parameter -name STYLE FAST]]
execute_assignment_batch $tcl_cmds
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Unsupported Tcl command: <*string*>. Specify one of the supported Tcl commands listed in the help description for <*string*> -h.

3.1.28.5. export_assignments (::quartus::project)

The following table displays information for the `export_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`export_assignments [-h | -help] [-long_help] [-reorganize] [-report_write_failure]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-reorganize`	Option to reorganize the Quartus Prime Settings File (.qsf)
	`-report_write_failure`	Option to report error if write fail

Description

```
Exports assignments for the current revision to the Quartus Prime
Settings File (.qsf).

Assignments created or modified during an open project are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## The most common use of export_assignments is to
## call it before doing a system call
## to call a compiler command-line executable
project_open $project_name
set_global_assignment -name FAMILY Stratix

## Before calling quartus_map,
## write out the FAMILY assignment
export_assignments

## Now, call quartus_map
qexec "[file join $::quartus(binpath) quartus_map] $project_name"
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Can't write settings file *.qsf. Make sure the *.qsf file is writeable.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.6. generate_project_tcl (::quartus::project)

The following table displays information for the `generate_project_tcl` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`generate_project_tcl [-h | -help] [-long_help] -filename <filename> [-include_default_assignments] [-overwrite]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-filename <filename>`	Tcl Filename	
	`-include_default_assignments`	Option to include default assignments in the tcl file.	
	`-overwrite`	Option to overwrite an existing tcl file	
Description	`With currently opened, write a tcl file that can create the project and populate it current set of assignments in the qsf.`		
Example Usage	`## Generate a tcl file to create the current project.` `generate_project_tcl -filename create_project.tcl` `## Generate a tcl file to create the current project and` `## also include the default assignments.` `generate_project_tcl -filename create_project.tcl -include_default_assignments`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Failed when attempting to read qsf file.
	TCL_ERROR	1	ERROR: Failed when attempting to write tcl file.
	TCL_ERROR	1	ERROR: Project is not open, and path and project name was not specified. Open an existing project, or specify the path and project name in the command.
	TCL_ERROR	1	ERROR: Tcl file already exist. Please use -overwrite option to overwrite

3.1.28.7. get_all_assignment_names (::quartus::project)

The following table displays information for the `get_all_assignment_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_all_assignment_names [-h | -help] [-long_help] [-family <family> ] [-module <all|ip_generate|map|tlg|fit|tan|asm|eda|drc|power|generic> ] [-type <all|global|instance> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-family <family>`		Option to filter based on the specified device family. Defaults to all families.
	`-module <all|ip_generate|map|tlg|fit|tan|asm|eda|drc|power|generic>`		Option to filter based on the specified flow module. Defaults to all.
	`-type <all|global|instance>`		Option to filter based on the specified assignment type. Defaults to all.
Description			
Example Usage			
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Assignment <*string*> is not supported in this edition of the Quartus Prime software.

```
Returns a filtered output list of all available, matching
assignment names.

The module option takes one of the following values:

Module           Description
--------         -----------------------------------
ip_generate      IP Generation assignment names
tlg              Support Logic Generation assignment names
map              Analysis & Synthesis assignment names
fit              Fitter assignment names
asm              Assembler assignment names
eda              EDA Netlist Writer assignment names
drc              Design Assistant assignment names
power            Power Analyzer assignment names
generic          Other assignment names not included in any
                 of the above flow modules
all              All assignment names
```

```
## Print out all available global assignments
foreach i [get_all_assignment_names -type global] {
    puts $i
}

## Print out all available global assignments
## for the Stratix family
foreach i [get_all_assignment_names -type global -family Stratix] {
    puts $i
}

## Print out all available global assignments
## for the Stratix family required
## by the Analysis & Synthesis module
foreach i [get_all_assignment_names -type global -family Stratix -module map] {
    puts $i
}
```

continued...

	TCL_ERROR	1	ERROR: Illegal flow module: <*string*>. Specify <*string*>, <*string*>, <*string*>, <*string*>, <*string*>, <*string*>, or <*string*>.
	TCL_ERROR	1	ERROR: Illegal type: <*string*>. Specify <*string*>, <*string*>, or <*string*>.
	TCL_ERROR	1	ERROR: Illegal device family: <*string*>. Specify a legal device family.

3.1.28.8. get_all_assignments (::quartus::project)

The following table displays information for the `get_all_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_all_assignments [-h | -help] [-long_help] [-entity` <*entity_name*> `] [-fall] [-from` <*source*> `] -name` <*name*> `[-rise] [-section_id` <*section id*> `] [-tag` <*data*> `] [-to` <*destination*> `] -type` <*global|instance|parameter|default*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity` <*entity_name*>	Entity name
	`-fall`	Option applies to falling edge
	`-from` <*source*>	Source name (string pattern is matched using Tcl string matching)
	`-name` <*name*>	Assignment name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-section_id` <*section id*>	Section id
	`-tag` <*data*>	Option to tag data to this assignment
	`-to` <*destination*>	Destination name (string pattern is matched using Tcl string matching)
	`-type` <*global|instance|parameter|default*>	Option to specify the type of assignments to return
Description		

```
Returns a collection of all matching global, instance, parameter,
or default assignment ids. To iterate through each assignment id
in this collection, use the Tcl command "foreach_in_collection".

To view details for the assignment that is associated with the
assignment id, use the Tcl command "get_assignment_info".

The "get_all_assignments" command is easier to use than the
deprecated commands listed in Table 1.

* Table 1. The -type Option

Value for
-type Option  Deprecated Tcl command        Description
------------  ---------------------        -----------
default       get_all_quartus_defaults     Returns only default assignments.

global        get_all_global_assignments   Returns only global assignments.

instance      get_all_instance_assignments Returns only instance assignments.

parameter      get_all_parameters           Returns only parameter assignments.

The "-name" option is not case sensitive.
```

continued...

```
The "-to" and "-from" options are case sensitive.

These options can take string patterns containing special
characters from the set "*?\[]" as values. The values are
matched using Tcl string matching. Note that bus names are
automatically detected and do not need to be escaped. Bus
names have the following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
get_all_assignments -type instance -name LOCATION -to a[0]

# To match all indices of bus "a", type:
get_all_assignments -type instance -name LOCATION -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

get_all_assignments -type instance LOCATION -to "a[escape_brackets \[]\[0-2\][escape_brackets
\]]"

For more information about escaping square brackets, type
"escape_brackets -h".

This Tcl command reads in the global, instance, and parameter
assignments found in the Quartus Prime Settings File (.qsf) and
reads in the default assignments found inside the
Quartus Prime Default Settings File (.qdf).

If you tagged data by making assignments with the -tag option,
then the information can be searched using the -tag option.

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one clock section
each containing its own set of clock assignments. To uniquely
identify sections of this type, use the -section_id option.

For entity-specific assignments, use the "-entity" option to
retrieve assignments from a specific entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.
```

Example Usage

```
    ## View all the timing requirements using wildcards
    ## to match TSU_REQUIREMENT, TCO_REQUIREMENT,
    ## and others.
foreach_in_collection asgn_id [get_all_assignments -type instance -name *_REQUIREMENT] {

    set from   [get_assignment_info $asgn_id -from]
    set to     [get_assignment_info $asgn_id -to]
    set name   [get_assignment_info $asgn_id -name]
    set value  [get_assignment_info $asgn_id -value]
    set entity [get_assignment_info $asgn_id -entity]
    set sid    [get_assignment_info $asgn_id -section_id]
    set tag    [get_assignment_info $asgn_id -tag]

    puts "$entity: $name ($from -> $to) = $value"
}

    ## View all global assignments
foreach_in_collection asgn_id [get_all_assignments -type global -name *] {

    set name   [get_assignment_info $asgn_id -name]
    set value  [get_assignment_info $asgn_id -value]
    set entity [get_assignment_info $asgn_id -entity]
    set sid    [get_assignment_info $asgn_id -section_id]
    set tag    [get_assignment_info $asgn_id -tag]

    puts "$entity: $name = $value"
}

    ## View all project-wide default parameter values
foreach_in_collection asgn_id [get_all_assignments -type parameter -name *] {

    set name  [get_assignment_info $asgn_id -name]
    set value [get_assignment_info $asgn_id -value]
```

continued...

```
    set tag    [get_assignment_info $asgn_id -tag]

    puts "$name = $value"
}

    ## View all entity-specific parameter values
foreach_in_collection asgn_id [get_all_assignments -type parameter -name * -to *] {

    set dest  [get_assignment_info $asgn_id -to]
    set name  [get_assignment_info $asgn_id -name]
    set value [get_assignment_info $asgn_id -value]
    set tag    [get_assignment_info $asgn_id -tag]

    puts "$name (-> $dest) = $value"
}

    ## View all default assignments
foreach_in_collection asgn_id [get_all_assignments -type default -name * -to *] {

    set name    [get_assignment_info $asgn_id -name]
    set value   [get_assignment_info $asgn_id -value]

    puts "$name = $value"
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Assignment <*string*> is not supported in this edition of the Quartus Prime software.
TCL_ERROR	1	ERROR: Assignment is not an instance assignment: <*string*> -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Illegal assignment type: <*string*>. Specify <*string*>, <*string*>, <*string*>, or <*string*>.
TCL_ERROR	1	ERROR: Illegal option <*string*>. The specified option is illegal for <*string*> assignments.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Ignored assignment: <*string*>. The assignment is no longer supported.

continued...

	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.9. get_all_global_assignments (::quartus::project)

The following table displays information for the `get_all_global_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_all_global_assignments [-h | -help] [-long_help] [-entity` <*entity_name*> `] [-fall] -name` <*name*> `[-rise] [-section_id` <*section id*> `] [-tag` <*data*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity` <*entity_name*>	Entity to which assignment belongs
	`-fall`	Option applies to falling edge
	`-name` <*name*>	Assignment name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-section_id` <*section id*>	Section id
	`-tag` <*data*>	Option to tag data to this assignment
Description		

```
Returns a filtered output collection of all matching global
assignment values. To access each element of the output
collection, use the Tcl command "foreach_in_collection". To
see example usage, type "foreach_in_collection -long_help".

In version 5.0 of the ::quartus::project package, two new
Tcl commands "get_all_assignments" and "get_assignment_info"
have been introduced to replace the "get_all_global_assignments"
command. These two new commands simplify the interface to retrieve
information about Quartus Prime Settings File (.qsf) assignments.
The "get_all_global_assignments" command is still supported for
backward compatibility.

The "-name" option is not case sensitive. This option can take
string patterns containing special characters from the set
"*?\[]" as the value. The value is matched using Tcl string
matching.

This Tcl command reads the global assignments found in
the Quartus Prime Settings File (.qsf). This Tcl command filters
the assignment data in the .qsf and outputs the data based
on the values given by the "-name" option.

Each element of the collection is a list with the following
format:
{ {<Section Id>} {<Assignment name>} {<Assignment value>} {<Entity name>} {<Tag data>} }

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                    Description
```

continued...

```
--------------------------   ------------------------------
CHIP                         Same as revision name
LOGICLOCK_REGION             A unique name
EDA_TOOL_SETTINGS            A unique name
CLIQUE                       A unique name
BREAKPOINT                   A unique name
CLOCK                        A unique name
AUTO_INSERT_SLD_NODE_ENTITY  A unique name

If you tagged data by making assignments with the -tag option,
then the information will be displayed in the <Tag data> field.

For entity-specific assignments, use the "-entity" option to
retrieve the assignment(s) from the specified entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.
```

Example Usage

```
## Print out all the registered source files
## using the foreach_in_collection method
set file_asgn_col [get_all_global_assignments -name SOURCE_FILE]
foreach_in_collection file_asgn $file_asgn_col {

    ## Each element in the collection has the following
    ## format: {} {SOURCE_FILE} {<file_name>} {} {}
    puts [lindex $file_asgn 2]
}

## Print out all global assignments
set asgn_col [get_all_global_assignments -name *]

foreach_in_collection asgn $asgn_col {

    ## Each element in the collection has the following
    ## format: { {} {<Assignment name>} {<Assignment value>} {<Entity name>} {<Tag data>} }
    set name   [lindex $asgn 1]
    set value  [lindex $asgn 2]
    set entity [lindex $asgn 3]
    set tag    [lindex $asgn 4]

    puts "$entity: $name = $value"
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Assignment <*string*> is not supported in this edition of the Quartus Prime software.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.

continued...

	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Ignored assignment: <*string*>. The assignment is no longer supported.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.10. get_all_instance_assignments (::quartus::project)

The following table displays information for the `get_all_instance_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_all_instance_assignments [-h | -help] [-long_help] [-entity <entity_name> ] [-fall] [-from <source> ] -name <name> [-rise] [-section_id <section id> ] [-tag <data> ] [-to <destination> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which assignment belongs
	`-fall`	Option applies to falling edge
	`-from <source>`	Source of assignment (string pattern is matched using Tcl string matching)
	`-name <name>`	Assignment name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-section_id <section id>`	Section id
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of assignment (string pattern is matched using Tcl string matching)
Description		

```
Returns a filtered output collection of all matching instance
assignment values. To access each element of this output
collection, use the Tcl command "foreach_in_collection". To
see example usage, type "foreach_in_collection -long_help".

In version 5.0 of the ::quartus::project package, two new
Tcl commands "get_all_assignments" and "get_assignment_info"
have been introduced to replace the "get_all_instance_assignments"
command. These two new commands simplify the interface to retrieve
information about Quartus Prime Settings File (.qsf) assignments.
The "get_all_instance_assignments" command is still supported
for backward compatibility.
```

continued...

```
The "-name" option is not case sensitive.
The "-to" and "-from" options are case sensitive.

These options can take string patterns containing special
characters from the set "*?\[]" as values. The values are
matched using Tcl string matching. Note that bus names are
automatically detected and do not need to be escaped. Bus
names have the following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
get_all_instance_assignments -name LOCATION -to a[0]

# To match all indices of bus "a", type:
get_all_instance_assignments -name LOCATION -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

get_all_instance_assignments -name LOCATION -to "a[escape_brackets \[]\[0-2\][escape_brackets
\]]"

For more information about escaping square brackets, type
"escape_brackets -h".

This Tcl command reads in the instance assignments found in
the Quartus Prime Settings File (.qsf). The command filters
the assignments data found in the .qsf and outputs the
data based on the values specified by the "-name", "-from",
and "-to" options.

Each element of the collection is a list with the following
format:
{ {<Section Id>} {<Source>} {<Destination>} {<Assignment name>} {<Assignment value>} {<Entity
name>} {<Tag data>} }

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                   Description
--------------------------   ------------------------------
CHIP                         Same as revision name
LOGICLOCK_REGION             A unique name
EDA_TOOL_SETTINGS            A unique name
CLIQUE                       A unique name
BREAKPOINT                   A unique name
CLOCK                        A unique name
AUTO_INSERT_SLD_NODE_ENTITY  A unique name

If you tagged data by making assignments with the -tag option,
then the information will be displayed in the <Tag data> field.

For entity-specific assignments, use the "-entity" option to
retrieve the assignment(s) from the specified entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.
```

Example Usage

```
## Print out all the timing requirements
## using the foreach_in_collection method.
## Use wildcards to catch TSU_REQUIREMENT, TCO_REQUIREMENT,
## and others.
set asgn_col [get_all_instance_assignments -name *_REQUIREMENT]

foreach_in_collection asgn $asgn_col {

    ## Each element in the collection has the following
    ## format: { {} {<Source>} {<Destination>} {<Assignment name>} {<Assignment value>}
{<Entity name>} {<Tag data>} }
    set from   [lindex $asgn 1]
```

continued...

```
    set to     [lindex $asgn 2]
    set name   [lindex $asgn 3]
    set value  [lindex $asgn 4]
    set entity [lindex $asgn 5]
    set tag    [lindex $asgn 6]

    puts "$entity: $name ($from -> $to) = $value"
}

## Get all the location assignments with
## the destination bus name "timeo".
set bus_name "timeo"
set location_asgns [get_all_instance_assignments -name LOCATION -to $bus_name[*]]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Assignment is not an instance assignment: <*string*> -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.11. get_all_parameters (::quartus::project)

The following table displays information for the `get_all_parameters` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_all_parameters [-h | -help] [-long_help] [-entity <entity_name> ] [-fall] -name <name> [-rise] [-tag <data> ] [-to <destination> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which parameter belongs
	`-fall`	Option applies to falling edge
	`-name <name>`	Parameter name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of the parameter (string pattern is matched using Tcl string matching)
Description		

```
Returns a filtered output collection of all matching parameter
values. To access each element of this output collection, use
the Tcl command "foreach_in_collection". To see example usage,
type "foreach_in_collection -long_help".

In version 5.0 of ::quartus::project package, two new
Tcl commands "get_all_assignments" and "get_assignment_info"
have been introduced to replace the "get_all_parameters"
command. These two new commands simplify the interface to retrieve
information about Quartus Prime Settings File (.qsf) assignments.
The "get_all_parameters" command is still supported for backward
compatibility.

The "-name" option is not case sensitive.
The "-to" option is case sensitive.

If the "-to" argument is specified, the function returns
the parameter values for the current entity. The values are
retrieved from the PARAMETERS section of the entity. Otherwise,
the function returns the project-wide default parameter values
obtained from the DEFAULT_PARAMETERS section.

This Tcl command filters the parameter data found in the
Quartus Prime Settings File (.qsf) and outputs the data based on
the values specified by the "-name" and "-to" options. These
options can take string patterns containing special characters
from the set "*?\[]" as values. The values are matched using
Tcl string matching. Note that bus names are automatically
detected and do not need to be escaped. Bus names have the
following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
get_all_parameters -name * -to a[0]

# To match all indices of bus "a", type:
get_all_parameters -name * -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

get_all_parameters -name * -to "a[escape_brackets \[]\[0-2\][escape_brackets \]]"
```

continued...

```
For more information about escaping square brackets, type
"escape_brackets -h".

Each element of the collection is a list with the following
format:
{ {<Destination>} {<Parameter name>} {<Parameter value>} {<Entity name>} {<Tag data>} }

If you tagged data by making assignments with the -tag option,
then the information will be displayed in the <Tag data> field.

Use the "-entity" option to retrieve the parameter values from
the specified entity. If the "-entity" option is not specified,
the value for the FOCUS_ENTITY_NAME assignment is used. If the
FOCUS_ENTITY_NAME value is not found, the revision name is used.
```

Example Usage

```
## Display all project-wide default parameter values
set parameter_col [get_all_parameters -name *]

foreach_in_collection parameter $parameter_col {

    ## Each element in the collection has the following
    ## format: { {} {<Parameter name>} {<Parameter value>} {} {} }
    set name [lindex $parameter 1]
    set value [lindex $parameter 2]

    ## Now, display the content of the parameter
    puts "Parameter Name ($name)"
    puts "Parameter Value ($value)"
}

## Display all entity-specific parameter values
foreach_in_collection parameter [get_all_parameters -name * -to *] {

    ## Each element in the collection has the following
    ## format: { {Destination} {<Parameter name>} {<Parameter value>} {} {} }
    set dest [lindex $parameter 0]
    set name [lindex $parameter 1]
    set value [lindex $parameter 2]

    ## Now, display the content of the parameter
    puts "Destination ($dest)"
    puts "Parameter Name ($name)"
    puts "Parameter Value ($value)"
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Illegal default parameter: <*string*>. Specify a legal default parameter name.
TCL_ERROR	1	ERROR: Illegal parameter: <*string*>. Specify a legal parameter name.
TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.12. get_all_quartus_defaults (::quartus::project)

The following table displays information for the `get_all_quartus_defaults` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_all_quartus_defaults [-h | -help] [-long_help] [-fall] [-name <name> ] [-rise] [-section_id <section id> ] [-tag <data> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fall`	Option applies to falling edge
	`-name <name>`	Assignment name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-section_id <section id>`	Section id
	`-tag <data>`	Option to tag data to this assignment
Description		

```
Returns a filtered output collection of all matching default
assignment values. To access each element of the output
collection, use the Tcl command "foreach_in_collection". To
see example usage, type "foreach_in_collection -long_help".

In version 5.0 of ::quartus::project package, two new
Tcl commands "get_all_assignments" and "get_assignment_info"
have been introduced to replace the "get_all_quartus_defaults"
command. These two new commands simplify the interface to retrieve
information about Quartus Prime Settings File (.qsf) assignments.
The "get_all_quartus_defaults" command is still supported
for backward compatibility.

The "-name" option is not case sensitive. This option can take
string patterns containing special characters from the set
"*?\[]" as the value. The value is matched using Tcl string
matching.

This Tcl command reads in the default assignments found inside
the Quartus Prime Default Settings File (.qdf). It filters the
assignments data found inside the .qdf and outputs the data
based on the values specified by the "-name" option.

Each element of the collection is a list with the following
format:
{ {<Section Id>} {<Assignment name>} {<Assignment value>} }

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                    Description
---------------------------   -----------------------------
CHIP                          Same as revision name
LOGICLOCK_REGION              A unique name
EDA_TOOL_SETTINGS             A unique name
CLIQUE                        A unique name
BREAKPOINT                    A unique name
CLOCK                         A unique name
AUTO_INSERT_SLD_NODE_ENTITY   A unique name
```

Example Usage

```
## Print out all the default assignments using
## the foreach_in_collection method

set default_asgns_col [get_all_quartus_defaults]
foreach_in_collection default $default_asgns_col {
```

continued...

```
    set sect_id [lindex $default 0]
    set name [lindex $default 1]
    set value [lindex $default 2]

    ## Now, display the content of the assignment
    puts "Section ID ($sect_id)"
    puts "Assignment Name ($name)"
    puts "Assignment Value ($value)"
}

## Using wildcards
set default_asgns_col [get_all_quartus_defaults -name *]
foreach_in_collection default $default_asgns_col {
    set sect_id [lindex $default 0]
    set name [lindex $default 1]
    set value [lindex $default 2]

    ## Now, display the content of the assignment
    puts "Section ID ($sect_id)"
    puts "Assignment Name ($name)"
    puts "Assignment Value ($value)"
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.13. get_all_user_option_names (::quartus::project)

The following table displays information for the `get_all_user_option_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_all_user_option_names [-h | -help] [-long_help] [-name <name> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <name>`	User option name (string pattern is matched using Tcl string matching)	
Description	`Returns a filtered output list of all available, matching user option names.` `If the "-name" option is not specified, all available user option names are returned. Otherwise, only the matching user option names are returned.` `The "-name" option is not case sensitive. This option can take string patterns containing special characters from the set "*?\[]" as the value. The value is matched using Tcl string matching.`		
Example Usage	`## Print out all available user option names` `foreach i [get_all_user_option_names] {` `  puts $i` `}` `## Display all user option names that contain` `## the word "talkback" and also display the` `## value for each of the user option names` `foreach i [get_all_user_option_names -name *talkback*] {` `  set name $i` `  set value [get_user_option -name $i]` `  puts "$name = $value"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.28.14. get_assignment_info (::quartus::project)

The following table displays information for the `get_assignment_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_assignment_info [-h | -help] [-long_help] [-comments] [-entity] [-from] [-get_tcl_command] [-name] [-section_id] [-tag] [-to] [-value] <asgn_id>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comments`	Option to get the assignment comment
	`-entity`	Option to get the assignment entity
	`-from`	Option to get the assignment source
	`-get_tcl_command`	Option to get the tcl command that sets the assignment

continued...

	`-name`		Option to get the assignment name
	`-section_id`		Option to get the assignment section id
	`-tag`		Option to get the assignment tag
	`-to`		Option to get the assignment destination
	`-value`		Option to get the assignment value
	<asgn_id>		Assignment id
Description	`Returns information for the assignment id based on the specified option.` `The assignment id is obtained from the "get_all_assignments" Tcl command.`		
Example Usage	`## View all the instance assignments` `foreach_in_collection asgn_id [get_all_assignments -type instance -name *] {` `set from [get_assignment_info $asgn_id -from]` `set to [get_assignment_info $asgn_id -to]` `set name [get_assignment_info $asgn_id -name]` `set value [get_assignment_info $asgn_id -value]` `set entity [get_assignment_info $asgn_id -entity]` `set tag [get_assignment_info $asgn_id -tag]` `puts "$entity: $name ($from -> $to) = $value"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal assignment id: *<string>*. Specify a legal assignment id that was retrieved from the Tcl command get_all_assignments.

3.1.28.15. get_assignment_name_info (::quartus::project)

The following table displays information for the `get_assignment_name_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_assignment_name_info [-h | -help] [-long_help] <name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<name>		Assignment name
Description	`Returns information for the specified assignment name.`		
Example Usage	`## View information for all assignment names` `foreach name [get_all_assignment_names] {` `puts [get_assignment_name_info $name]` `}`		
Return Value	**Code Name**	**Code**	**String Return**

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Assignment <*string*> is not supported in this edition of the Quartus Prime software.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.

3.1.28.16. get_current_project (::quartus::project)

The following table displays information for the `get_current_project` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_current_project [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns the name of the current project.`		
Example Usage	`# Get the current name for` `# the currently open project "chiptrip"` `project_open chiptrip` `set project_name [get_current_project]` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.28.17. get_current_revision (::quartus::project)

The following table displays information for the `get_current_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_current_revision [-h | -help] [-long_help] [ <project_name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<*project_name*>		Project name
Description	`Returns the name of the current revision for the specified` `project. If the project name is not specified, the current` `project name is used.`		
Example Usage	`# Get the current revision name for` `# the currently open project "chiptrip"` `project_open chiptrip` `set revision_name [get_current_revision]` `project_close` `# Get the current revision name for` `# a project that is not currently open` `set revision_name [get_current_revision chiptrip]`		

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: <*string*>. Specify a legal project name.
	TCL_ERROR	1	ERROR: Project name was not specified or open project does not exist. Open an existing project or specify the project name.

3.1.28.18. get_database_version (::quartus::project)

The following table displays information for the `get_database_version` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_database_version [-h | -help] [-long_help] <project_name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<*project_name*>	Project name	
Description	`Return the project's database version. This is the version of software that the database was created.`		
Example Usage	`## Print message on database version` `set db_version [get_database_version chiptrip]` `post_message "The database is created from \"$db_version\""`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The input project <*string*> is not found.

3.1.28.19. get_global_assignment (::quartus::project)

The following table displays information for the `get_global_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_global_assignment [-h | -help] [-long_help] [-entity <entity_name> ] [-fall] [-front] -name <name> [-rise] [-section_id <section id> ] [-tag <data> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which assignment belongs
	`-fall`	Option applies to falling edge
	`-front`	Option to return the first assignment if there is more than one assignment found
	`-name <name>`	Assignment name
	`-rise`	Option applies to rising edge

continued...

	`-section_id` <*section id*>	Section id
	`-tag` <*data*>	Option to tag data to this assignment
Description	`Returns the value of the global assignment.` `The "-name" option is not case sensitive.` `For entity-specific assignments, use the "-entity" option to retrieve the assignment from the specified entity. If the "-entity" option is not specified, the value for the FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME value is not found, the revision name is used.`	
Example Usage	`## Get the value of the FAMILY assignment` `get_global_assignment -name FAMILY`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Assignment <*string*> is not supported in this edition of the Quartus Prime software.
	TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
	TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Ignored assignment: <*string*>. The assignment is no longer supported.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.
	TCL_ERROR	1	ERROR: Assignment <*string*> has multiple values. Use the <*string*> command to get all values or use the <*string*> -front command to get the first value.

3.1.28.20. get_instance_assignment (::quartus::project)

The following table displays information for the `get_instance_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_instance_assignment [-h | -help] [-long_help] [-entity <entity_name> ] [-fall] [-from <source> ] [-front] -name <name> [-rise] [-section_id <section id> ] [-tag <data> ] [-to <destination> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-entity <entity_name>`	Entity to which assignment belongs	
	`-fall`	Option applies to falling edge	
	`-from <source>`	Source of assignment	
	`-front`	Option to return the first assignment if there is more than one assignment found	
	`-name <name>`	Assignment name	
	`-rise`	Option applies to rising edge	
	`-section_id <section id>`	Section id	
	`-tag <data>`	Option to tag data to this assignment	
	`-to <destination>`	Destination of assignment	
Description	`Returns the value of the instance assignment.` `The "-name" option is not case sensitive. The "-entity", "-to", and "-from" options are case sensitive.` `For entity-specific assignments, use the "-entity" option to retrieve the assignment from the specified entity. If the "-entity" option is not specified, the value for the FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME value is not found, the revision name is used.`		
Example Usage	`## Get the TSU_REQUIREMENT from mypin to any register` `set value [get_instance_assignment -from "mypin" -to * -name TSU_REQUIREMENT]` `puts "TSU_REQUIREMENT(mypin->*) = $value"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Assignment is not an instance assignment: *<string>* -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
	TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.

continued...

	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Ignored assignment: <*string*>. The assignment is no longer supported.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.
	TCL_ERROR	1	ERROR: Assignment <*string*> has multiple values. Use the <*string*> command to get all values or use the <*string*> -front command to get the first value.

3.1.28.21. get_location_assignment (::quartus::project)

The following table displays information for the `get_location_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_location_assignment [-h | -help] [-long_help] [-fall] [-rise] [-tag` <*data*> `] -to` <*destination*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-fall`		Option applies to falling edge
	`-rise`		Option applies to rising edge
	`-tag` <*data*>		Option to tag data to this assignment
	`-to` <*destination*>		Destination of assignment
Description	`Returns the value of a location assignment.` `The "-chip" option is not case sensitive.` `The "-to" option is case sensitive.`		
Example Usage	`get_location_assignment -to dst`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.22. get_name_info (::quartus::project)

The following table displays information for the `get_name_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_name_info [-h | -help] [-long_help] [-info` *<parent_name_id|base_name|entity_name|entity_definition|instance_name|full_path|short_full_path|node_type|creator|signaltapii|file_location|library|children|parameters>* `] [-observable_type` *<all|pre_synthesis|post_synthesis|post_fitter|post_asm|stp_pre_synthesis>* `]` *<name_id>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-info` *<parent_name_id|base_name|entity_name|entity_definition|instance_name|full_path|short_full_path|node_type|creator|signaltapii|file_location|library|children|parameters>*	Option to specify the type of information to display.
	`-observable_type` *<all|pre_synthesis|post_synthesis|post_fitter|post_asm|stp_pre_synthesis>*	Option to specify the observable type of the name ID
	<name_id>	Option to specify the node name ID
Description		

```
Displays the specified type of information for the specified node
name id. Type "get_names -long_help" to view how to get a collection
of node name IDs.

If the "-observable_type" option is not specified, the default
value is "all". The specified observable type must have the same
observable type as specified in the "get_names" Tcl command which
returned the currently specified node name id.

The value for "-observable_type" option can be one of
the following:

Observable Type      Description
---------------      -----------------------------------
all                  Use post-Fitter information. If it is
                     not available, post-synthesis
                     information is used. Otherwise,
                     pre-synthesis information is used if it
                     exists.
pre_synthesis        Use pre-synthesis information.
post_synthesis       Use post-synthesis information.
post_fitter          Use post-Fitter information.
post_asm             Use post-Assembler information. The
                     post-Assembler information is only supported
                     for designs using the HardCopy II device family.
stp_pre_synthesis    Use Signal Tap pre-synthesis information.

The info type for the "-info" option can be one of
the following:

Info Type            Description
--------------       -----------------------------------
parent_name_id       The name id for the node's parent.
base_name            The node name, which consists of an
                     entity name and/or an instance name
```

continued...

```
                    separated by a colon if necessary.
entity_name         The entity name.
entity_definition   The entity definition.
instance_name       The instance name.
full_path           The full hierarchy path name, which
                    consists of entity name(s) and/or the
                    instance name(s). This path name
                    excludes the current focus entity.
                    If there is nothing shown, the
                    name id is the current focus entity's
                    name id.
short_full_path     The short full hierarchy path name,
                    which consists of the instance name(s).
                    This path name excludes the current
                    focus entity. If nothing is
                    shown, the name id is the
                    current focus entity's name id.
node_type           The node type, which can be one of
                    the types supported by "get_names",
                    namely, "input", "output", "bidirectional",
                    "register", "combinational",
                    "hierarchy", "memory", or "bus".
                    If "pin" type was specified for
                    "get_names" command, the node type shown
                    here is expanded to be "input", "output", or
                    "bidirectional".
                    Node type value of "qsf" indicates name
                    originates from qsf settings file.
creator             The creator of the node, which is either
                    "user_entered" or "compiler_generated".
signaltapii         If this node can be connected to a
                    Signal Tap embedded logic analyzer,
                    1 is shown. Otherwise, 0 is shown.
file_location       The source file location. For example,
                    the source file location for the entity
                    chiptrip is "chiptrip.v". To get the full
                    path to the source file, use the command
                    "resolve_file_path" which exists only in
                    version 4.0 or later of ::quartus::project
                    package.
library             Library associated with the instance name.
children            Collection of all the children names of the
            specified name. The children will include all
            the names in the specified hierarchy.
parameters          Collection of parameters associated with the name.
            Each element of the collection is a triplet that contains
            the name, value and the type of the parameter.
```

Example Usage

```
# Get the name id of the current focus entity
set current_focus_entity_id [get_top_level_entity]

# The full path name of the current focus entity
# is empty because the full path excludes the
# current focus entity
set msg "Full path of the current focus entity => ("
append msg [get_name_info -info full_path $current_focus_entity_id]
append msg ")"
puts $msg
puts ""

# Get the node type of the current focus entity
# The node type should be a hierarchy type
set msg "Node type of the current focus entity => ("
append msg [get_name_info -info node_type $current_focus_entity_id]
append msg ")"
puts $msg
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <*string*>. At the minimum, run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
TCL_ERROR	1	ERROR: Illegal info type: <*string*>. Specify parent_name_id, base_name, entity_name, instance_name, full_path, short_full_path, node_type, creator, or signaltapii.

continued...

	TCL_ERROR	1	ERROR: Illegal name id: <*string*>. Specify a name id that exists in a compiled Quartus Prime project.
	TCL_ERROR	1	ERROR: Invalid name id for top level entity: <*string*>. Specify a valid top level entity id value. In Quartus Prime Pro, get_name_info query through integer value is only supported for get_top_level_entity TCL command. For other names query, please refer to get_names TCL command.
	TCL_ERROR	1	ERROR: Invalid name id: <*string*>. Specify an integer greater than or equal to zero.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Post-Assembler compiler database does not exist for revision name: <*string*>. Run Assembler (quartus_asm) with the specified revision name before using this Tcl command.

3.1.28.23. get_names (::quartus::project)

The following table displays information for the `get_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_names [-h | -help] [-long_help] [-entity` <*wildcard*> `] -filter` <*wildcard*> `[-library` <*wildcard*> `] [-node_type` <*all|comb|reg|pin|input|output|bidir|hierarchy|mem|bus|qsf|state_machine|assigned|unassigned|all_reg|partition|virtual*> `] [-observable_type` <*all|pre_synthesis|post_synthesis|post_fitter|post_asm|stp_pre_synthesis*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity` <*wildcard*>	Option to specify the entity to get names from hierarchies instantiated by the entity
	`-filter` <*wildcard*>	Option to specify the node's full path name and/or wildcard character(s)
	`-library` <*wildcard*>	Option to specify the containing library
	`-node_type` <*all|comb|reg|pin|input|output|bidir|hierarchy|mem|bus|qsf|state_machine|assigned|unassigned|all_reg|partition|virtual*>	Option to filter based on the specified node type.
	`-observable_type` <*all|pre_synthesis|post_synthesis|post_fitter|post_asm|stp_pre_synthesis*>	Option to filter based on the specified observable type
Description		

```
Returns a filtered output collection of all matching
node name IDs found in a compiled Quartus Prime project.

To access each element of the output collection, use the Tcl
command "foreach_in_collection". To see example usage, type
"get_names -long_help" or "foreach_in_collection -long_help".

If the "-node_type" option is not specified, the default
value is "all". Similarly, if the "-observable_type" option
is not specified, the default value is "all".

The node type "pin" includes "input", "output", "bidir", "assinged" and "unassigned".
The node type "qsf" include names from qsf settings file.
The node type "all" includes all node types.
The node type "all_reg" includes all node types and register post-fitting

The value for "-observable_type" option can be one of
```

continued...

```
the following:

Observable Type      Description
---------------      -----------------------------------
all                  Use post-Fitter information. If it is
                     not available, post-Synthesis
                     information is used. Otherwise,
                     pre-synthesis information is used if it
                     exists.
pre_synthesis        Use pre-synthesis information.
post_synthesis       Use post-synthesis information.
post_fitter          Use post-Fitter information.
post_asm             Use post-Assembler information. The
                     post-Assembler information is only supported
                     for designs using the HardCopy II device family.
stp_pre_synthesis    Use Signal Tap pre-synthesis information.
```

Example Usage

```
# Search for a single post-Fitter pin with the name accel and
# make assignments
set accel_name_id [get_names -filter accel -node_type pin -observable_type post_fitter]
foreach_in_collection name_id $accel_name_id {

    # Get the full path name of the node
    set target [get_name_info -info full_path $name_id]

    # Set multicycle assignment
    set_multicycle_assignment -to $target 2

    # Set location assignment
    set_location_assignment -to $target Pin_E22
}
# Search for nodes of any post-Fitter node type with name length <= 5
# The default node type is "all"
set name_ids [get_names -filter ????? -observable_type post_fitter]
foreach_in_collection name_id $name_ids {

    # Print the name id
    puts $name_id

    # Print the node type
    puts [get_name_info -info node_type $name_id]

    # Print the full path (which excludes the current
    # focus entity from the path)
    puts [get_name_info -info full_path $name_id]
}
# Search for nodes of any post-Fitter node type that end in "eed".
# The default node type is "all"
set name_ids [get_names -filter *eed -observable_type post_fitter]
foreach_in_collection name_id $name_ids {

    # Print the name id
    puts $name_id

    # Print the node type
    puts [get_name_info -info node_type $name_id]

    # Print the full path (which excludes the current
    # focus entity from the path)
    puts [get_name_info -info full_path $name_id]
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Get names cannot return <*string*> because the name was found in a partition that's not the root partition. Refine your get_names search pattern to exclude child partitions
TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <*string*>. At the minimum, run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
TCL_ERROR	1	ERROR: Illegal node type: <*string*>. Specify all, comb, reg, pin, hierarchy, or bus.

continued...

	TCL_ERROR	1	ERROR: Illegal observable type: <*string*>. Specify all, pre_synthesis, post_synthesis, or post_fitter.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Post-Assembler compiler database does not exist for revision name: <*string*>. Run Assembler (quartus_asm) with the specified revision name before using this Tcl command.

3.1.28.24. get_parameter (::quartus::project)

The following table displays information for the `get_parameter` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_parameter [-h | -help] [-long_help] [-entity <entity_name> ] [-fall] -name <name> [-rise] [-tag <data> ] [-to <destination> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-entity <entity_name>`	Entity to which parameter belongs	
	`-fall`	Option applies to falling edge	
	`-name <name>`	Parameter name	
	`-rise`	Option applies to rising edge	
	`-tag <data>`	Option to tag data to this assignment	
	`-to <destination>`	Destination of parameter	
Description	`Returns the value of the parameter.` `The "-name" option is not case sensitive.` `The "-to" option is case sensitive.` `If the "-to" argument is specified, the function returns the parameter value for the current entity. The value is retrieved from the PARAMETERS section of the entity. Otherwise, the function returns the project-wide default parameter value obtained from the DEFAULT_PARAMETERS section.` `Use the "-entity" option to retrieve the parameter from the specified entity. If the "-entity" option is not specified, the value for the FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME value is not found, the revision name is used.`		
Example Usage	`## Get project-wide, default parameter value` `get_parameter -name WIDTH` `## Get entity-specific parameter value` `get_parameter -name inst1 -to SIZE`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.

continued...

	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Illegal default parameter: <*string*>. Specify a legal default parameter name.
	TCL_ERROR	1	ERROR: Illegal parameter: <*string*>. Specify a legal parameter name.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.25. get_project_directory (::quartus::project)

The following table displays information for the `get_project_directory` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_project_directory [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns the project directory for currently open project.`		
Example Usage	`project_open one_wire` `# Print the current project directory` `puts [get_project_directory]` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.26. get_project_revisions (::quartus::project)

The following table displays information for the `get_project_revisions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_project_revisions [-h | -help] [-long_help] [ <project_name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*project_name*>	Project name
Description	`Returns a list of revisions included in the specified project.` `If the project name is not specified, the current project name` `is used by default.` `The first element in the list of revisions is the current` `revision and is the same as the return value for the` `"get_current_revision" command.`	

continued...

Example Usage	<pre># Set the device family assignment to Stratix # for all revisions project_open chiptrip set original_revision [get_current_revision] foreach revision [get_project_revisions] { puts "$revision" set_current_revision $revision set_global_assignment -name FAMILY Stratix export_assignments } set_current_revision $original_revision project_close # Open the project with the first available revision # and set the device family assignment to Stratix set revision [lindex [get_project_revisions chiptrip] 0] open_project -revision $revision chiptrip set_global_assignment -name FAMILY Stratix project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: <*string*>. Specify a legal project name.
	TCL_ERROR	1	ERROR: Project name was not specified or open project does not exist. Open an existing project or specify the project name.

3.1.28.27. get_revision_description (::quartus::project)

The following table displays information for the `get_revision_description` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_revision_description [-h | -help] [-long_help] [` <*revision_name*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*revision_name*>	Revision name
Description	<pre>Returns the description for the specified revision.</pre>	
Example Usage	<pre>## Get the description for the current revision project_open chiptrip set revision_name [get_current_revision] get_revision_description $revision_name project_close</pre>	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Revision is not included in the current project: <*string*> . Use the create_revision command to create the revision.
	TCL_ERROR	1	ERROR: Project name was not specified or open project does not exist. Open an existing project or specify the project name.

3.1.28.28. get_top_level_entity (::quartus::project)

The following table displays information for the `get_top_level_entity` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`get_top_level_entity [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Returns the name id for the current focus entity.`		
Example Usage	`# Get the name id of the current focus entity` `set current_focus_entity_id [get_top_level_entity]` `# Print out the entity name of the current focus entity` `set msg "Entity name of the current focus entity => ("` `append msg [get_name_info -info entity_name $current_focus_entity_id]` `append msg ")"` `puts ""` `puts $msg`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <*string*>. At the minimum, run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.29. get_user_option (::quartus::project)

The following table displays information for the `get_user_option` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`get_user_option [-h | -help] [-long_help] -name` <*name*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name` <*name*>	User option name
Description	`Returns the user option value for the name specified by` `the "-name" option.` `To get a list of all available user option names, use` `the "get_all_user_option_names" command.`	
Example Usage	`## Get the value for the user option` `## "TALKBACK_ENABLED"` `set value [get_user_option -name TALKBACK_ENABLED]` `puts "TALKBACK_ENABLED = $value"`	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal user option name: <*string*>. Specify a legal user option name. To get a list of legal names, use the get_all_user_option_names command.

3.1.28.30. is_database_version_compatible (::quartus::project)

The following table displays information for the `is_database_version_compatible` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`is_database_version_compatible [-h | -help] [-long_help] <project_name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<*project_name*>	Project name	
Description	`Checks whether a project's database is compatible with current version of the software.` `Returns 1, if the database is compatible; returns 0, otherwise.` `Mainly this check is for databases that are restored from another` `version of the software`		
Example Usage	`## Check project database version compatibility` `if [is_database_version_compatible chiptrip] {` `  # proceed if compatible` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.28.31. is_fitter_in_qhd_mode (::quartus::project)

The following table displays information for the `is_fitter_in_qhd_mode` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`is_fitter_in_qhd_mode [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Returns true if we're running in QHD mode`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.28.32. is_project_open (::quartus::project)

The following table displays information for the `is_project_open` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`is_project_open [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Checks whether a project is currently open. Returns 1, if a project is currently open; returns 0, otherwise.`		
Example Usage	`## Close the project if open` `if [is_project_open] {` `  project_close` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.28.33. open_side_revision (::quartus::project)

The following table displays information for the `open_side_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`open_side_revision [-h | -help] [-long_help]` *<revision_name>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<revision_name>		Revision name
Description	`Loads the specified revision name into memory, without making it the current revision. Required step before assignments can be modified on the side with the use of the -revision option in get/set/remove assignments.` `Changes are not saved until close_side_revision is called.` `The specified revision cannot be the current revision, and once opened, cannot be made the current revision until it is closed.`		
Example Usage	`## Create and open "new_rev" as a side revision. Apply an assignment and close the revision.` `create_revision new_rev -based_on my_rev -copy_results` `open_side_revision new_rev` `set_global_assignment -name OPTIMIZATION_TECHNIQUE "Area" -revision new_rev` `close_side_revision new_rev`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	WARNING: Revision is already the current revision: *<string>*. No action is required.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

continued...

	TCL_ERROR	1	ERROR: Revision file does not exist: <*string*>.qsf. Use delete_revision to delete the revision from the current project. Then use create_revision to create the revision and its .qsf before setting <*string*> as the current revision.
	TCL_ERROR	1	ERROR: Revision is not included in the current project: <*string*> . Use the create_revision command to create the revision.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.34. project_archive (::quartus::project)

The following table displays information for the `project_archive` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`project_archive [-h | -help] [-long_help] [-all_revisions] [-include_libraries] [-include_outputs] [-overwrite] [-use_file_set` <*file_set*> `] [-version_compatible_database]` <*archive_name*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all_revisions`	Option to archive all revisions
	`-include_libraries`	Option to include related system libraries
	`-include_outputs`	Option to include output files in archive
	`-overwrite`	Option to overwrite any currently existing archive file
	`-use_file_set` <*file_set*>	Option to create the archive using the specified file set
	`-version_compatible_database`	Option to include version-compatible database if supported
	<*archive_name*>	Archive file name
Description		

```
Archives an open project and its related files into a
Quartus Prime Archive File (.qar).

The description of operations is as follows:

Option                       Description
------                       -------------------------
use_file_set                 Creates the archive using the specified
                             file set. By default, the 'basic'
                             file set is used. For more information
                             about file sets, type:

                             quartus_sh --archive -list_file_sets

all_revisions                Archives all revisions.

overwrite                    Overwrites existing archive file.

include_outputs              Includes output files in archive.

include_libraries            Includes related Megafunction and
                             IP library files.

version_compatible_database  Includes version-compatible database
                             if supported.
```

Example Usage

```
## Default mode: Archive current revisions without output files or libraries
project_archive chiptrip.qar

## Archive all revisions without output files or libraries
project_archive chiptrip.qar -all_revisions

## Archive current revision with version-compatible database if supported
```

continued...

```
project_archive chiptrip.qar -version_compatible_database

## Same as first one, but overwrite any existing archive file
project_archive chiptrip.qar -overwrite

## Include output files and libraries
project_archive chiptrip.qar -include_outputs -include_libraries
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Error(s) found while archiving the project. See error message(s) for details.
	TCL_ERROR	1	ERROR: Project archive failed. Some files could not be processed. Refer to the Quartus Prime Archive Log File (<archive_name>.qarlog).
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.35. project_clean (::quartus::project)

The following table displays information for the `project_clean` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`project_clean [-h | -help] [-long_help] [-current_version] [-revision` <*revision_name*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-current_version`	Only clean project files for the current Quartus version.
	`-revision` <*revision_name*>	Revision to clean (if omitted, all revisions of the open project are cleaned). Revision name can contain wildcards (i.e., '*').
Description		

```
Cleans database and compiler-generated output for the specified
revision (or all revisions if no revision is specified).  Cleaning
revisions removes database and other files generated by the Quartus Prime
software, including report and programming files.

This command closes the currently open project before cleaning.

Specifically, this cleans all of the following files/folders (for all matching revisions):

<revision>.*.rpt
<revision>.*.rpt.htm
<revision>.*.rpt.htm_files
<revision>.*.msf
<revision>.*.smsg
<revision>.*.summary
<revision>.jdi
<revision>.pin
<revision>.pof
<revision>.sof
<revision>.done
<revision>.sld
(all revision files in the qdb and persona directories)
```

continued...

Example Usage	<pre>project_clean -revision "foo" # Cleans revision "foo" project_clean -revision "foo*" # Cleans all revisions starting with "foo" (e.g., "foo", "foobar") project_clean # Cleans all revisions in project</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Failed when attempting to remove database files in <*string*> for revision <*string*>.
	TCL_ERROR	1	ERROR: Failed when attempting to remove file <*string*> for revision <*string*>.
	TCL_ERROR	1	ERROR: Couldn't delete file or folder named '<*string*>'.
	TCL_ERROR	1	ERROR: Couldn't open configuration settings...the settings object is currently locked.
	TCL_ERROR	1	ERROR: Can't access revision '<*string*>'. The revision may not exist, or there may be an error in the revision settings.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.36. project_close (::quartus::project)

The following table displays information for the `project_close` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`project_close [-h | -help] [-long_help] [-dont_export_assignments]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-dont_export_assignments`	Do not export assignments to file
Description	<pre>Closes an open project. The assignments created or modified during an open project are committed to the Quartus Prime Settings File (.qsf) during a "project_close", unless you use the "-dont_export_assignments" option.</pre>	
Example Usage	<pre>## Close the project if open if [is_project_open] { project_close } ## Close the project if open ## and do not export the assignments if [is_project_open] { project_close -dont_export_assignments }</pre>	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.

continued...

	TCL_ERROR	1	ERROR: Failed when attempting to write assignments back to QSF.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.37. project_exists (::quartus::project)

The following table displays information for the `project_exists` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`project_exists [-h | -help] [-long_help] <project_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<*project_name*>		Project name
Description	`Checks whether a project exists. Returns 1, if a project exists; returns 0, otherwise.`		
Example Usage	`## Create project if one does not exist.` `## Open existing project otherwise.` `if [project_exists chiptrip] {` `  project_open chiptrip` `} else {` `  project_new chiptrip` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.28.38. project_new (::quartus::project)

The following table displays information for the `project_new` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`project_new [-h | -help] [-long_help] [-family <family> ] [-overwrite] [-part <part> ] [-revision <revision_name> ] <project_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-family <family>`		Family name
	`-overwrite`		Option to overwrite existing project and revision
	`-part <part>`		Part name
	`-revision <revision_name>`		Revision name
	<*project_name*>		Project name

continued...

Description

```
Creates and opens a new project with the specified project name.

If the "-revision" option is not specified, the project name
is used to create the revision.

Assignments created or modified by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## Create project "chiptrip" and revision "chiptrip"
project_new chiptrip

## Create project "chiptrip" and revision "auto_max"
project_new -revision auto_max chiptrip

## Create project "chiptrip" and revision "chiptrip"
## Overwrite any Quartus Prime Settings File (.qsf) if it exists
project_new chiptrip -overwrite

## Create project "chiptrip" and revision "chiptrip"
## Set the FAMILY assignment to Stratix
project_new chiptrip -family Stratix
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: The -<*string*> option must also be used when you use the -<*string*> option. Specify both options.
TCL_ERROR	1	ERROR: Can't create project: <*string*>. Specify a legal project name.
TCL_ERROR	1	ERROR: Can't create revision: <*string*>. Specify a legal revision name using the -<*string*> option.
TCL_ERROR	1	ERROR: Can't create revision: <*string*>. Specify a legal revision name.
TCL_ERROR	1	ERROR: Can't create settings files for project: <*string*>. Make sure the .psf, .csf, and .ssf files are writeable.
TCL_ERROR	1	ERROR: Can't open project: <*string*>
TCL_ERROR	1	ERROR: Can't remove Quartus Prime Settings File: <*string*>. Make sure the file is writeable.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
TCL_ERROR	1	ERROR: Project already exists: <*string*>. Specify a different project name or use the -overwrite option.

3.1.28.39. project_open (::quartus::project)

The following table displays information for the `project_open` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`project_open [-h | -help] [-long_help] [-current_revision] [-force] [-preserve_revision_order] [-revision` *<revision_name>* `] <project_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-current_revision`		Option to open the current revision automatically
	`-force`		Option to open the project and overwrite the compilation database if the database version is incompatible.
	`-preserve_revision_order`		Option to not move the revision to top in the qpf file and preserve existing revision ordering
	`-revision` *<revision_name>*		Revision name
	<project_name>		Project name
Description	`Opens an existing project. To create a new project, use the project_new command.` `If the -revision option is not specified, the project name is specified as the revision name.` `The project_open command gives an error when the compilation database version is not compatible with the current version of Quartus Prime software. You may specify the "-force" option to avoid the error and overwrite the database.`		
Example Usage	`## Open project "chiptrip" and revision "chiptrip"` `project_open chiptrip` `## Open project "chiptrip" and revision "auto_max"` `project_open -revision auto_max chiptrip` `## Get the current revision before opening` `## the project with the current revision` `set project_name chiptrip` `set current_revision [get_current_revision $project_name]` `project_open -revision $current_revision $project_name` `puts [get_global_assignment -name FAMILY]` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	WARNING: Project is already open: *<string>*
	TCL_ERROR	1	ERROR: Can't open project: *<string>*. First close the currently open project: *<string>*.
	TCL_ERROR	1	ERROR: Can't open project: *<string>*
	TCL_ERROR	1	ERROR: Can't set revision: *<string>*. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: *<string>*. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.

continued...

	TCL_ERROR	1	ERROR: Cannot open project: <*string*>. The project is not compatible with the installed version of the Quartus Prime software. Opening the project will overwrite the old project database. If you wish to overwrite the old project database, make sure to specify the -<*string*> option.
	TCL_ERROR	1	ERROR: Can't open revision: <*string*> (project: <*string*>). The revision is not compatible with the installed version of the Quartus Prime software. Opening the revision will overwrite the old revision database. If you wish to overwrite the old revision database, make sure to specify the -<*string*> option.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Revision does not exist: <*string*>. Specify a legal revision name using the -<*string*> option.
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: <*string*>. Specify a legal project name.

3.1.28.40. project_restore (::quartus::project)

The following table displays information for the `project_restore` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`project_restore [-h | -help] [-long_help] [-destination` <*directory*> `] [-overwrite] [-update_included_file_info]` <*archive_file*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-destination` <*directory*>		Directory where restored files are placed
	`-overwrite`		Option to overwrite files in destination directory
	`-update_included_file_info`		Option to update included file information
	<*archive_file*>		Archive file name
Description			

```
Restores a Quartus Prime Archive File (.qar) that contains
the project and its related files.

By default, the archive is restored into the current
directory. Use the "-destination" option to restore the
files into a new directory.

By default, the command fails if the archive already
contains files in the destination directory. Use the
"-overwrite" option to overwrite any existing files
in the destination directory.
```

Example Usage

```
## Restore archive and expand files into current directory
project_restore chiptrip.qar
## or
project_restore chiptrip.qar -destination

## Restore archive. Expand files into current directory,
## but overwrite any existing files in "."
project_restore chiptrip.qar -destination . -overwrite

## Restore project into a "restored" sub-directory
project_restore chiptrip.qar -destination "restored" -overwrite
```

Return Value	**Code Name**	**Code**	**String Return**

continued...

TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Error(s) found while restoring the archive. See error message(s) for details.

3.1.28.41. remove_all_global_assignments (::quartus::project)

The following table displays information for the `remove_all_global_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`remove_all_global_assignments [-h | -help] [-long_help] [-entity` <*entity_name*> `] [-fall] -name` <*name*> `[-rise] [-section_id` <*section id*> `] [-tag` <*data*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity` <*entity_name*>	Entity to which assignment belongs
	`-fall`	Option applies to falling edge
	`-name` <*name*>	Assignment name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-section_id` <*section id*>	Section id
	`-tag` <*data*>	Option to tag data to this assignment
Description		

```
Removes all matching global assignments.

The "-name" option is not case sensitive. This option can take
string patterns containing special characters from the set
"*?\[]" as the value. The value is matched using Tcl string
matching.

This Tcl command reads the global assignments found in
the Quartus Prime Settings File (.qsf). This Tcl command filters
the assignments data found in the .qsf and removes the data
based on the values specified by the "-name" option.

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                    Description
---------------------------   -----------------------------
CHIP                          Same as revision name
LOGICLOCK_REGION              A unique name
EDA_TOOL_SETTINGS             A unique name
CLIQUE                        A unique name
BREAKPOINT                    A unique name
CLOCK                         A unique name
AUTO_INSERT_SLD_NODE_ENTITY   A unique name

For entity-specific assignments, use the "-entity" option to
remove the assignment(s) from the specified entity. If the
"-entity" option is not specified, the value for the
```

continued...

```
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.

Assignments removed by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## Remove all the registered source files

remove_all_global_assignments -name SOURCE_FILE

# Using wildcards
remove_all_global_assignments -name SOURCE*
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: <*string*> global assignment(s) were removed
TCL_OK	0	WARNING: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.

continued...

	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.42. remove_all_instance_assignments (::quartus::project)

The following table displays information for the `remove_all_instance_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`remove_all_instance_assignments [-h | -help] [-long_help] [-entity` <*entity_name*> `] [-fall] [-from` <*source*> `] -name` <*name*> `[-rise] [-section_id` <*section id*> `] [-tag` <*data*> `] [-to` <*destination*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity` <*entity_name*>	Entity to which assignment belongs
	`-fall`	Option applies to falling edge
	`-from` <*source*>	Source of the assignment (string pattern is matched using Tcl string matching)
	`-name` <*name*>	Assignment name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-section_id` <*section id*>	Section id
	`-tag` <*data*>	Option to tag data to this assignment
	`-to` <*destination*>	Destination of the assignment (string pattern is matched using Tcl string matching)
Description		

```
Removes all matching instance assignment values.

The "-name" option is not case sensitive.
The "-to" and "-from" options are case sensitive.

These options can take string patterns containing special
characters from the set "*?\[]" as values. The values are
matched using Tcl string matching. Note that bus names are
automatically detected and do not need to be escaped. Bus
names have the following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
remove_all_instance_assignments -name LOCATION -to a[0]

# To match all indices of bus "a", type:
remove_all_instance_assignments -name LOCATION -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:
```

continued...

```
remove_all_instance_assignments -name LOCATION -to "a[escape_brackets \[]\[0-2\][escape_brackets
\]]"

For more information about escaping square brackets, type
"escape_brackets -h".

This Tcl command reads the instance assignments found in
the Quartus Prime Settings File (.qsf) and removes this data based
on the values specified by the "-name", "-from", and "-to"
options.

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                    Description
---------------------------   ------------------------------
CHIP                          Same as revision name
LOGICLOCK_REGION              A unique name
EDA_TOOL_SETTINGS             A unique name
CLIQUE                        A unique name
BREAKPOINT                    A unique name
CLOCK                         A unique name
AUTO_INSERT_SLD_NODE_ENTITY   A unique name

For entity-specific assignments, use the "-entity" option to
remove the assignment(s) from the specified entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.

Assignments removed by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## Remove all the timing requirements
## Use wildcards to catch TSU_REQUIREMENT, TCO_REQUIREMENT,
## and others
remove_all_instance_assignments -name *_REQUIREMENT

## Remove all the location assignments with
## the destination bus name "timeo".
set bus_name "timeo"
remove_all_instance_assignments -name LOCATION -to $bus_name[*]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: <*string*> instance assignment(s) were removed
TCL_OK	0	WARNING: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not an instance assignment: <*string*> -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.

continued...

	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.43. remove_all_parameters (::quartus::project)

The following table displays information for the `remove_all_parameters` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`remove_all_parameters [-h | -help] [-long_help] [-entity <entity_name> ] [-fall] -name <name> [-rise] [-tag <data> ] [-to <destination> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which parameter belongs
	`-fall`	Option applies to falling edge
	`-name <name>`	Parameter name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of the parameter (string pattern is matched using Tcl string matching)
Description		

```
Removes all matching parameters.

The "-name" option is not case sensitive.
The "-to" option is case sensitive.

If the "-to" argument is specified, the function removes
the parameters from the current entity. The parameters are
```

continued...

```
removed from the PARAMETERS section of the entity. Otherwise,
the function removes the project-wide default parameters
obtained from the DEFAULT_PARAMETERS section.

This Tcl command filters the parameter data found in the
Quartus Prime Settings File (.qsf) and removes the data based on
the values specified by the "-name" and "-to" options. These
options can take string patterns containing special characters
from the set "*?\[]" as values. The values are matched using
Tcl string matching. Note that bus names are automatically
detected and do not need to be escaped. Bus names have the
following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
remove_all_parameters -name * -to a[0]

# To match all indices of bus "a", type:
remove_all_parameters -name * -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

remove_all_parameters -name * -to "a[escape_brackets \[]\[0-2\][escape_brackets \]]"

For more information about escaping square brackets, type
"escape_brackets -h".

Use the "-entity" option to remove the parameters from the
specified entity. If the "-entity" option is not specified,
the value for the FOCUS_ENTITY_NAME assignment is used. If the
FOCUS_ENTITY_NAME value is not found, the revision name is used.

The parameters removed by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## The following 3 examples remove project-wide,
## default parameter values
remove_all_parameters -name WIDTH
remove_all_parameters -name *ID*
remove_all_parameters -name *

## The following 3 examples remove entity-specific
## parameter values
remove_all_parameters -name inst1 -to SIZE
remove_all_parameters -name inst1 -to *IZ*
remove_all_parameters -name inst1 -to *
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: <*string*> parameter(s) were removed
TCL_OK	0	INFO: Removed parameter: <*string*>
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.

continued...

	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Parameter does not exist and cannot be removed: <*string*>. Specify an existing parameter name.
	TCL_ERROR	1	ERROR: Illegal default parameter: <*string*>. Specify a legal default parameter name.
	TCL_ERROR	1	ERROR: Illegal parameter: <*string*>. Specify a legal parameter name.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.28.44. resolve_file_path (::quartus::project)

The following table displays information for the `resolve_file_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`resolve_file_path [-h | -help] [-long_help] <file_name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*file_name*>	Option to specify the file name
Description		

```
Returns the resolved full path of the specified file name.
If the file does not exist, the original file name is
returned.

The Quartus Prime software resolves relative paths by searching
for the file in the following directories in the following
order:

1) Project directory, which is the directory where the
   Quartus Prime Settings File (.qsf) is found.
2) Project database directory, which is the "db" directory
   found under the project directory.
3) Project library directories, which are the directories
   containing the user-specified libraries that are used
   only by the current project.
4) User library directories, which are the directories
   containing the user-specified libraries that are
   used by all Quartus Prime projects.
5) Quartus Prime library directory, which is the directory
   containing Quartus Prime libraries.
```

Example Usage

```
project_new chiptrip -overwrite

# Set one Verilog source file assignment
set_global_assignment -name VERILOG_FILE chiptrip.v

# Display the resolved full path of the Verilog
# source file assignment
set filename [get_global_assignment -name VERILOG_FILE]
set resolved_fullpath [resolve_file_path $filename]

puts "Full Path: $resolved_fullpath"

# Set more Verilog source file assignments
set_global_assignment -name VERILOG_FILE auto_max.v
set_global_assignment -name VERILOG_FILE speed_ch.v
set_global_assignment -name VERILOG_FILE tick_cnt.v
set_global_assignment -name VERILOG_FILE time_cnt.v

# Display the resolved full path of all the Verilog
# source file assignments
set file_asgns [get_all_global_assignments -name VERILOG_FILE]
foreach_in_collection file_asgn $file_asgns {
```

continued...

```
    ## Each element in the collection has the following
    ## format: {} {VERILOG_FILE} {<file_name>}

    set filename [lindex $file_asgn 2]
    set resolved_fullpath [resolve_file_path $filename]

    puts "Full Path: $resolved_fullpath"
}

project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.45. revision_exists (::quartus::project)

The following table displays information for the `revision_exists` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`revision_exists [-h | -help] [-long_help] [-project` *<project_name>* `]` *<revision_name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-project` *<project_name>*	Project name
	<revision_name>	Revision name
Description		

```
Checks whether the revision exists for the specified project
or currently open project.

Returns 1, if the revision exists; returns 0, otherwise.
```

Example Usage

```
## Check if the specified revision exists
## in the specified project
if [revision_exists -ARG(project) chiptrip speed_ch] {
    puts "Revision exists"
} else {
    puts "Revision does not exist"
}

## Create revision for the currently open
## project if it does not exist
## Set the current revision otherwise
project_open chiptrip
if [revision_exists speed_ch] {
    set_current_revision speed_ch
} else {
    create_revision speed_ch
}
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: *<string>*. Specify a legal project name.
	TCL_ERROR	1	ERROR: Project name was not specified or open project does not exist. Open an existing project or specify the project name.

3.1.28.46. set_current_revision (::quartus::project)

The following table displays information for the `set_current_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`set_current_revision [-h | -help] [-long_help] [-force] <revision_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-force`		Option to open the revision and overwrite the compilation database if the database version is incompatible.
	<revision_name>		Revision name
Description	`Sets the specified revision name as the current revision. All assignments created or modified during an open project will also be saved to the Quartus Prime Settings File (.qsf).` `In 8.1 or later versions of Quartus Prime software, set_current_revision gives an error when the compilation database version is not compatible with the current version of Quartus Prime software. You may specify the "-force" option to avoid the error and overwrite the database.`		
Example Usage	`## Sets "auto_max" as the current revision` `set_current_revision auto_max`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	WARNING: Revision is already the current revision: *<string>*. No action is required.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Revision file does not exist: *<string>*.qsf. Use delete_revision to delete the revision from the current project. Then use create_revision to create the revision and its .qsf before setting *<string>* as the current revision.
	TCL_ERROR	1	ERROR: Revision is not included in the current project: *<string>* . Use the create_revision command to create the revision.

3.1.28.47. set_global_assignment (::quartus::project)

The following table displays information for the `set_global_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`set_global_assignment [-h | -help] [-long_help] [-comment <comment> ] [-disable] [-entity <entity_name> ] [-fall] -name <name> [-remove] [-rise] [-section_id <section id> ] [-tag <data> ] [ <value> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-comment` <*comment*>	Comment
	`-disable`	Option to disable assignment
	`-entity` <*entity_name*>	Entity to which to add assignment
	`-fall`	Option applies to falling edge
	`-name` <*name*>	Assignment name
	`-remove`	Option to remove assignment
	`-rise`	Option applies to rising edge
	`-section_id` <*section id*>	Section id
	`-tag` <*data*>	Option to tag data to this assignment
	<*value*>	Assignment value

Description

```
Sets or removes a global assignment.

Assignments created or modified by using this Tcl command are not
saved to the Quartus Prime Settings File (.qsf) unless you explicitly
call one of the following two Tcl commands (from the
::quartus::project Tcl package):

1) export_assignments
2) project_close (unless -dont_export_assignments is specified as an
argument to project_close)

You must save assignment changes before you run Quartus Prime
command-line executables. Note, however, that the Tcl commands
execute_flow and execute_module (from the ::quartus::flow Tcl
package) call "export_assignments" before they run
command-line executables.

For entity-specific assignments, use the -entity option to force the
assignment to specified entity. If the -entity option is not
specified, the value for the FOCUS_ENTITY_NAME assignment is used. If
the FOCUS_ENTITY_NAME value is not found, the revision name is used.

If the Quartus Prime Settings File contains a USER_LIBRARIES assignment
and you call set_global_assignment to set a SEARCH_PATH or
USER_LIBRARIES assignment, the existing USER_LIBRARIES assignment
expands into one or more SEARCH_PATH assignments.

Note that values that begin with a dash ("-") should be enclosed in a
backslash followed by a quote. In the following example, -02 is
enclosed by \" at the beginning and the end.

set_global_assignment -name ARM_CPP_COMMAND_LINE \"-O2\"
```

Example Usage

```
## Specify Stratix as the family to use when compiling
set_global_assignment -name FAMILY Stratix

## If the family name has empty spaces, use quotes
set_global_assignment -name FAMILY "Stratix GX"

## or remove any empty space
set_global_assignment -name FAMILY StratixGX
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Assignment <*string*> is not supported in this edition of the Quartus Prime software.
TCL_OK	0	WARNING: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.

continued...

	TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: The assignment <*string*> is from Design Template and can't be changed/removed.
	TCL_ERROR	1	ERROR: File name <*string*> exceeds maximum of <*string*> characters. Specify a file name with fewer characters.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: The -<*string*> option is not required but was specified with the value: <*string*>. Delete the option.
	TCL_ERROR	1	ERROR: The -<*string*> option is required but was not specified. Specify the required option.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.
	TCL_ERROR	1	ERROR: Assignment <*string*> cannot be removed -- it has multiple values. Specify one value to remove or use the <*string*> command to remove all values for the assignment.
	TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.28.48. set_high_effort_fmax_optimization_assignments (::quartus::project)

The following table displays information for the `set_high_effort_fmax_optimization_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`set_high_effort_fmax_optimization_assignments [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Sets assignments that together implement the high-effort fmax optimization flow. This Tcl command only sets the assignments but does not run a compilation.`		
Example Usage	`## Open the project` `project_open $project_name` `## Set assignments that implement the high-effort` `## fmax optimization flow` `set_high_effort_fmax_optimization_assignments`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Set global assignment <*string*> to <*string*>.
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.28.49. set_instance_assignment (::quartus::project)

The following table displays information for the `set_instance_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`set_instance_assignment [-h | -help] [-long_help] [-comment` <*comment*> `] [-disable] [-entity` <*entity_name*> `] [-fall] [-from` <*source*> `] -name` <*name*> `[-remove] [-rise] [-section_id` <*section id*> `] [-tag` <*data*> `] [-to` <*destination*> `] [` <*value*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment` <*comment*>	Comment
	`-disable`	Option to disable assignment
	`-entity` <*entity_name*>	Entity to which to add assignment
	`-fall`	Option applies to falling edge

continued...

	`-from` *<source>*	Source of assignment
	`-name` *<name>*	Assignment name
	`-remove`	Option to remove assignment
	`-rise`	Option applies to rising edge
	`-section_id` *<section id>*	Section id
	`-tag` *<data>*	Option to tag data to this assignment
	`-to` *<destination>*	Destination of assignment
	<value>	Assignment value

Description

```
Sets or removes an instance assignment.

Assignments created or modified by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.

For entity-specific assignments, use the "-entity" option to
force the assignment to specified entity. If the "-entity"
option is not specified, the value for the FOCUS_ENTITY_NAME
assignment is used. If the FOCUS_ENTITY_NAME value is not found,
the revision name is used.
```

Example Usage

```
## Specify a TSU_REQUIREMENT of 2ns from mypin to any register
set_instance_assignment -from "mypin" -to * -name TSU_REQUIREMENT 2ns

## Remove the TSU_REQUIREMENT from mypin to all registers
set_instance_assignment -from "mypin" -to * -name TSU_REQUIREMENT -remove

## Specify the entity to which the assignment is added,
## use the -entity option
## This is needed if the top-level entity name is other than
## that of the project name
## The following command generates a top_level entity
set_instance_assignment -from "mypin" -to * -entity top_level -name TSU_REQUIREMENT 2ns
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	WARNING: Ignored assignment: *<string>*. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not an instance assignment: *<string>* -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: *<string>*. Specify a legal entity name.

continued...

	TCL_ERROR	1	ERROR: The assignment <*string*> is from Design Template and can't be changed/removed.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: The -<*string*> option is not required but was specified with the value: <*string*>. Delete the option.
	TCL_ERROR	1	ERROR: The -<*string*> option is required but was not specified. Specify the required option.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.
	TCL_ERROR	1	ERROR: Assignment <*string*> cannot be removed -- it has multiple values. Specify one value to remove or use the <*string*> command to remove all values for the assignment.
	TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.28.50. set_io_assignment (::quartus::project)

The following table displays information for the `set_io_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`set_io_assignment [-h | -help] [-long_help] [-comment` <*comment*> `] [-disable] [-fall] [-io_standard` <*io standard*> `] -name` <*name*> `[-remove] [-rise] [-tag` <*data*> `] [` <*value*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment` <*comment*>	Comment
	`-disable`	Option to disable assignment
	`-fall`	Option applies to falling edge
	`-io_standard` <*io standard*>	Option to specify the io standard

continued...

	`-name` *<name>*	Assignment name
	`-remove`	Option to remove assignment
	`-rise`	Option applies to rising edge
	`-tag` *<data>*	Option to tag data to this assignment
	<value>	Assignment value

Description

```
Sets or removes an io assignment.

Assignments created or modified by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## Specify LVTTL as the IO Standard for OUTPUT_PIN_LOAD assignment
set_io_assignment 30 -name OUTPUT_PIN_LOAD -io_standard LVTTL
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	WARNING: Ignored assignment: *<string>*. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: *<string>* -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: *<string>*. Specify a legal entity name.
TCL_ERROR	1	ERROR: File name *<string>* exceeds maximum of *<string>* characters. Specify a file name with fewer characters.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: *<string>*. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value *<string>* for the -*<string>* option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Option -*<string>* for *<string>* assignment is illegal. Specify a legal option or remove the option.
TCL_ERROR	1	ERROR: Options cannot be specified together: -*<string>*, -*<string>* and -*<string>*. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Value *<string>* for *<string>* assignment is illegal. Specify a legal value.

continued...

	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: The -<*string*> option is not required but was specified with the value: <*string*>. Delete the option.
	TCL_ERROR	1	ERROR: The -<*string*> option is required but was not specified. Specify the required option.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: Assignment <*string*> cannot be removed -- it has multiple values. Specify one value to remove or use the <*string*> command to remove all values for the assignment.
	TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.28.51. set_location_assignment (::quartus::project)

The following table displays information for the `set_location_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`set_location_assignment [-h | -help] [-long_help] [-comment` <*comment*> `] [-disable] [-fall] [-remove] [-rise] [-tag` <*data*> `] -to` <*destination*> `[` <*value*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment` <*comment*>	Comment
	`-disable`	Option to disable assignment
	`-fall`	Option applies to falling edge
	`-remove`	Option to remove assignment
	`-rise`	Option applies to rising edge
	`-tag` <*data*>	Option to tag data to this assignment
	`-to` <*destination*>	Destination of assignment
	<*value*>	Assignment value
Description		

```
Sets or removes a location assignment.

               Valid location assignments, and settings for those
               assignments for your design are determined by the
               target device, package type, and pin count.

               To explore possible assignments and settings for your
               design and device, in the Quartus Prime Assignment
               Editor, specify Location for Assignment Name, and then
               click Browse in the Value column. In the Location
               dialog box, explore assignable device resources in the
               Element list, and then use the lists that appear
               (based on the element selected) to determine locations
```

continued...

```
                of resources that can be specified as the value for
                the assignment.

Assignments created or modified by using this Tcl
command are not saved to the Quartus Prime Settings File
(.qsf) unless you explicitly call one of the following
two Tcl commands:

1) export_assignments

                2) project_close (unless "-dont_export_assignments" is
                specified)

These two Tcl commands reside in the
::quartus::project Tcl package. You must save
assignment changes before you run Quartus Prime
command-line executables. Note, however, that the Tcl
commands "execute_flow" and "execute_module" (part of
the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line
executables.
```

Example Usage

```
set_location_assignment -to dst LOC
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: The assignment <*string*> is from Design Template and can't be changed/removed.
TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.28.52. set_parameter (::quartus::project)

The following table displays information for the `set_parameter` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`set_parameter [-h | -help] [-long_help] [-comment` <*comment*> `] [-disable] [-fall] -name` <*name*> `[-remove] [-rise] [-tag` <*data*> `] [` <*value*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment` <*comment*>	Comment
	`-disable`	Option to disable parameter
	`-fall`	Option applies to falling edge
	`-name` <*name*>	Parameter name
	`-remove`	Option to remove parameter
	`-rise`	Option applies to rising edge

continued...

	`-tag` <*data*>	Option to tag data to this assignment
	<*value*>	Parameter value
Description		

```
Sets or removes the specified parameter name.

The "-name" option is not case sensitive.

The parameters created or modified by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.

set_parameter can be used to overwrite the parameters in the top-level
entity of the design. A warning will be given if the parameter
can not be applied.

set_parameter assignments with "-entity" and "-to" are ignored and
a critical warning is given if they are used.
```

Example Usage

```
## Set project-wide, default WIDTH parameter value
set_parameter -name WIDTH 8
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Removed parameter: <*string*>
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: The -<*string*> option is not required but was specified with the value: <*string*>. Delete the option.
TCL_ERROR	1	ERROR: The -<*string*> option is required but was not specified. Specify the required option.
TCL_ERROR	1	ERROR: Parameter does not exist and cannot be removed: <*string*>. Specify an existing parameter name.
TCL_ERROR	1	ERROR: An unknown error has occured.
TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.28.53. set_power_file_assignment (::quartus::project)

The following table displays information for the `set_power_file_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`set_power_file_assignment [-h | -help] [-long_help] [-remove] [-saf_file <saf_file> ] [-section_id <section_id> ] [-to <to> ] [-vcd_end_time <vcd_end_time> ] [-vcd_file <vcd_file> ] [-vcd_start_time <vcd_start_time> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-remove`	Option to remove assignment
	`-saf_file <saf_file>`	SAF file name
	`-section_id <section_id>`	Section id
	`-to <to>`	Entity to which to apply power input file
	`-vcd_end_time <vcd_end_time>`	End time for VCD file parsing
	`-vcd_file <vcd_file>`	VCD file name
	`-vcd_start_time <vcd_start_time>`	Start time for VCD file parsing
Description		

```
Sets or removes a power input file assignment. Power input file
assignments are specified using multiple global assignments, and
a single instance assignment as illustrated in the following
example:

set_global_assignment -name POWER_INPUT_FILE_NAME "test.vcd" -section_id test.vcd
set_global_assignment -name POWER_INPUT_FILE_TYPE VCD -section_id test.vcd
set_global_assignment -name POWER_VCD_FILE_START_TIME "10 ns" -section_id test.vcd
set_global_assignment -name POWER_VCD_FILE_END_TIME "1000 ns" -section_id test.vcd
set_instance_assignment -name POWER_READ_INPUT_FILE test.vcd -to test_design

The power input file assignment serves as a wrapper for all of the
above assignments. If the "-remove" setting is not set, the
set_power_file_assignment will also make the following assignment
to enable the use of input files:

set_global_assignment -name POWER_USE_INPUT_FILES ON

If you do not specify a "-section_id", a new section identifier is
created for the input file assignment. If a "-section_id" is
specified and it does not already exist, it is used as the new
section identifier. If a "-section_id" is specified and it does
exist, the existing input file assignments are removed and a new
input file assignment is created using the given parameters and
section identifier.

If an entity name given by "-to" is not specified, the input file
assignment applys to the top level design entity.

If the "-remove" setting is used, the input file assignment given
by the "-section_id", "-vcd_file", or "-saf_file" is removed
from the project.

Assignments created or modified by using this Tcl command are
saved to the Quartus Prime Settings File (.qsf).
```

Example Usage

```
## Specify an input SAF file applied to the top level entity
## A default section will be created
set_power_file_assignment -saf_file test.saf

## Specify an input VCD file applied to design_top|counter1
## Use the given section_id to create a new section
set_power_file_assignment -vcd_file test.vcd -to design_top|counter1 -section_id test.vcd

## Update the previous input VCD file assignment to specify a
## start and end time
```

continued...

	`` set_power_file_assignment -vcd_file test.vcd -to design_top|counter1 -vcd_start_time 10ns - vcd_end_time 100ns -section_id test.vcd ## Remove the input SAF file assignment using the file name set_power_file_assignment -saf_file test.saf -remove ## Remove the input VCD file assignment using the section identifier set_power_file_assignment -section_id test.vcd -remove ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <*string*>. At the minimum, run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
	TCL_ERROR	1	ERROR: Exactly one of the following file name options must be specified: -<*string*> or -<*string*>.
	TCL_ERROR	1	ERROR: If -<*string*> is set, exactly one of the following options must be specified: -<*string*>, -<*string*> or -<*string*>. All other options must not be set.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: -<*string*> and -<*string*> cannot be used with -<*string*> option.

3.1.28.54. set_revision_description (::quartus::project)

The following table displays information for the `set_revision_description` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`set_revision_description [-h | -help] [-long_help] -description` <*description*> [<*revision_name*>]		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-description` <*description*>	Revision description	
	<*revision_name*>	Revision name	
Description	``` Sets the description for the specified revision. ```		
Example Usage	``` ## Create a description for the revision "auto_max" set_revision_description auto_max -description "I am auto_max" ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Revision is not included in the current project: <*string*> . Use the create_revision command to create the revision.

3.1.28.55. set_user_option (::quartus::project)

The following table displays information for the `set_user_option` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373		
Syntax	`set_user_option [-h | -help] [-long_help] -name <name> [ <value> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <name>`	User option name	
	<value>	User option value	
Description	`Sets the user option value for the name specified by the "-name" option. The user option is written to the quartus2.ini file.` `To get a list of all available user option names, use the "get_all_user_option_names" command.`		
Example Usage	`## Set TALKBACK_ENABLED to "on"` `set_user_option -name TALKBACK_ENABLED on`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal user option name: *<string>*. Specify a legal user option name. To get a list of legal names, use the get_all_user_option_names command.
	TCL_ERROR	1	ERROR: Illegal user option value: *<string>*. Specify a legal user option value.

3.1.28.56. test_assignment_trait (::quartus::project)

The following table displays information for the `test_assignment_trait` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project on page 373	
Syntax	`test_assignment_trait [-h | -help] [-long_help] -name <name> -trait <trait_name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	Assignment name
	`-trait <trait_name>`	Trait name
Description	`Checks whether the assignment name has the specified trait.` `Returns 1, if the assignment name has the trait;` `returns 0, otherwise.`	
Example Usage	`## Test if the assignment name is case-sensitive` `if {[test_assignment_trait -name VHDL_FILE -trait CASE_SENSITIVE]} {` `  puts "VHDL_FILE assignment is case-sensitive."`	

continued...

	`} else {` `  puts "VHDL_FILE assignment is not case-sensitive."` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Assignment <*string*> is not supported in this edition of the Quartus Prime software.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Ignored assignment: <*string*>. The assignment is no longer supported.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: Illegal trait: <*string*>. Specify a legal trait name.

3.1.29. ::quartus::project2

The following table displays information for the **::quartus::project2** Tcl package:

Tcl Package and Version	::quartus::project2 1.0
Description	`This package contains no general description.`
Availability	`This package is loaded by default in the following executables:` `hdb_debug` `qpro` `qpro_sh` `quartus` `quartus_asm` `quartus_bpps` `quartus_cdb` `quartus_design` `quartus_eda` `quartus_fit` `quartus_idb` `quartus_ipd` `quartus_ipgenerate` `quartus_map` `quartus_sh` `quartus_si` `quartus_sim` `quartus_sta` `quartus_stp` `quartus_syn` `quartus_tlg`
Tcl Commands	`quartus::close_project (::quartus::project2)` on page 439 `quartus::open_project (::quartus::project2)` on page 439

3.1.29.1. quartus::close_project (::quartus::project2)

The following table displays information for the `quartus::close_project` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project2 on page 438		
Syntax	`quartus::close_project [-h | -help] [-long_help] [-dont_export_assignments]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-dont_export_assignments`	Do not export assignments to file	
Description	`Closes an open project.` `The assignments created or modified during an open project are committed to the Quartus Prime Settings File (.qsf) during a "project_close", unless you use the "-dont_export_assignments" option.`		
Example Usage	`## Close the project if open` `if [is_project_open] {` `  quartus::close_project` `}` `## Close the project if open` `## and do not export the assignments` `if [is_project_open] {` `  quartus::close_project -dont_export_assignments` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Failed when attempting to write assignments back to QSF.
	TCL_ERROR	1	ERROR: Can't write settings file *.qsf. Make sure the *.qsf file is writeable.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.29.2. quartus::open_project (::quartus::project2)

The following table displays information for the `quartus::open_project` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project2 on page 438
Syntax	`quartus::open_project [-h | -help] [-long_help] [-current_revision] [-force] [-preserve_revision_order] [-revision` <*revision_name*> `]` <*project_name*>

continued...

Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-current_revision`	Option to open the current revision automatically
	`-force`	Option to open the project and overwrite the compilation database if the database version is incompatible.
	`-preserve_revision_order`	Option to not move the revision to top in the qpf file and preserve existing revision ordering
	`-revision` *<revision_name>*	Revision name
	<project_name>	Project name

Description

```
Opens an existing project. To create a new project, use the
"project_new" command.

If the "-revision" option is not specified, the project name
is used to open the revision.

The project_open command gives an error when the compilation database
version is not compatible with the current version of Quartus Prime
software.  You may specify the "-force" option to avoid the error and
overwrite the database.
```

Example Usage

```
## Open project "chiptrip" and revision "chiptrip"
quartus::open_project chiptrip

## Open project "chiptrip" and revision "auto_max"
quartus::open_project -revision auto_max chiptrip

## Get the current revision before opening
## the project with the current revision
set project_name chiptrip
set current_revision [get_current_revision $project_name]
quartus::open_project -revision $current_revision $project_name
puts [get_global_assignment -name FAMILY]
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	WARNING: Specified values of options -*<string>* and -*<string>* differ. Ignored value of -*<string>* option. No action is required.
TCL_OK	0	WARNING: Project is already open: *<string>*
TCL_ERROR	1	ERROR: Can't open project: *<string>*. First close the currently open project: *<string>*.
TCL_ERROR	1	ERROR: Can't open project: *<string>*
TCL_ERROR	1	ERROR: Can't set revision: *<string>*. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: *<string>*. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Cannot open project: *<string>*. The project is not compatible with the installed version of the Quartus Prime software. Opening the project will overwrite the old project database. If you wish to overwrite the old project database, make sure to specify the -*<string>* option.
TCL_ERROR	1	ERROR: Can't open revision: *<string>* (project: *<string>*). The revision is not compatible with the installed version of the Quartus Prime software. Opening the revision will

continued...

			overwrite the old revision database. If you wish to overwrite the old revision database, make sure to specify the -<*string*> option.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Revision does not exist: <*string*>. Specify a legal revision name using the -<*string*> option.
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: <*string*>. Specify a legal project name.
	TCL_ERROR	1	ERROR: Project name was not specified or open project does not exist. Open an existing project or specify the project name.

3.1.30. ::quartus::project_ui

The following table displays information for the **::quartus::project_ui** Tcl package:

Tcl Package and Version	::quartus::project_ui 2.0
Description	This package contains no general description.
Availability	This package is loaded by default in the following executables: qpro quartus
Tcl Commands	assignment_group (::quartus::project_ui) on page 442 delete_revision (::quartus::project_ui) on page 444 execute_assignment_batch (::quartus::project_ui) on page 444 export_assignments (::quartus::project_ui) on page 445 get_all_assignment_names (::quartus::project_ui) on page 446 get_all_assignments (::quartus::project_ui) on page 447 get_all_global_assignments (::quartus::project_ui) on page 450 get_all_instance_assignments (::quartus::project_ui) on page 452 get_all_parameters (::quartus::project_ui) on page 454 get_all_quartus_defaults (::quartus::project_ui) on page 456 get_all_user_option_names (::quartus::project_ui) on page 458 get_assignment_info (::quartus::project_ui) on page 458 get_assignment_name_info (::quartus::project_ui) on page 459 get_current_project (::quartus::project_ui) on page 460 get_current_revision (::quartus::project_ui) on page 460 get_global_assignment (::quartus::project_ui) on page 461 get_instance_assignment (::quartus::project_ui) on page 462 get_location_assignment (::quartus::project_ui) on page 463 get_parameter (::quartus::project_ui) on page 464 get_project_directory (::quartus::project_ui) on page 465 get_project_revisions (::quartus::project_ui) on page 465 get_user_option (::quartus::project_ui) on page 466 is_project_open (::quartus::project_ui) on page 467 project_archive (::quartus::project_ui) on page 467 project_close (::quartus::project_ui) on page 468 project_exists (::quartus::project_ui) on page 469 project_new (::quartus::project_ui) on page 470 project_open (::quartus::project_ui) on page 471 project_restore (::quartus::project_ui) on page 472 remove_all_global_assignments (::quartus::project_ui) on page 473 remove_all_instance_assignments (::quartus::project_ui) on page 475 remove_all_parameters (::quartus::project_ui) on page 477 resolve_file_path (::quartus::project_ui) on page 479 revision_exists (::quartus::project_ui) on page 480 set_current_revision (::quartus::project_ui) on page 481 set_global_assignment (::quartus::project_ui) on page 482 set_instance_assignment (::quartus::project_ui) on page 484 set_io_assignment (::quartus::project_ui) on page 486 set_location_assignment (::quartus::project_ui) on page 488 set_parameter (::quartus::project_ui) on page 489 set_power_file_assignment (::quartus::project_ui) on page 490 set_user_option (::quartus::project_ui) on page 492 test_assignment_trait (::quartus::project_ui) on page 493

3.1.30.1. assignment_group (::quartus::project_ui)

The following table displays information for the `assignment_group` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`assignment_group [-h | -help] [-long_help] [-add_exception <name> ] [-add_member <name> ] [-comment <comment> ] [-disable] [-fall] [-get_exceptions] [-get_members] [-overwrite] [-remove] [-remove_exception <name> ] [-remove_member <name> ] [-rise] [-tag <data> ] <group_name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add_exception <name>`	Tcl list of exception names to add
	`-add_member <name>`	Tcl list of member names to add
	`-comment <comment>`	Comment
	`-disable`	Option to disable assignment
	`-fall`	Option applies to falling edge
	`-get_exceptions`	Option to get collection of assignment group exceptions
	`-get_members`	Option to get collection of assignment group members
	`-overwrite`	Option to overwrite existing assignment group with the same group name
	`-remove`	Option to remove assignment group
	`-remove_exception <name>`	Tcl list of exception names to remove
	`-remove_member <name>`	Tcl list of member names to remove
	`-rise`	Option applies to rising edge
	`-tag <data>`	Option to tag data to this assignment
	<group_name>	Assignment group name
Description		

```
Adds, removes, gets members of, or gets exceptions to an
assignment group.

The "assignment_group" command replaces the deprecated
"timegroup" command.

An assignment group is a custom group of registers and pins. You can
use the "-add_member" option to specify register or pin names
you want to include in the assignment group. You can use the
"-add_exception" option to specify names you want to exclude
from the assignment group.

You can specify the names using wildcards, that is, using "?"
or "*". For example, to add all registers and pins that start
with a "b" except those that start with "b|c|" to a particular
assignment group named "group_b", type:

assignment_group "group_b" -add_member "b*" -add_exception "b|c|*"

To remove members or exceptions from a assignment group, use the
"-remove_member" or "-remove_exception" options respectively.

The "-get_members" option returns a collection of members in
the assignment group. The "-get_exceptions" option returns a collection
of exceptions to the assignment group. To access each element of the
collection, use the Tcl command "foreach_in_collection". To see
example usage, type "assignment_group -long_help" or
"foreach_in_collection -long_help".
```

continued...

```
Specifying registers and pins in terms of an assignment group
allows you to set timing constraints easily. For example, to make
a multicycle assignment from nodes "a1" and "a2" to nodes "b1",
"b2", and "b3", type the following:

assignment_group "group_a" -add_member [list "a1" "a2"]
assignment_group "group_b" -add_member [list "b1" "b2" "b3"]

set_multicycle_assignment -from "group_a" -to "group_b" 2

This command sets a multicycle assignment from every member of
"group_a" to every member of "group_b". Quartus Prime timing
analysis is optimized to use assignment groups in handling timing
constraints.

To disable assignment group assignments for the entire group,
use the "-disable" option, for example:

assignment_group "group_a" -disable

To disable a particular assignment group assignment, use the
"-disable" option with the "-add_member" or "-add_exception"
options, for example:

assignment_group "group_a" -add_member "m1" -disable
assignment_group "group_a" -add_exception "e1" -disable

Assignments created or modified by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
# Make timing cut assignment from nodes starting
# with "r" except those starting with "r|s|"
# and except those starting with "r|t|"
# to nodes "t1", "t2", and "t3"
assignment_group "tg1" -add_member "r*" -add_exception "r|s|*"
assignment_group "tg1" -add_exception "r|t|*"

assignment_group "tg2" -add_member [list "t1" "t2" "t3"]

set_timing_cut_assignment -from "group_a" -to "group_b" 2

# Remove the "t1" from a particular assignment group named "tg2"
assignment_group "tg2" -remove_member "t1"

# Display the members of a particular assignment group named "tg1"
foreach_in_collection member [assignment_group "tg1" -get_members] {

    # Print the name of the member
    puts $member
}

# Display the exceptions to a particular assignment group named "tg1"
foreach_in_collection exception [assignment_group "tg1" -get_exceptions] {

    # Print the name of the exception
    puts $exception
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't set revision: <*string*>. Make sure there is an open, active revision name.

continued...

	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Revision does not exist: <*string*>. Specify a legal revision name using the -<*string*> option.

3.1.30.2. delete_revision (::quartus::project_ui)

The following table displays information for the `delete_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`delete_revision [-h | -help] [-long_help] <revision_name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<*revision_name*>	Revision name	
Description	`Deletes the specified revision from the current project. The corresponding <revision name>.qsf file is deleted as well.`		
Example Usage	`## Delete the revision called "tmp"` `delete_revision tmp`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't delete the current revision: <*string*>. Specify a different revision name.
	TCL_ERROR	1	ERROR: Can't delete revision because it is not included in the current project: <*string*> . Specify a revision name that is included in the project.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.30.3. execute_assignment_batch (::quartus::project_ui)

The following table displays information for the `execute_assignment_batch` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`execute_assignment_batch [-h | -help] [-long_help] <tcl commands>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*tcl commands*>	Tcl list of Tcl commands
Description	`Iterates through the specified Tcl list of Tcl commands and executes each command sequentially in batch mode.` `In batch mode, Tcl commands that set Quartus Prime Settings File (.qsf) assignments are optimized to prevent them from repeatedly`	

continued...

	write-locking and write-unlocking the QSF during consecutive calls, thereby slowing down the execution. Currently, only the following commands are supported: assignment_group remove_all_global_assignments remove_all_instance_assignments remove_all_parameters set_global_assignment set_instance_assignment set_io_assignment set_location_assignment set_parameter set_power_file_assignment
Example Usage	`project_open one_wire` `set tcl_cmds [list [list set_global_assignment -name FAMILY StratixII] \` `[list set_global_assignment -name DEVICE AUTO] \` `[list set_global_assignment -name TOP_LEVEL_ENTITY one_wire] \` `[list set_global_assignment -name SAVE_DISK_SPACE OFF] \` `[list set_location_assignment PIN_1 -to in1] \` `[list set_instance_assignment -name MULTICYCLE 4 -from in1 -to out1] \` `[list set_parameter -name STYLE FAST]]` `execute_assignment_batch $tcl_cmds` `project_close`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Unsupported Tcl command: <*string*>. Specify one of the supported Tcl commands listed in the help description for <*string*> -h.

3.1.30.4. export_assignments (::quartus::project_ui)

The following table displays information for the `export_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`export_assignments [-h | -help] [-long_help] [-reorganize]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-reorganize`	Option to reorganize the Quartus Prime Settings File (.qsf)
Description	Exports assignments for the current revision to the Quartus Prime Settings File (.qsf). Assignments created or modified during an open project are not saved to the Quartus Prime Settings File (.qsf) unless you explicitly call one of the following two Tcl commands: 1) export_assignments 2) project_close (unless "-dont_export_assignments" is specified) These two Tcl commands reside in the ::quartus::project Tcl package. You must save assignment changes before you run Quartus Prime command-line executables. Note, however, that the Tcl commands "execute_flow" and "execute_module" (part of the ::quartus::flow Tcl package) automatically call "export_assignments" before they run command-line executables.	
Example Usage	`## The most common use of export_assignments is to` `## call it before doing a system call` `## to call a compiler command-line executable` `project_open $project_name` `set_global_assignment -name FAMILY Stratix` `## Before calling quartus_map,`	

continued...

```
## write out the FAMILY assignment
export_assignments

## Now, call quartus_map
qexec "[file join $::quartus(binpath) quartus_map] $project_name"
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.30.5. get_all_assignment_names (::quartus::project_ui)

The following table displays information for the `get_all_assignment_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_all_assignment_names [-h | -help] [-long_help] [-family` <*family*> `] [-module` <*all*|*ip_generate*|*map*|*tlg*|*fit*|*tan*|*asm*|*eda*|*drc*|*power*|*generic*> `] [-type` <*all*|*global*|*instance*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-family` <*family*>	Option to filter based on the specified device family. Defaults to all families.
	`-module` <*all*|*ip_generate*|*map*|*tlg*|*fit*|*tan*|*asm*|*eda*|*drc*|*power*|*generic*>	Option to filter based on the specified flow module. Defaults to all.
	`-type` <*all*|*global*|*instance*>	Option to filter based on the specified assignment type. Defaults to all.
Description		

```
Returns a filtered output list of all available, matching
assignment names.

The module option takes one of the following values:

Module           Description
--------         -------------------------------------
ip_generate      IP Generation assignment names
tlg              Support Logic Generation assignment names
map              Analysis & Synthesis assignment names
fit              Fitter assignment names
asm              Assembler assignment names
eda              EDA Netlist Writer assignment names
drc              Design Assistant assignment names
power            Power Analyzer assignment names
generic          Other assignment names not included in any
                 of the above flow modules
tan              Classic Timing Analyzer assignment names
all              All assignment names
```

Example Usage

```
## Print out all available global assignments
foreach i [get_all_assignment_names -type global] {
    puts $i
}
```

continued...

```
## Print out all available global assignments
## for the Stratix family
foreach i [get_all_assignment_names -type global -family Stratix] {
    puts $i
}

## Print out all available global assignments
## for the Stratix family required
## by the Analysis & Synthesis module
foreach i [get_all_assignment_names -type global -family Stratix

-module map] {
    puts $i
}
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal flow module: <*string*>. Specify <*string*>, <*string*>, <*string*>, <*string*>, <*string*>, <*string*>, or <*string*>.
	TCL_ERROR	1	ERROR: Illegal type: <*string*>. Specify <*string*>, <*string*>, or <*string*>.
	TCL_ERROR	1	ERROR: Illegal device family: <*string*>. Specify a legal device family.

3.1.30.6. get_all_assignments (::quartus::project_ui)

The following table displays information for the `get_all_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_all_assignments [-h | -help] [-long_help] [-entity` <*entity_name*> `] [-fall] [-from` <*source*> `] -name` <*name*> `[-rise] [-section_id` <*section id*> `] [-tag` <*data*> `] [-to` <*destination*> `] -type` <*global|instance|parameter|default*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity` <*entity_name*>	Entity name
	`-fall`	Option applies to falling edge
	`-from` <*source*>	Source name (string pattern is matched using Tcl string matching)
	`-name` <*name*>	Assignment name (string pattern is matched using Tcl string matching)
	`-rise`	Option applies to rising edge
	`-section_id` <*section id*>	Section id
	`-tag` <*data*>	Option to tag data to this assignment
	`-to` <*destination*>	Destination name (string pattern is matched using Tcl string matching)
	`-type` <*global|instance|parameter|default*>	Option to specify the type of assignments to return
Description		

```
Returns a collection of all matching global, instance, parameter,
or default assignment ids. To iterate through each assignment id
in this collection, use the Tcl command "foreach_in_collection".
```

continued...

```
To view details for the assignment that is associated with the
assignment id, use the Tcl command "get_assignment_info".

The "get_all_assignments" command is easier to use than the
deprecated commands listed in Table 1.

* Table 1. The -type Option

Value for
-type Option  Deprecated Tcl command        Description
------------  ---------------------        -----------
default       get_all_quartus_defaults      Returns only default assignments.

global        get_all_global_assignments    Returns only global assignments.

instance      get_all_instance_assignments  Returns only instance assignments.

parameter      get_all_parameters            Returns only parameter assignments.

The "-name" option is not case sensitive.
The "-to" and "-from" options are case sensitive.

These options can take string patterns containing special
characters from the set "*?\[]" as values. The values are
matched using Tcl string matching. Note that bus names are
automatically detected and do not need to be escaped. Bus
names have the following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
get_all_assignments -type instance -name LOCATION -to a[0]

# To match all indices of bus "a", type:
get_all_assignments -type instance -name LOCATION -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

get_all_assignments -type instance LOCATION -to "a[escape_brackets \[]\[0-2\][escape_brackets
\]]"

For more information about escaping square brackets, type
"escape_brackets -h".

This Tcl command reads in the global, instance, and parameter
assignments found in the Quartus Prime Settings File (.qsf) and
reads in the default assignments found inside the
Quartus Prime Default Settings File (.qdf).

If you tagged data by making assignments with the -tag option,
then the information can be searched using the -tag option.

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one clock section
each containing its own set of clock assignments. To uniquely
identify sections of this type, use the -section_id option.

For entity-specific assignments, use the "-entity" option to
retrieve assignments from a specific entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.
```

Example Usage

```
    ## View all the timing requirements using wildcards
    ## to match TSU_REQUIREMENT, TCO_REQUIREMENT,
    ## and others.
foreach_in_collection asgn_id [get_all_assignments -type instance -name *_REQUIREMENT] {

    set from   [get_assignment_info $asgn_id -from]
    set to     [get_assignment_info $asgn_id -to]
    set name   [get_assignment_info $asgn_id -name]
    set value  [get_assignment_info $asgn_id -value]
    set entity [get_assignment_info $asgn_id -entity]
    set sid    [get_assignment_info $asgn_id -section_id]
    set tag    [get_assignment_info $asgn_id -tag]
```

continued...

```
    puts "$entity: $name ($from -> $to) = $value"
}

    ## View all global assignments
foreach_in_collection asgn_id [get_all_assignments -type global -name *] {

    set name   [get_assignment_info $asgn_id -name]
    set value  [get_assignment_info $asgn_id -value]
    set entity [get_assignment_info $asgn_id -entity]
    set sid    [get_assignment_info $asgn_id -section_id]
    set tag    [get_assignment_info $asgn_id -tag]

    puts "$entity: $name = $value"
}

    ## View all project-wide default parameter values
foreach_in_collection asgn_id [get_all_assignments -type parameter -name *] {

    set name  [get_assignment_info $asgn_id -name]
    set value [get_assignment_info $asgn_id -value]
    set tag    [get_assignment_info $asgn_id -tag]

    puts "$name = $value"
}

    ## View all entity-specific parameter values
foreach_in_collection asgn_id [get_all_assignments -type parameter -name * -to *] {

    set dest  [get_assignment_info $asgn_id -to]
    set name  [get_assignment_info $asgn_id -name]
    set value [get_assignment_info $asgn_id -value]
    set tag    [get_assignment_info $asgn_id -tag]

    puts "$name (-> $dest) = $value"
}

    ## View all default assignments
foreach_in_collection asgn_id [get_all_assignments -type default -name * -to *] {

    set name    [get_assignment_info $asgn_id -name]
    set value   [get_assignment_info $asgn_id -value]

    puts "$name = $value"
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not an instance assignment: <*string*> -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Illegal assignment type: <*string*>. Specify <*string*>, <*string*>, <*string*>, or <*string*>.

continued...

TCL_ERROR	1	ERROR: Illegal option <*string*>. The specified option is illegal for <*string*> assignments.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.7. get_all_global_assignments (::quartus::project_ui)

The following table displays information for the `get_all_global_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_all_global_assignments [-h | -help] [-long_help] [-entity <entity_name> ] -name <name> [-section_id <section id> ] [-tag <data> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which assignment belongs
	`-name <name>`	Assignment name (string pattern is matched using Tcl string matching)
	`-section_id <section id>`	Section id
	`-tag <data>`	Option to tag data to this assignment
Description		

```
Returns a filtered output collection of all matching global
assignment values. To access each element of the output
collection, use the Tcl command "foreach_in_collection". To
see example usage, type "foreach_in_collection -long_help".

The "-name" option is not case sensitive. This option can take
string patterns containing special characters from the set
"*?\[]" as the value. The value is matched using Tcl string
matching.

This Tcl command reads the global assignments found in
the Quartus Prime Settings File (.qsf). This Tcl command filters
the assignment data in the .qsf and outputs the data based
on the values given by the "-name" option.

Each element of the collection is a list with the following
format:
{ {<Section Id>} {<Assignment name>} {<Assignment value>} }

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
```

continued...

```
the associated <Section Id> description:

Section Id                    Description
---------------------------   -----------------------------
CHIP                          Same as revision name
LOGICLOCK_REGION              A unique name
EDA_TOOL_SETTINGS             A unique name
CLIQUE                        A unique name
BREAKPOINT                    A unique name
CLOCK                         A unique name
AUTO_INSERT_SLD_NODE_ENTITY   A unique name

For entity-specific assignments, use the "-entity" option to
retrieve the assignment(s) from the specified entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.
```

Example Usage

```
## Print out all the registered source files
## using the foreach_in_collection method
set file_asgn_col [get_all_global_assignments -name SOURCE_FILE]
foreach_in_collection file_asgn $file_asgn_col {

    ## Each element in the collection has the following
    ## format: {} {SOURCE_FILE} {<file_name>}
    puts [lindex $file_asgn 2]
}

# Using wildcards
get_all_global_assignments -name SOURCE*
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.

continued...

	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.8. get_all_instance_assignments (::quartus::project_ui)

The following table displays information for the `get_all_instance_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_all_instance_assignments [-h | -help] [-long_help] [-entity <entity_name> ] [-from <source> ] -name <name> [-section_id <section id> ] [-tag <data> ] [-to <destination> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which assignment belongs
	`-from <source>`	Source of assignment (string pattern is matched using Tcl string matching)
	`-name <name>`	Assignment name (string pattern is matched using Tcl string matching)
	`-section_id <section id>`	Section id
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of assignment (string pattern is matched using Tcl string matching)
Description		

```
Returns a filtered output collection of all matching instance
assignment values. To access each element of this output
collection, use the Tcl command "foreach_in_collection". To
see example usage, type "foreach_in_collection -long_help".

The "-name" option is not case sensitive.
The "-to" and "-from" options are case sensitive.

These options can take string patterns containing special
characters from the set "*?\[]" as values. The values are
matched using Tcl string matching. Note that bus names are
automatically detected and do not need to be escaped. Bus
names have the following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
get_all_instance_assignments -name LOCATION -to a[0]

# To match all indices of bus "a", type:
get_all_instance_assignments -name LOCATION -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

get_all_instance_assignments -name LOCATION -to "a[escape_brackets \[]\[0-2\][escape_brackets
```

continued...

```
\]]"

For more information about escaping square brackets, type
"escape_brackets -h".

This Tcl command reads in the instance assignments found in
the Quartus Prime Settings File (.qsf). This Tcl command filters
the assignments data found in the .qsf and outputs the
data based on the values specified by the "-name", "-from",
and "-to" options.

Each element of the collection is a list with the following
format:
{ {<Section Id>} {<Source>} {<Destination>} {<Assignment name>} {<Assignment value>} }

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                    Description
---------------------------   ------------------------------
CHIP                          Same as revision name
LOGICLOCK_REGION              A unique name
EDA_TOOL_SETTINGS             A unique name
CLIQUE                        A unique name
BREAKPOINT                    A unique name
CLOCK                         A unique name
AUTO_INSERT_SLD_NODE_ENTITY   A unique name

For entity-specific assignments, use the "-entity" option to
retrieve the assignment(s) from the specified entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.
```

Example Usage

```
## Print out all the timing requirements
## using the foreach_in_collection method
## Use wildcards to catch TSU_REQUIREMENT, TCO_REQUIREMENT,
## and others
set asgn_col [get_all_instance_assignments -name *_REQUIREMENT]
foreach_in_collection asgn $asgn_col {

    ## Each element in the collection has the following
    ## format: { {} {<Source>} {<Destination>} {<Assignment name>} {<Assignment value>} }
    set from  [lindex $asgn 1]
    set to    [lindex $asgn 2]
    set name  [lindex $asgn 3]
    set value [lindex $asgn 4]
    puts "$name ($from -> $to) = $value"
}

## Print out all the location assignments with
## the destination bus name "timeo".
set bus_name "timeo"
get_all_instance_assignments -name LOCATION -to $bus_name[*]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not an instance assignment: <*string*> -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.

continued...

	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.9. get_all_parameters (::quartus::project_ui)

The following table displays information for the `get_all_parameters` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_all_parameters [-h | -help] [-long_help] [-entity <entity_name> ] -name <name> [-tag <data> ] [-to <destination> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which parameter belongs
	`-name <name>`	Parameter name (string pattern is matched using Tcl string matching)
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of the parameter (string pattern is matched using Tcl string matching)
Description		

```
Returns a filtered output collection of all matching parameter
values. To access each element of this output collection, use
the Tcl command "foreach_in_collection". To see example usage,
type "foreach_in_collection -long_help".

The "-name" option is not case sensitive.
The "-to" option is case sensitive.

If the "-to" argument is specified, the function returns
the parameter values for the current entity. The values are
retrieved from the PARAMETERS section of the entity. Otherwise,
the function returns the project-wide default parameter values
obtained from the DEFAULT_PARAMETERS section.

This Tcl command filters the parameter data found in the
Quartus Prime Settings File (.qsf) and outputs the data based on
the values specified by the "-name" and "-to" options. These
```

continued...

```
options can take string patterns containing special characters
from the set "*?\[]" as values. The values are matched using
Tcl string matching. Note that bus names are automatically
detected and do not need to be escaped. Bus names have the
following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
get_all_parameters -name * -to a[0]

# To match all indices of bus "a", type:
get_all_parameters -name * -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

get_all_parameters -name * -to "a[escape_brackets \[]\[0-2\][escape_brackets \]]"

For more information about escaping square brackets, type
"escape_brackets -h".

Each element of the collection is a list with the following
format:
{ {<Destination>} {<Parameter name>} {<Parameter value>} }

Use the "-entity" option to retrieve the parameter values from
the specified entity. If the "-entity" option is not specified,
the value for the FOCUS_ENTITY_NAME assignment is used. If the
FOCUS_ENTITY_NAME value is not found, the revision name is used.
```

Example Usage

```
## Display all project-wide default parameter values
foreach_in_collection parameter [get_all_parameters -name *] {

    set name [lindex $parameter 1]
    set value [lindex $parameter 2]

    ## Now, display the content of the parameter
    puts "Parameter Name ($name)"
    puts "Parameter Value ($value)"
}

## Display all entity-specific parameter values
foreach_in_collection parameter [get_all_parameters -name * -to *] {

    set dest [lindex $parameter 0]
    set name [lindex $parameter 1]
    set value [lindex $parameter 2]

    ## Now, display the content of the parameter
    puts "Destination ($dest)"
    puts "Parameter Name ($name)"
    puts "Parameter Value ($value)"
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.

continued...

	TCL_ERROR	1	ERROR: Illegal default parameter: <*string*>. Specify a legal default parameter name.
	TCL_ERROR	1	ERROR: Illegal parameter: <*string*>. Specify a legal parameter name.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.10. get_all_quartus_defaults (::quartus::project_ui)

The following table displays information for the `get_all_quartus_defaults` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_all_quartus_defaults [-h | -help] [-long_help] [-name` <*name*> `] [-section_id` <*section id*> `] [-tag` <*data*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name` <*name*>	Assignment name (string pattern is matched using Tcl string matching)
	`-section_id` <*section id*>	Section id
	`-tag` <*data*>	Option to tag data to this assignment
Description		

```
Returns a filtered output collection of all matching default
assignment values. To access each element of the output
collection, use the Tcl command "foreach_in_collection". To
see example usage, type "foreach_in_collection -long_help".

The "-name" option is not case sensitive. This option can take
string patterns containing special characters from the set
"*?\[]" as the value. The value is matched using Tcl string
matching.

This Tcl command reads in the default assignments found inside
the Quartus Prime Default Settings File (.qdf). It filters the
assignments data found inside the .qdf and outputs the data
based on the values specified by the "-name" option.

Each element of the collection is a list with the following
format:
{ {<Section Id>} {<Assignment name>} {<Assignment value>} }

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                    Description
---------------------------   -----------------------------
CHIP                          Same as revision name
LOGICLOCK_REGION              A unique name
EDA_TOOL_SETTINGS             A unique name
CLIQUE                        A unique name
BREAKPOINT                    A unique name
CLOCK                         A unique name
AUTO_INSERT_SLD_NODE_ENTITY   A unique name
```

Example Usage

```
## Print out all the default assignments using
## the foreach_in_collection method

set default_asgns_col [get_all_quartus_defaults]
foreach_in_collection default $default_asgns_col {
    set sect_id [lindex $default 0]
```

continued...

```
    set name [lindex $default 1]
    set value [lindex $default 2]

    ## Now, display the content of the assignment
    puts "Section ID ($sect_id)"
    puts "Assignment Name ($name)"
    puts "Assignment Value ($value)"
}

## Using wildcards
set default_asgns_col [get_all_quartus_defaults -name *]
foreach_in_collection default $default_asgns_col {
    set sect_id [lindex $default 0]
    set name [lindex $default 1]
    set value [lindex $default 2]

    ## Now, display the content of the assignment
    puts "Section ID ($sect_id)"
    puts "Assignment Name ($name)"
    puts "Assignment Value ($value)"
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.11. get_all_user_option_names (::quartus::project_ui)

The following table displays information for the `get_all_user_option_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_all_user_option_names [-h | -help] [-long_help] [-name <name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	User option name (string pattern is matched using Tcl string matching)
Description		

```
Returns a filtered output list of all available, matching
user option names.

If the "-name" option is not specified, all available user
option names are returned. Otherwise, only the matching
user option names are returned.

The "-name" option is not case sensitive. This option can take
string patterns containing special characters from the set
"*?\[]" as the value. The value is matched using Tcl string
matching.
```

Example Usage

```
## Print out all available user option names
foreach i [get_all_user_option_names] {
    puts $i
}

## Display all user option names that contain
## the word "talkback" and also display the
## value for each of the user option names
foreach i [get_all_user_option_names -name *talkback*] {
    set name $i
    set value [get_user_option -name $i]
    puts "$name = $value"
}
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.30.12. get_assignment_info (::quartus::project_ui)

The following table displays information for the `get_assignment_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_assignment_info [-h | -help] [-long_help] [-comments] [-entity] [-from] [-get_tcl_command] [-name] [-section_id] [-tag] [-to] [-value] <asgn_id>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comments`	Option to get the assignment comment
	`-entity`	Option to get the assignment entity
	`-from`	Option to get the assignment source
	`-get_tcl_command`	Option to get the tcl command that sets the assignment

continued...

	-name		Option to get the assignment name
	-section_id		Option to get the assignment section id
	-tag		Option to get the assignment tag
	-to		Option to get the assignment destination
	-value		Option to get the assignment value
	<*asgn_id*>		Assignment id
Description	Returns information for the assignment id based on the specified option. The assignment id is obtained from the "get_all_assignments" Tcl command.		
Example Usage	```## View all the instance assignments foreach_in_collection asgn_id [get_all_assignments -type instance -name *] { set from [get_assignment_info $asgn_id -from] set to [get_assignment_info $asgn_id -to] set name [get_assignment_info $asgn_id -name] set value [get_assignment_info $asgn_id -value] set entity [get_assignment_info $asgn_id -entity] set tag [get_assignment_info $asgn_id -tag] puts "$entity: $name ($from -> $to) = $value" }```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal assignment id: <*string*>. Specify a legal assignment id that was retrieved from the Tcl command get_all_assignments.

3.1.30.13. get_assignment_name_info (::quartus::project_ui)

The following table displays information for the `get_assignment_name_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`get_assignment_name_info [-h | -help] [-long_help] <name>`		
Arguments	-h | -help		Short help
	-long_help		Long help with examples and possible return values
	<*name*>		Assignment name
Description	Returns information for the specified assignment name.		
Example Usage	```## View information for all assignment names foreach name [get_all_assignment_names] { puts [get_assignment_name_info $name] }```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.

3.1.30.14. get_current_project (::quartus::project_ui)

The following table displays information for the `get_current_project` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`get_current_project [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns the name of the current project.`		
Example Usage	`# Get the current name for` `# the currently open project "chiptrip"` `project_open chiptrip` `set project_name [get_current_project]` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.30.15. get_current_revision (::quartus::project_ui)

The following table displays information for the `get_current_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`get_current_revision [-h | -help] [-long_help] [ <project_name> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<project_name>		Project name
Description	`Returns the name of the current revision for the specified` `project. If the project name is not specified, the current` `project name is used.`		
Example Usage	`# Get the current revision name for` `# the currently open project "chiptrip"` `project_open chiptrip` `set revision_name [get_current_revision]` `project_close` `# Get the current revision name for` `# a project that is not currently open` `set revision_name [get_current_revision chiptrip]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: *<string>*. Specify a legal project name.
	TCL_ERROR	1	ERROR: Project name was not specified or open project does not exist. Open an existing project or specify the project name.

3.1.30.16. get_global_assignment (::quartus::project_ui)

The following table displays information for the `get_global_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`get_global_assignment [-h | -help] [-long_help] [-entity <entity_name> ] [-front] -name <name> [-section_id <section id> ] [-tag <data> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-entity <entity_name>`	Entity to which assignment belongs	
	`-front`	Option to return the first assignment if there is more than one assignment found	
	`-name <name>`	Assignment name	
	`-section_id <section id>`	Section id	
	`-tag <data>`	Option to tag data to this assignment	
Description	`Returns the value of the global assignment.` `The "-name" option is not case sensitive.` `For entity-specific assignments, use the "-entity" option to retrieve the assignment from the specified entity. If the "-entity" option is not specified, the value for the FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME value is not found, the revision name is used.`		
Example Usage	`## Get the value of the FAMILY assignment` `get_global_assignment -name FAMILY`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
	TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
	TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.

continued...

	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.
	TCL_ERROR	1	ERROR: Assignment <*string*> has multiple values. Use the <*string*> command to get all values or use the <*string*> -front command to get the first value.

3.1.30.17. get_instance_assignment (::quartus::project_ui)

The following table displays information for the `get_instance_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_instance_assignment [-h | -help] [-long_help] [-entity <entity_name> ] [-from <source> ] [-front] -name <name> [-section_id <section id> ] [-tag <data> ] [-to <destination> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which assignment belongs
	`-from <source>`	Source of assignment
	`-front`	Option to return the first assignment if there is more than one assignment found
	`-name <name>`	Assignment name
	`-section_id <section id>`	Section id
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of assignment
Description		

```
Returns the value of the instance assignment.

The "-name" option is not case sensitive.  The
"-entity", "-to", and "-from" options are case
sensitive.

For entity-specific assignments, use the "-entity"
option to retrieve the assignment from the specified
entity. If the "-entity" option is not specified, the
value for the FOCUS_ENTITY_NAME assignment is used. If
the FOCUS_ENTITY_NAME value is not found, the revision
name is used.
```

continued...

Example Usage	```## Get the TSU_REQUIREMENT from mypin to any register set value [get_instance_assignment -from "mypin" -to * -name TSU_REQUIREMENT] puts "TSU_REQUIREMENT(mypin->*) = $value"```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
	TCL_ERROR	1	ERROR: Assignment is not an instance assignment: <*string*> -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
	TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.
	TCL_ERROR	1	ERROR: Assignment <*string*> has multiple values. Use the <*string*> command to get all values or use the <*string*> -front command to get the first value.

3.1.30.18. get_location_assignment (::quartus::project_ui)

The following table displays information for the `get_location_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_location_assignment [-h | -help] [-long_help] [-tag <data> ] -to <destination>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-tag <data>`	Option to tag data to this assignment

continued...

	`-to <destination>`		Destination of assignment
Description	`Returns the value of a location assignment.` `The "-chip" option is not case sensitive.` `The "-to" option is case sensitive.`		
Example Usage	`get_location_assignment -to dst`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.30.19. get_parameter (::quartus::project_ui)

The following table displays information for the `get_parameter` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`get_parameter [-h | -help] [-long_help] [-entity <entity_name> ] -name <name> [-tag <data> ] [-to <destination> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-entity <entity_name>`		Entity to which parameter belongs
	`-name <name>`		Parameter name
	`-tag <data>`		Option to tag data to this assignment
	`-to <destination>`		Destination of parameter
Description	`Returns the value of the parameter.` `The "-name" option is not case sensitive.` `The "-to" option is case sensitive.` `If the "-to" argument is specified, the function returns the parameter value for the current entity. The value is retrieved from the PARAMETERS section of the entity. Otherwise, the function returns the project-wide default parameter value obtained from the DEFAULT_PARAMETERS section.` `Use the "-entity" option to retrieve the parameter from the specified entity. If the "-entity" option is not specified, the value for the FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME value is not found, the revision name is used.`		
Example Usage	`## Get project-wide, default parameter value` `get_parameter -name WIDTH` `## Get entity-specific parameter value` `get_parameter -name inst1 -to SIZE`		
Return Value	**Code Name**	**Code**	**String Return**

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Illegal default parameter: <*string*>. Specify a legal default parameter name.
	TCL_ERROR	1	ERROR: Illegal parameter: <*string*>. Specify a legal parameter name.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.20. get_project_directory (::quartus::project_ui)

The following table displays information for the `get_project_directory` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`get_project_directory [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Returns the project directory for currently open project.`		
Example Usage	`project_open one_wire` `  # Print the current project directory` `puts [get_project_directory]` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.30.21. get_project_revisions (::quartus::project_ui)

The following table displays information for the `get_project_revisions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_project_revisions [-h | -help] [-long_help] [ <project_name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*project_name*>	Project name

continued...

Description	<pre>Returns a list of revisions included in the specified project. If the project name is not specified, the current project name is used by default. The first element in the list of revisions is the current revision and is the same as the return value for the "get_current_revision" command.</pre>		
Example Usage	<pre># Set the device family assignment to Stratix # for all revisions project_open chiptrip set original_revision [get_current_revision] foreach revision [get_project_revisions] { puts "$revision" set_current_revision $revision set_global_assignment -name FAMILY Stratix export_assignments } set_current_revision $original_revision project_close # Open the project with the first available revision # and set the device family assignment to Stratix set revision [lindex [get_project_revisions chiptrip] 0] open_project -revision $revision chiptrip set_global_assignment -name FAMILY Stratix project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: <*string*>. Specify a legal project name.
	TCL_ERROR	1	ERROR: Project name was not specified or open project does not exist. Open an existing project or specify the project name.

3.1.30.22. get_user_option (::quartus::project_ui)

The following table displays information for the `get_user_option` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`get_user_option [-h | -help] [-long_help] -name <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	User option name
Description	<pre>Returns the user option value for the name specified by the "-name" option. To get a list of all available user option names, use the "get_all_user_option_names" command.</pre>	
Example Usage	<pre>## Get the value for the user option ## "TALKBACK_ENABLED" set value [get_user_option -name TALKBACK_ENABLED] puts "TALKBACK_ENABLED = $value"</pre>	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal user option name: <*string*>. Specify a legal user option name. To get a list of legal names, use the get_all_user_option_names command.

3.1.30.23. is_project_open (::quartus::project_ui)

The following table displays information for the `is_project_open` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`is_project_open [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Checks whether a project is currently open. Returns 1, if a project is currently open; returns 0, otherwise.`	
Example Usage	`## Close the project if open` `if [is_project_open] {` `  project_close` `}`	

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.30.24. project_archive (::quartus::project_ui)

The following table displays information for the `project_archive` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`project_archive [-h | -help] [-long_help] [-all_revisions] [-include_libraries] [-include_outputs] [-overwrite] [-use_file_set` <*file_set*> `] [-version_compatible_database]` <*archive_name*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all_revisions`	Option to archive all revisions
	`-include_libraries`	Option to include related system libraries
	`-include_outputs`	Option to include output files in archive
	`-overwrite`	Option to overwrite any currently existing archive file
	`-use_file_set` <*file_set*>	Option to create the archive using the specified file set
	`-version_compatible_database`	Option to include version-compatible database if supported
	<*archive_name*>	Archive file name
Description	`Archives an open project and its related files into a Quartus Prime Archive File (.qar).`	

continued...

```
The description of operations is as follows:

Option                          Description
------                          -----------------------
use_file_set                    Creates the archive using the specified
                                file set. By default, the 'basic'
                                file set is used. For more information
                                about file sets, type: quartus_sh --archive -list_file_sets

all_revisions                   Archives all revisions.

overwrite                       Overwrites existing archive file.

include_outputs                 Includes output files in archive.

include_libraries               Includes related Megafunction and
                                IP library files.

version_compatible_database     Includes version-compatible database
                                if supported.
```

Example Usage

```
## Default mode: Archive current revisions without output files or libraries
project_archive chiptrip.qar

## Archive all revisions without output files or libraries
project_archive chiptrip.qar -all_revisions

## Archive current revision with version-compatible database if supported
project_archive chiptrip.qar -version_compatible_database

## Same as first one, but overwrite any existing archive file
project_archive chiptrip.qar -overwrite

## Include outut files and libraries
project_archive chiptrip.qar -include_outputs -include_libraries
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Error(s) found while archiving the project. See error message(s) for details.
TCL_ERROR	1	ERROR: Project archive failed. Some files could not be processed. Refer to the Quartus Prime Archive Log File (<archive_name>.qarlog).
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: *<string>*. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.30.25. project_close (::quartus::project_ui)

The following table displays information for the `project_close` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`project_close [-h | -help] [-long_help] [-dont_export_assignments]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-dont_export_assignments`	Do not export assignments to file
Description		

```
Closes an open project.

The assignments created or modified during an open project
```

continued...

	`` are committed to the Quartus Prime Settings File (.qsf) during a "project_close", unless you use the "-dont_export_assignments" option. ```		
Example Usage	``` ## Close the project if open if [is_project_open] { project_close } ## Close the project if open ## and do not export the assignments if [is_project_open] { project_close -dont_export_assignments } ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.30.26. project_exists (::quartus::project_ui)

The following table displays information for the `project_exists` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`project_exists [-h | -help] [-long_help] <project_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	<*project_name*>		Project name
Description	``` Checks whether a project exists. Returns 1, if a project exists; returns 0, otherwise. ```		
Example Usage	``` ## Create project if one does not exist. ## Open existing project otherwise. if [project_exists chiptrip] { project_open chiptrip } else { project_new chiptrip } ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.30.27. project_new (::quartus::project_ui)

The following table displays information for the `project_new` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`project_new [-h | -help] [-long_help] [-family` *<family>* `] [-overwrite] [-part` *<part>* `] [-revision` *<revision_name>* `] <project_name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-family` *<family>*	Family name	
	`-overwrite`	Option to overwrite existing project and revision	
	`-part` *<part>*	Part name	
	`-revision` *<revision_name>*	Revision name	
	<project_name>	Project name	
Description	`Creates and opens a new project with the specified project name.` `If the "-revision" option is not specified, the project name` `is used to create the revision.` `Assignments created or modified by using this Tcl command are` `not saved to the Quartus Prime Settings File (.qsf) unless you` `explicitly call one of the following two Tcl commands:` `1) export_assignments` `2) project_close (unless "-dont_export_assignments" is specified)` `These two Tcl commands reside in the ::quartus::project Tcl` `package. You must save assignment changes before you run` `Quartus Prime command-line executables. Note, however, that` `the Tcl commands "execute_flow" and "execute_module" (part` `of the ::quartus::flow Tcl package) automatically call` `"export_assignments" before they run command-line executables.`		
Example Usage	`## Create project "chiptrip" and revision "chiptrip"` `project_new chiptrip` `## Create project "chiptrip" and revision "auto_max"` `project_new -revision auto_max chiptrip` `## Create project "chiptrip" and revision "chiptrip"` `## Overwrite any Quartus Prime Settings File (.qsf) if it exists` `project_new chiptrip -overwrite` `## Create project "chiptrip" and revision "chiptrip"` `## Set the FAMILY assignment to Stratix` `project_new chiptrip -family Stratix`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The -*<string>* option must also be used when you use the -*<string>* option. Specify both options.
	TCL_ERROR	1	ERROR: Can't create project because device and family are mismatch: *<string>* and *<string>*.
	TCL_ERROR	1	ERROR: Can't create project because device is not installed: *<string>*.
	TCL_ERROR	1	ERROR: Can't create project because family requires specifying the part: *<string>*.

continued...

	TCL_ERROR	1	ERROR: Can't create project: <*string*>. Specify a legal project name.
	TCL_ERROR	1	ERROR: Can't create revision: <*string*>. Specify a legal revision name using the -<*string*> option.
	TCL_ERROR	1	ERROR: Can't create revision: <*string*>. Specify a legal revision name.
	TCL_ERROR	1	ERROR: Can't create settings files for project: <*string*>. Make sure the .psf, .csf, and .ssf files are writeable.
	TCL_ERROR	1	ERROR: Can't open project: <*string*>
	TCL_ERROR	1	ERROR: Can't remove Quartus Prime Settings File: <*string*>. Make sure the file is writeable.
	TCL_ERROR	1	ERROR: Can't create project with unknown device: <*string*>.
	TCL_ERROR	1	ERROR: Can't create project with unknown device family: <*string*>.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Project already exists: <*string*>. Specify a different project name or use the -overwrite option.

3.1.30.28. project_open (::quartus::project_ui)

The following table displays information for the `project_open` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`project_open [-h | -help] [-long_help] [-current_revision] [-error_on_incompatible_database] [-revision` <*revision_name*> `]` <*project_name*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-current_revision`	Option to open the current revision automatically
	`-error_on_incompatible_database`	Option not to open the project and issue and error if the database version is incompatible
	`-revision` <*revision_name*>	Revision name
	<*project_name*>	Project name
Description		

```
Opens an existing project. To create a new project, use the
"project_new" command.

If the "-revision" option is not specified, the project name
is used to open the revision.

By default, opening the project overwrites the database
created in a different version of the Quartus Prime software.
However, if the "-error_on_incompatible_database" option is
```

continued...

	specified, instead of opening the project, an error is issued when the database version differs from the current version of the Quartus Prime software.
Example Usage	## Open project "chiptrip" and revision "chiptrip" project_open chiptrip ## Open project "chiptrip" and revision "auto_max" project_open -revision auto_max chiptrip ## Get the current revision before opening ## the project with the current revision set project_name chiptrip set current_revision [get_current_revision $project_name] project_open -revision $current_revision $project_name puts [get_global_assignment -name FAMILY] project_close

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	WARNING: Project is already open: <*string*>
	TCL_ERROR	1	ERROR: Can't open project: <*string*>. First close the currently open project: <*string*>.
	TCL_ERROR	1	ERROR: Can't open project: <*string*>
	TCL_ERROR	1	ERROR: Can't set revision: <*string*>. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Cannot open project: <*string*>. The project is not compatible with the installed version of the Quartus Prime software. Opening the project will overwrite the old project database. If you wish to overwrite the old project database, make sure to specify the -<*string*> option.
	TCL_ERROR	1	ERROR: Can't open revision: <*string*> (project: <*string*>). The revision is not compatible with the installed version of the Quartus Prime software. Opening the revision will overwrite the old revision database. If you wish to overwrite the old revision database, make sure to specify the -<*string*> option.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Revision does not exist: <*string*>. Specify a legal revision name using the -<*string*> option.
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: <*string*>. Specify a legal project name.

3.1.30.29. project_restore (::quartus::project_ui)

The following table displays information for the `project_restore` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`project_restore [-h | -help] [-long_help] [-destination` <*directory*> `] [-overwrite] [-update_included_file_info]` <*archive_file*>	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`	Long help with examples and possible return values
	`-destination` *<directory>*	Directory where restored files are placed
	`-overwrite`	Option to overwrite files in destination directory
	`-update_included_file_info`	Option to update included file information
	<archive_file>	Archive file name
Description		

```
Restores a Quartus Prime Archive File (.qar) that
contains the project and its related files.

By default, the archive is restored into the current
directory. Use the "-destination" option to restore
the files into a new directory.

By default, the command fails if the archive already
contains files in the destination directory. Use the
"-overwrite" option to overwrite any existing files in
the destination directory.
```

Example Usage

```
## Restore archive and expand files into current directory
project_restore chiptrip.qar
## or
project_restore chiptrip.qar -destination

## Restore archive. Expand files into current directory,
## but overwrite any existing files in "."
project_restore chiptrip.qar -destination . -overwrite

## Restore project into a "restored" subdirectory
project_restore chiptrip.qar -destination "restored" -overwrite
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: *<string>*. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Error(s) found while restoring the archive. See error message(s) for details.

3.1.30.30. remove_all_global_assignments (::quartus::project_ui)

The following table displays information for the `remove_all_global_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`remove_all_global_assignments [-h | -help] [-long_help] [-entity` *<entity_name>* `] -name` *<name>* `[-section_id` *<section id>* `] [-tag` *<data>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity` *<entity_name>*	Entity to which assignment belongs
	`-name` *<name>*	Assignment name (string pattern is matched using Tcl string matching)
	`-section_id` *<section id>*	Section id
	`-tag` *<data>*	Option to tag data to this assignment

continued...

Description

```
Removes all matching global assignments.

The "-name" option is not case sensitive. This option can take
string patterns containing special characters from the set
"*?\[]" as the value. The value is matched using Tcl string
matching.

This Tcl command reads the global assignments found in
the Quartus Prime Settings File (.qsf). This Tcl command filters
the assignments data found in the .qsf and removes the data
based on the values specified by the "-name" option.

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                    Description
---------------------------   ------------------------------
CHIP                          Same as revision name
LOGICLOCK_REGION              A unique name
EDA_TOOL_SETTINGS             A unique name
CLIQUE                        A unique name
BREAKPOINT                    A unique name
CLOCK                         A unique name
AUTO_INSERT_SLD_NODE_ENTITY   A unique name

For entity-specific assignments, use the "-entity" option to
remove the assignment(s) from the specified entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.

Assignments removed by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## Remove all the registered source files

remove_all_global_assignments -name SOURCE_FILE

# Using wildcards
remove_all_global_assignments -name SOURCE*
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: <*string*> global assignment(s) were removed
TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.

continued...

	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.31. remove_all_instance_assignments (::quartus::project_ui)

The following table displays information for the `remove_all_instance_assignments` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`remove_all_instance_assignments [-h | -help] [-long_help] [-entity <entity_name> ] [-from <source> ] -name <name> [-section_id <section id> ] [-tag <data> ] [-to <destination> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which assignment belongs
	`-from <source>`	Source of the assignment (string pattern is matched using Tcl string matching)
	`-name <name>`	Assignment name (string pattern is matched using Tcl string matching)
	`-section_id <section id>`	Section id
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of the assignment (string pattern is matched using Tcl string matching)
Description		

```
Removes all matching instance assignment values.

The "-name" option is not case sensitive.
The "-to" and "-from" options are case sensitive.
```

continued...

```
These options can take string patterns containing special
characters from the set "*?\[]" as values. The values are
matched using Tcl string matching. Note that bus names are
automatically detected and do not need to be escaped. Bus
names have the following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
remove_all_instance_assignments -name LOCATION -to a[0]

# To match all indices of bus "a", type:
remove_all_instance_assignments -name LOCATION -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

remove_all_instance_assignments -name LOCATION -to "a[escape_brackets \[]\[0-2\][escape_brackets
\]]"

For more information about escaping square brackets, type
"escape_brackets -h".

This Tcl command reads the instance assignments found in
the Quartus Prime Settings File (.qsf) and removes this data based
on the values specified by the "-name", "-from", and "-to"
options.

Certain sections in the .qsf can appear more than once.
For example, because there may be more than one clock
used in a project, there may be more than one CLOCK section
each containing its own set of clock assignments. To uniquely
identify sections of this type, a <Section Id> is used.
<Section Id> can be one of three types. It can be the same as
the revision name, or it can be some unique name. The
following is a list of sections requiring a <Section Id> and
the associated <Section Id> description:

Section Id                     Description
---------------------------    -----------------------------
CHIP                           Same as revision name
LOGICLOCK_REGION               A unique name
EDA_TOOL_SETTINGS              A unique name
CLIQUE                         A unique name
BREAKPOINT                     A unique name
CLOCK                          A unique name
AUTO_INSERT_SLD_NODE_ENTITY    A unique name

For entity-specific assignments, use the "-entity" option to
remove the assignment(s) from the specified entity. If the
"-entity" option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.

Assignments removed by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## Remove all the timing requirements
## Use wildcards to catch TSU_REQUIREMENT, TCO_REQUIREMENT,
## and others
remove_all_instance_assignments -name *_REQUIREMENT

## Remove all the location assignments with
## the destination bus name "timeo".
set bus_name "timeo"
remove_all_instance_assignments -name LOCATION -to $bus_name[*]
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: <*string*> instance assignment(s) were removed
	TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
	TCL_ERROR	1	ERROR: Assignment is not an instance assignment: <*string*> -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
	TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.32. remove_all_parameters (::quartus::project_ui)

The following table displays information for the `remove_all_parameters` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`remove_all_parameters [-h | -help] [-long_help] [-entity <entity_name> ] -name <name> [-tag <data> ] [-to <destination> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-entity <entity_name>`	Entity to which parameter belongs

continued...

	`-name <name>`	Parameter name (string pattern is matched using Tcl string matching)
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of the parameter (string pattern is matched using Tcl string matching)

Description

```
Removes all matching parameters.

The "-name" option is not case sensitive.
The "-to" option is case sensitive.

If the "-to" argument is specified, the function removes
the parameters from the current entity. The parameters are
removed from the PARAMETERS section of the entity. Otherwise,
the function removes the project-wide default parameters
obtained from the DEFAULT_PARAMETERS section.

This Tcl command filters the parameter data found in the
Quartus Prime Settings File (.qsf) and removes the data based on
the values specified by the "-name" and "-to" options. These
options can take string patterns containing special characters
from the set "*?\[]" as values. The values are matched using
Tcl string matching. Note that bus names are automatically
detected and do not need to be escaped. Bus names have the
following format:

<bus name>[<bus index>] or <bus name>[*]

The <bus name> portion is a string of alphanumeric characters.
The <bus index> portion is an integer greater than or equal to
zero or it can be the character "*" used for string matching.
Notice that the <bus index> is enclosed by the square
brackets "[" and "]". For example, "a[0]" and "a[*]" are
supported bus names and can be used as follows:

# To match index 0 of bus "a", type:
remove_all_parameters -name * -to a[0]

# To match all indices of bus "a", type:
remove_all_parameters -name * -to a[*]

All other uses of square brackets must be escaped if you do
not intend to use them as string patterns. For example,
to match indices 0, 1, and 2 of the bus "a", type:

remove_all_parameters -name * -to "a[escape_brackets \[]\[0-2\][escape_brackets \]]"

For more information about escaping square brackets, type
"escape_brackets -h".

Use the "-entity" option to remove the parameters from the
specified entity. If the "-entity" option is not specified,
the value for the FOCUS_ENTITY_NAME assignment is used. If the
FOCUS_ENTITY_NAME value is not found, the revision name is used.

The parameters removed by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.
```

Example Usage

```
## The following 3 examples remove project-wide,
## default parameter values
remove_all_parameters -name WIDTH
remove_all_parameters -name *ID*
remove_all_parameters -name *

## The following 3 examples remove entity-specific
## parameter values
remove_all_parameters -name inst1 -to SIZE
remove_all_parameters -name inst1 -to *IZ*
remove_all_parameters -name inst1 -to *
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: <*string*> parameter(s) were removed
	TCL_OK	0	INFO: Removed parameter: <*string*>
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
	TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
	TCL_ERROR	1	ERROR: Parameter does not exist and cannot be removed: <*string*>. Specify an existing parameter name.
	TCL_ERROR	1	ERROR: Illegal default parameter: <*string*>. Specify a legal default parameter name.
	TCL_ERROR	1	ERROR: Illegal parameter: <*string*>. Specify a legal parameter name.
	TCL_ERROR	1	ERROR: An unknown error has occured.

3.1.30.33. resolve_file_path (::quartus::project_ui)

The following table displays information for the `resolve_file_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`resolve_file_path [-h | -help] [-long_help] <file_name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*file_name*>	Option to specify the file name
Description		

```
Returns the resolved full path of the specified file name.
If the file does not exist, the original file name is
returned.

The Quartus Prime software resolves relative paths by searching
for the file in the following directories in the following
order:

1) Project directory, which is the directory where the
   Quartus Prime Settings File (.qsf) is found.
2) Project database directory, which is the "db" directory
   found under the project directory.
3) Project library directories, which are the directories
   containing the user-specified libraries that are used
   only by the current project.
4) User library directories, which are the directories
   containing the user-specified libraries that are
   used by all Quartus Prime projects.
5) Quartus Prime library directory, which is the directory
   containing Quartus Prime libraries.
```

Example Usage

```
project_new chiptrip -overwrite

# Set one Verilog source file assignment
set_global_assignment -name VERILOG_FILE chiptrip.v

# Display the resolved full path of the Verilog
```

continued...

```
# source file assignment
set filename [get_global_assignment -name VERILOG_FILE]
set resolved_fullpath [resolve_file_path $filename]

puts "Full Path: $resolved_fullpath"

# Set more Verilog source file assignments
set_global_assignment -name VERILOG_FILE auto_max.v
set_global_assignment -name VERILOG_FILE speed_ch.v
set_global_assignment -name VERILOG_FILE tick_cnt.v
set_global_assignment -name VERILOG_FILE time_cnt.v

# Display the resolved full path of all the Verilog
# source file assignments
set file_asgns [get_all_global_assignments -name VERILOG_FILE]
foreach_in_collection file_asgn $file_asgns {

    ## Each element in the collection has the following
    ## format: {} {VERILOG_FILE} {<file_name>}

    set filename [lindex $file_asgn 2]
    set resolved_fullpath [resolve_file_path $filename]

    puts "Full Path: $resolved_fullpath"
}

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.30.34. revision_exists (::quartus::project_ui)

The following table displays information for the `revision_exists` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`revision_exists [-h | -help] [-long_help] [-project` *<project_name>* `]` *<revision_name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-project` *<project_name>*	Project name
	<revision_name>	Revision name

Description

```
Checks whether the revision exists for the specified project
or currently open project.

Returns 1, if the revision exists; returns 0, otherwise.
```

Example Usage

```
## Check if the specified revision exists
## in the specified project
if [revision_exists -ARG(project) chiptrip speed_ch] {
    puts "Revision exists"
} else {
    puts "Revision does not exist"
}

## Create revision for the currently open
## project if it does not exist
## Set the current revision otherwise
project_open chiptrip
if [revision_exists speed_ch] {
    set_current_revision speed_ch
} else {
```

continued...

	`` create_revision speed_ch } project_close ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Project does not exist or has illegal name characters: <*string*>. Specify a legal project name.
	TCL_ERROR	1	ERROR: Project name was not specified or open project does not exist. Open an existing project or specify the project name.

3.1.30.35. set_current_revision (::quartus::project_ui)

The following table displays information for the `set_current_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`set_current_revision [-h | -help] [-long_help] [-force]` <*revision_name*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-force`		Option to open the revision and overwrite the compilation database if the database version is incompatible.
	<*revision_name*>		Revision name
Description	``` Sets the specified revision name as the current revision. All assignments created or modified during an open project will also be saved to the Quartus Prime Settings File (.qsf). In 8.1 or later versions of Quartus Prime software, set_current_revision gives an error when the compilation database version is not compatible with the current version of Quartus Prime software. You may specify the "-force" option to avoid the error and overwrite the database. ```		
Example Usage	``` ## Sets "auto_max" as the current revision set_current_revision auto_max ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	WARNING: Revision is already the current revision: <*string*>. No action is required.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Revision file does not exist: <*string*>.qsf. Use delete_revision to delete the revision from the current project. Then use create_revision to create the revision and its .qsf before setting <*string*> as the current revision.
	TCL_ERROR	1	ERROR: Revision is not included in the current project: <*string*> . Use the create_revision command to create the revision.

3.1.30.36. set_global_assignment (::quartus::project_ui)

The following table displays information for the `set_global_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`set_global_assignment [-h | -help] [-long_help] [-comment <comment> ] [-disable] [-entity <entity_name> ] [-fall] -name <name> [-remove] [-rise] [-section_id <section id> ] [-tag <data> ] [ <value> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment <comment>`	Comment
	`-disable`	Option to disable assignment
	`-entity <entity_name>`	Entity to which to add assignment
	`-fall`	Option applies to falling edge
	`-name <name>`	Assignment name
	`-remove`	Option to remove assignment
	`-rise`	Option applies to rising edge
	`-section_id <section id>`	Section id
	`-tag <data>`	Option to tag data to this assignment
	<value>	Assignment value
Description		

```
Sets or removes a global assignment.

Assignments created or modified by using this Tcl
command are not saved to the Quartus Prime Settings File
(.qsf) unless you explicitly call one of the following
two Tcl commands (from the ::quartus::project Tcl
package):

1) export_assignments
2) project_close (unless -dont_export_assignments is
specified as an argument to project_close)

You must save assignment changes before you run
Quartus Prime command-line executables. Note, however,
that the Tcl commands execute_flow and execute_module
(from the ::quartus::flow Tcl package) call
"export_assignments" before they run command-line
executables.

For entity-specific assignments, use the -entity
option to force the assignment to specified entity. If
the -entity option is not specified, the value for the
FOCUS_ENTITY_NAME assignment is used. If the
FOCUS_ENTITY_NAME value is not found, the revision
name is used.

If the Quartus Prime Settings File contains a
USER_LIBRARIES assignment and you call
set_global_assignment to set a SEARCH_PATH or
USER_LIBRARIES assignment, the existing USER_LIBRARIES
assignment expands into one or more SEARCH_PATH
assignments.

Note that values that begin with a dash ("-") should
be enclosed in a backslash followed by a quote. In the
following example, -02 is enclosed by \" at the
beginning and the end.

set_global_assignment -name ARM_CPP_COMMAND_LINE \"-O2\"
```

continued...

Example Usage

```
## Specify Stratix as the family to use when compiling
set_global_assignment -name FAMILY Stratix

## If the family name has empty spaces, use quotes
set_global_assignment -name FAMILY "Stratix GX"

## or remove any empty space
set_global_assignment -name FAMILY StratixGX
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: File name <*string*> exceeds maximum of <*string*> characters. Specify a file name with fewer characters.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
TCL_ERROR	1	ERROR: The -<*string*> option is not required but was specified with the value: <*string*>. Delete the option.
TCL_ERROR	1	ERROR: The -<*string*> option is required but was not specified. Specify the required option.
TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.

continued...

	TCL_ERROR	1	ERROR: An unknown error has occured.
	TCL_ERROR	1	ERROR: Assignment <*string*> cannot be removed -- it has multiple values. Specify one value to remove or use the <*string*> command to remove all values for the assignment.
	TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.30.37. set_instance_assignment (::quartus::project_ui)

The following table displays information for the `set_instance_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`set_instance_assignment [-h | -help] [-long_help] [-comment <comment> ] [-disable] [-entity <entity_name> ] [-fall] [-from <source> ] -name <name> [-remove] [-rise] [-section_id <section id> ] [-tag <data> ] [-to <destination> ] [ <value> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment <comment>`	Comment
	`-disable`	Option to disable assignment
	`-entity <entity_name>`	Entity to which to add assignment
	`-fall`	Option applies to falling edge
	`-from <source>`	Source of assignment
	`-name <name>`	Assignment name
	`-remove`	Option to remove assignment
	`-rise`	Option applies to rising edge
	`-section_id <section id>`	Section id
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of assignment
	<*value*>	Assignment value
Description		

```
Sets or removes an instance assignment.

Assignments created or modified by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.

For entity-specific assignments, use the "-entity" option to
force the assignment to specified entity. If the "-entity"
```

continued...

```
option is not specified, the value for the FOCUS_ENTITY_NAME
assignment is used. If the FOCUS_ENTITY_NAME value is not found,
the revision name is used.
```

Example Usage

```
## Specify a TSU_REQUIREMENT of 2ns from mypin to any register
set_instance_assignment -from "mypin" -to * -name TSU_REQUIREMENT 2ns

## Remove the TSU_REQUIREMENT from mypin to all registers
set_instance_assignment -from "mypin" -to * -name TSU_REQUIREMENT -remove

## Specify the entity to which the assignment is added,
## use the -entity option
## This is needed if the top-level entity name is other than
## that of the project name
## The following command generates a top_level entity
set_instance_assignment -from "mypin" -to * -entity top_level -name TSU_REQUIREMENT 2ns
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not an instance assignment: <*string*> -- it is a global assignment. Specify an instance assignment name or use the global assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
TCL_ERROR	1	ERROR: The -<*string*> option is not required but was specified with the value: <*string*>. Delete the option.
TCL_ERROR	1	ERROR: The -<*string*> option is required but was not specified. Specify the required option.

continued...

	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: An unknown error has occured.
	TCL_ERROR	1	ERROR: Assignment <*string*> cannot be removed -- it has multiple values. Specify one value to remove or use the <*string*> command to remove all values for the assignment.
	TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.30.38. set_io_assignment (::quartus::project_ui)

The following table displays information for the `set_io_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`set_io_assignment [-h | -help] [-long_help] [-comment` <*comment*> `] [-disable] [-fall] [-io_standard` <*io standard*> `] -name` <*name*> `[-remove] [-rise] [-tag` <*data*> `] [` <*value*> `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-comment` <*comment*>		Comment
	`-disable`		Option to disable assignment
	`-fall`		Option applies to falling edge
	`-io_standard` <*io standard*>		Option to specify the io standard
	`-name` <*name*>		Assignment name
	`-remove`		Option to remove assignment
	`-rise`		Option applies to rising edge
	`-tag` <*data*>		Option to tag data to this assignment
	<*value*>		Assignment value
Description	`Sets or removes an io assignment.` `Assignments created or modified by using this Tcl command are not saved to the Quartus Prime Settings File (.qsf) unless you explicitly call one of the following two Tcl commands:` `1) export_assignments` `2) project_close (unless "-dont_export_assignments" is specified)` `These two Tcl commands reside in the ::quartus::project Tcl package. You must save assignment changes before you run Quartus Prime command-line executables. Note, however, that the Tcl commands "execute_flow" and "execute_module" (part of the ::quartus::flow Tcl package) automatically call "export_assignments" before they run command-line executables.`		
Example Usage	`## Specify LVTTL as the IO Standard for OUTPUT_PIN_LOAD assignment` `set_io_assignment 30 -name OUTPUT_PIN_LOAD -io_standard LVTTL`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
TCL_ERROR	1	ERROR: Assignment is not a global assignment: <*string*> -- it is an instance assignment. Specify a global assignment name or use the instance assignment commands.
TCL_ERROR	1	ERROR: Can't find file(s) associated with assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find section information for assignment. Specify a different assignment name.
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: File name <*string*> exceeds maximum of <*string*> characters. Specify a file name with fewer characters.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Option -<*string*> for <*string*> assignment is illegal. Specify a legal option or remove the option.
TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Found two options: -<*string*> and -<*string*>. Choose one of the options.
TCL_ERROR	1	ERROR: The -<*string*> option is not required but was specified with the value: <*string*>. Delete the option.
TCL_ERROR	1	ERROR: The -<*string*> option is required but was not specified. Specify the required option.
TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
TCL_ERROR	1	ERROR: Assignment <*string*> cannot be removed -- it has multiple values. Specify one value to remove or use the <*string*> command to remove all values for the assignment.
TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.30.39. set_location_assignment (::quartus::project_ui)

The following table displays information for the `set_location_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`set_location_assignment [-h | -help] [-long_help] [-comment <comment> ] [-disable] [-remove] [-tag <data> ] -to <destination> [ <value> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-comment <comment>`	Comment	
	`-disable`	Option to disable assignment	
	`-remove`	Option to remove assignment	
	`-tag <data>`	Option to tag data to this assignment	
	`-to <destination>`	Destination of assignment	
	<value>	Assignment value	
Description	`Sets or removes a location assignment.` `Assignments created or modified by using this Tcl command are not saved to the Quartus Prime Settings File (.qsf) unless you explicitly call one of the following two Tcl commands:` `1) export_assignments` `2) project_close (unless "-dont_export_assignments" is specified)` `These two Tcl commands reside in the ::quartus::project Tcl package. You must save assignment changes before you run Quartus Prime command-line executables. Note, however, that the Tcl commands "execute_flow" and "execute_module" (part of the ::quartus::flow Tcl package) automatically call "export_assignments" before they run command-line executables.`		
Example Usage	`set_location_assignment -to dst LOC`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Missing destination for assignment. Specify the destination for the assignment.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.30.40. set_parameter (::quartus::project_ui)

The following table displays information for the `set_parameter` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441	
Syntax	`set_parameter [-h | -help] [-long_help] [-comment <comment> ] [-disable] [-entity <entity_name> ] -name <name> [-remove] [-tag <data> ] [-to <destination> ] [ <value> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-comment <comment>`	Comment
	`-disable`	Option to disable parameter
	`-entity <entity_name>`	Entity to which to add parameter
	`-name <name>`	Parameter name
	`-remove`	Option to remove parameter
	`-tag <data>`	Option to tag data to this assignment
	`-to <destination>`	Destination of parameter
	<value>	Parameter value
Description		

```
Sets or removes the specified parameter name.

The "-name" option is not case sensitive.
The "-to" option is case sensitive.

The parameters created or modified by using this Tcl command are
not saved to the Quartus Prime Settings File (.qsf) unless you
explicitly call one of the following two Tcl commands:

1) export_assignments
2) project_close (unless "-dont_export_assignments" is specified)

These two Tcl commands reside in the ::quartus::project Tcl
package. You must save assignment changes before you run
Quartus Prime command-line executables. Note, however, that
the Tcl commands "execute_flow" and "execute_module" (part
of the ::quartus::flow Tcl package) automatically call
"export_assignments" before they run command-line executables.

Use the "-entity" option to force the parameter to the specified
entity. If the "-entity" option is not specified, the value for
the FOCUS_ENTITY_NAME assignment is used. If the FOCUS_ENTITY_NAME
value is not found, the revision name is used.

A parameter is an attribute of a megafunction, macrofunction,
or certain primitives that determines the logic created or used
to implement the function. The parameter information can be
used to determine the actual primitives and other subdesigns
needed to implement the logic of the function.

The following general guidelines apply to parameters:

*  All logic options can be assigned as parameters for individual
   instances of megafunctions or macrofunctions. For a given
   logic OPTION the precedence for parameters is:

      1) Instance specific logic option settings
      2) Instance specific parameter settings
      3) Project-wide default parameter settings

*  You cannot assign a value to the predefined Intel(R)
   parameter DEVICE_FAMILY, which represents the device family
   assigned for the project. However, you can use the parameter
   value in comparisons.

*  The predefined Intel LPM_PIPELINE and LATENCY parameters
   can be assigned to an instance of a megafunction or
```

continued...

```
   macrofunction. However, the parameter applies only to that
   instance, and is not inherited by the subdesigns of that
   instance.

*  All logic options can be assigned as parameters for individual
   megafunctions or macrofunctions. However, logic options cannot
   be assigned global, project-wide default parameter values.
```

Example Usage

```
## Set project-wide, default WIDTH parameter value
set_parameter -name WIDTH 8

## Set entity-specific SIZE parameter value
## to "my_ram" entity
set_parameter -entity my_ram -name SIZE 16

## Specify the same parameter to my_ram
## but inside "top_level" entity
set_parameter -entity top_level -to my_ram -name SIZE 16
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Removed parameter: <*string*>
TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Entity does not exist or uses illegal name characters: <*string*>. Specify a legal entity name.
TCL_ERROR	1	ERROR: Can't run Tcl command while a process is in progress: <*string*>. To run the command, stop the compilation or simulation; or wait for the compilation or simulation to complete.
TCL_ERROR	1	ERROR: Value <*string*> for <*string*> assignment is illegal. Specify a legal value.
TCL_ERROR	1	ERROR: Options are mutually exclusive: <*string*> and <*string*>. Specify only one of the two options.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: The -<*string*> option is not required but was specified with the value: <*string*>. Delete the option.
TCL_ERROR	1	ERROR: The -<*string*> option is required but was not specified. Specify the required option.
TCL_ERROR	1	ERROR: Parameter does not exist and cannot be removed: <*string*>. Specify an existing parameter name.
TCL_ERROR	1	ERROR: An unknown error has occured.
TCL_ERROR	1	ERROR: Missing <<*string*>> for <*string*> assignment. Specify the required <*string*>.

3.1.30.41. set_power_file_assignment (::quartus::project_ui)

The following table displays information for the `set_power_file_assignment` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441
Syntax	`set_power_file_assignment [-h | -help] [-long_help] [-remove] [-saf_file <saf_file> ] [-section_id <section_id> ] [-to <to> ] [-vcd_end_time <vcd_end_time> ] [-vcd_file <vcd_file> ] [-vcd_start_time <vcd_start_time> ]`

continued...

Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-remove`	Option to remove assignment
	`-saf_file` *<saf_file>*	SAF file name
	`-section_id` *<section_id>*	Section id
	`-to` *<to>*	Entity to which to apply power input file
	`-vcd_end_time` *<vcd_end_time>*	End time for VCD file parsing
	`-vcd_file` *<vcd_file>*	VCD file name
	`-vcd_start_time` *<vcd_start_time>*	Start time for VCD file parsing

Description

```
Sets or removes a power input file assignment. Power input file
assignments are specified using multiple global assignments, and
a single instance assignment as illustrated in the following
example:

set_global_assignment -name POWER_INPUT_FILE_NAME "test.vcd" -section_id test.vcd
set_global_assignment -name POWER_INPUT_FILE_TYPE VCD -section_id test.vcd
set_global_assignment -name POWER_VCD_FILE_START_TIME "10 ns" -section_id test.vcd
set_global_assignment -name POWER_VCD_FILE_END_TIME "1000 ns" -section_id test.vcd
set_instance_assignment -name POWER_READ_INPUT_FILE test.vcd -to test_design

The power input file assignment serves as a wrapper for all of the
above assignments. If the "-remove" setting is not set, the
set_power_file_assignment will also make the following assignment
to enable the use of input files:

set_global_assignment -name POWER_USE_INPUT_FILES ON

If you do not specify a "-section_id", a new section identifier is
created for the input file assignment. If a "-section_id" is
specified and it does not already exist, it is used as the new
section identifier. If a "-section_id" is specified and it does
exist, the existing input file assignments are removed and a new
input file assignment is created using the given parameters and
section identifier.

If an entity name given by "-to" is not specified, the input file
assignment applys to the top level design entity.

If the "-remove" setting is used, the input file assignment given
by the "-section_id", "-vcd_file", or "-saf_file" is removed
from the project.

Assignments created or modified by using this Tcl command are
saved to the Quartus Prime Settings File (.qsf).
```

Example Usage

```
## Specify an input SAF file applied to the top level entity
## A default section will be created
set_power_file_assignment -saf_file test.saf

## Specify an input VCD file applied to design_top|counter1
## Use the given section_id to create a new section
set_power_file_assignment -vcd_file test.vcd -to design_top|counter1 -section_id test.vcd

## Update the previous input VCD file assignment to specify a
## start and end time
set_power_file_assignment -vcd_file test.vcd -to design_top|counter1 -vcd_start_time 10ns -
vcd_end_time 100ns -section_id test.vcd

## Remove the input SAF file assignment using the file name
set_power_file_assignment -saf_file test.saf -remove

## Remove the input VCD file assignment using the section identifier
set_power_file_assignment -section_id test.vcd -remove
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: Can't find active revision name. Make sure there is an open, active revision name.
	TCL_ERROR	1	ERROR: Compiler database does not exist for revision name: <*string*>. At the minimum, run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
	TCL_ERROR	1	ERROR: Exactly one of the following file name options must be specified: -<*string*> or -<*string*>.
	TCL_ERROR	1	ERROR: If -<*string*> is set, exactly one of the following options must be specified: -<*string*>, -<*string*> or -<*string*>. All other options must not be set.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: -<*string*> and -<*string*> cannot be used with -<*string*> option.

3.1.30.42. set_user_option (::quartus::project_ui)

The following table displays information for the `set_user_option` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`set_user_option [-h | -help] [-long_help] -name <name> [ <value> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		User option name
	<*value*>		User option value
Description	`Sets the user option value for the name specified by the "-name" option. The user option is written to the quartus2.ini file.` `To get a list of all available user option names, use the "get_all_user_option_names" command.`		
Example Usage	`## Set TALKBACK_ENABLED to "on"` `set_user_option -name TALKBACK_ENABLED on`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal user option name: <*string*>. Specify a legal user option name. To get a list of legal names, use the get_all_user_option_names command.
	TCL_ERROR	1	ERROR: Illegal user option value: <*string*>. Specify a legal user option value.

3.1.30.43. test_assignment_trait (::quartus::project_ui)

The following table displays information for the `test_assignment_trait` Tcl command:

Tcl Package and Version	Belongs to ::quartus::project_ui on page 441		
Syntax	`test_assignment_trait [-h | -help] [-long_help] -name <name> -trait <trait_name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <name>`	Assignment name	
	`-trait <trait_name>`	Trait name	
Description	`Checks whether the assignment name has the specified trait.` `Returns 1, if the assignment name has the trait;` `returns 0, otherwise.`		
Example Usage	`## Test if the assignment name is case-sensitive` `if {[test_assignment_trait -name VHDL_FILE -trait CASE_SENSITIVE]} {` `  puts "VHDL_FILE assignment is case-sensitive."` `} else {` `  puts "VHDL_FILE assignment is not case-sensitive."` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Ignored assignment: <*string*>. The assignment is no longer supported.
	TCL_ERROR	1	ERROR: Value <*string*> for the -<*string*> option is illegal. Specify a legal value.
	TCL_ERROR	1	ERROR: Options cannot be specified together: -<*string*>, -<*string*> and -<*string*>. Specify only one or two of the three options.
	TCL_ERROR	1	ERROR: Illegal assignment name: <*string*>. Specify a legal assignment name. To view the list of legal assignment names, run get_all_assignment_names.
	TCL_ERROR	1	ERROR: Illegal trait: <*string*>. Specify a legal trait name.

3.1.31. ::quartus::qed

The following table displays information for the **::quartus::qed** Tcl package:

Tcl Package and Version	::quartus::qed 1.0
Description	`This package contains Tcl commands to work in the Quartus Exploration` `Dashboard. Commands perform workspace operations,` `create and manipulate project and project group objects,` `control execution of background Quartus Prime software instances, and` `import data from Quartus Prime projects.`
Availability	`This package is available for loading in the following executable:` `  quartus_sta`

continued...

Tcl Commands	

3.1.31.1. qed::add_projects_from_archive (::quartus::qed)

The following table displays information for the `qed::add_projects_from_archive` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::add_projects_from_archive [-h | -help] [-long_help] [-subdir` *<subdir>* `] [-num_seeds` *<num_seeds>* `] [-id_pattern` *<id_pattern>* `] [-group` *<group>* `]` *<archive>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-subdir` *<subdir>*	Directory name to create and restore the archive into. If specified, the directory must not yet exist. If unspecified, a sub-directory named after the archive will be created within the current directory.
	`-num_seeds` *<num_seeds>*	Number of new seeds to create. Each seed will have its own associated project in a separate subdirectory.
	`-id_pattern` *<id_pattern>*	Pattern used to generate revision names. \"%BASE_ID%\" will be replaced with the ID the first project found inside the archive. \"%SEED%\" will be replaced with the seed value being assigned to the new revision.

continued...

	`-group <group>`	Group ID to add the restored projects to
	<archive>	Archive file to restore and add to the workspace
Description		

```
Restores a Quartus Prime Pro archive file (.qar) into a fresh directory
and connects the created project to an opened workspace.

Since this command launches remote connections before completing, the
newly created projects must have access to a valid inherited compute
specification, either via the workspace or a group object that is provided
by "-group". See qed::get_inherited_compute_spec_args and qed::launch_connection
for more details.

The "-subdir" argument specifies the name of the fresh directory to create
during the restoration process.  If specified, the directory must not yet
exist. If unspecified, a sub-directory named after the archive will be created
within the current directory.

The "-num_seeds" argument controls how many times the archive is restored.
If the value of the "-num_seeds" argument is greater than 1, a separate
directory will be created for each restored project and the SEED global
assignment value will be set to a different value in each project before
this command completes.

The "-id_pattern" option controls each "id" property of the project objects which are created
when this command completes. These IDs must be unique within the workspace. In order to
guarantee this uniqueness, the seed value may be incorporated into the generated ID by including
the substring "%SEED%" into the -id_pattern value. This substring will be replaced with the
unique
seed that will be set once the revision is created. For example, if the original project
had a seed value of 4, -num_seeds 3 is specified, and the -id_pattern argument is
set to "seed_%SEED%", the resulting projects will have IDs of "seed_5", "seed_6" and "seed_7"
respectively. To derive the IDs of the new projects after the archive project's name,
include the substring "%BASE_ID%" in the -id_pattern argument.
```

Example Usage

```
    qed::workspace_new my_workspace
    qed::configure_local_compute_spec my_workspace
    qed::add_projects_from_archive my_archive.qar -num_seeds 3
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.2. qed::check_properties (::quartus::qed)

The following table displays information for the `qed::check_properties` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::check_properties [-h | -help] [-long_help] [-checks <checks> ] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-checks <checks>`	List of Tcl expressions that evaluate to true for objects to retrieve. Expressions may use properties as variables of the same name during the check.
	<object>	Identifier associated with the object, must be unique
Description		

```
Evalutes a series of expressions that operate on values of object
properties and returns a boolean expression of the result of the
evaluation.

The "qed::check_properties" command simplifies the process of
performing a conditional check on an object for a set of
parameter values. The "-checks" option requires a list of
Tcl expressions, even if you specify only one expression. If you
specify only one expression, make the one expression a list itself
as shown in the examples below.
```

continued...

	Each expression uses semantics of the Tcl "expr" command. Within each expression, access object properties with variables named according to the desired properties. Accessing object properties within expressions follows semantics of the Tcl "dict with" command.		
Example Usage	`qed::create_object -type project project_A -user_data {this is some data!}` `qed::create_object -type group group_one -projects {project_A}` `# Checks for the string "this is " appearing in the` `# user_data property of the project_A object.` `# Returns: 1` `qed::check_properties project_A -checks [list {"this is " in $user_data}]` `# Checks for the project group with ID "group_one" existing in` `# the groups property of the project_A object.` `# Returns: 1` `qed::check_properties project_A -checks [list {group_one in $groups}]` `# Checks that the project with ID project_A does not appear in` `# the projects property of the group_one object.` `# Returns: 0` `qed::check_properties group_one -checks [list {project_A ni $projects}]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.3. qed::check_properties_of_projects (::quartus::qed)

The following table displays information for the `qed::check_properties_of_projects` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::check_properties_of_projects [-h | -help] [-long_help] [-checks` *<checks>* `]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-checks` *<checks>*	List of Tcl expressions that evaluate to true for objects to retrieve. Expressions may use properties as variables of the same name during the check.
	<object>	Identifier associated with the object, must be unique
Description	Evalutes a series of expressions that operate on values of object properties for all projects in a project group, and returns the information in a Tcl dict. The keys of the Tcl dict are the IDs of the projects in the group. The values are two element lists of the form { <code> <value> }. If the <code> value is 1, an error occurred checking the property expression. The <result> contains the received error message. If the <code> value is 0, the <result> value contains the boolean expression that is the result of the evaluation. Refer to help for "qed::check_properties" command for more information.	
Example Usage	`qed::create_object -type project project_A -user_data {this is some data!}` `qed::create_object -type group group_one -projects {project_A}` `# Returns: {project_A {0 1}}` `qed::check_properties_of_projects group_one -checks [list {"this is " in $user_data}]`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.4. qed::compile (::quartus::qed)

The following table displays information for the `qed::compile` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::compile [-h | -help] [-long_help] [-async] [-load_db_state] [-timeout` *<timeout>* `] <object>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-async`		Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	`-load_db_state`		Flag to load the necessary compilation database state for the accessor executable once the compile is complete. For example, if the accessor executable is "quartus_sta", this initializes the timing netlist
	`-timeout` *<timeout>*		Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<object>		Identifier associated with the object, must be unique
Description	`Launches a compile on a connected project or group of connected projects.` `Use the "-load_db_state" option to prepare the project for deeper analysis after it opens. The specific operations performed will depend on the executable used to access the project. For example, if the project is accessed with "quartus_sta", timing analysis preparation follows the initialization procedure in the Quartus Prime Pro Timing Analyzer: creating the timing netlist, reading any SDC files, and updating the timing netlist.`		
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf -revision rev_A` `qed::launch_connection project_A -open_project` `qed::compile project_A`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.5. qed::configure_local_compute_spec (::quartus::qed)

The following table displays information for the `qed::configure_local_compute_spec` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::configure_local_compute_spec [-h | -help] [-long_help] [-exe` *<quartus_sta|quartus_cdb|quartus_sh>* `] [-exe_options` *<exe_options>* `] [-lm_license_file` *<lm_license_file>* `] [-quartus_rootdir` *<quartus_rootdir>* `] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exe` *<quartus_sta|quartus_cdb|quartus_sh>*	Executable to open the project with. Dictates the packages and commands that will be available to run.

continued...

	`-exe_options` *<exe_options>*	List of options to pass to executable.
	`-lm_license_file` *<lm_license_file>*	LM_LICENSE_FILE environment setting
	`-quartus_rootdir` *<quartus_rootdir>*	Appropriate Quartus Prime Pro installation folder to access the project
	<object>	Identifier associated with the object, must be unique
Description		

```
Sets properties on the specified object to cause background Quartus
Prime Pro software instances to execute on the same (local) machine as
the Quartus Exploration Dashboard is running on.

The "-exe" option specifies the Quartus Prime Pro executable to run.
The supported options are quartus_sta, quartus_cdb, and quartus_sh.

The "-exe_options" option specifies any options passed to the
executable that is specified with the "-exe" option.

The "-quartus_rootdir" option specifies the directory location of the
desired Quartus Prime Pro software installation. This is the root directory
of the software installation, not the bin directory. If you do
not specify a value, the root directory associated with the
Quartus Exploration Dashboard is used.

The "-lm_license_file" option specifies the LM_LICENSE_FILE value
that will be passed to the Quartus Prime Pro software executable. If you
don't know the value, look in the Tools > License Setup dialog in
the Quartus Prime Pro software GUI.
```

Example Usage

```
    qed::configure_local_compute_spec workspace_id -exe quartus_sta -quartus_rootdir /path/to/
acds/quartus
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.6. qed::configure_lsf_compute_spec (::quartus::qed)

The following table displays information for the `qed::configure_lsf_compute_spec` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::configure_lsf_compute_spec [-h | -help] [-long_help] [-exe` *<quartus_sta|quartus_cdb|quartus_sh>* `] [-exe_options` *<exe_options>* `] [-env` *<env>* `] [-error_file` *<error_file>* `] [-initial_work_dir` *<initial_work_dir>* `] [-lm_license_file` *<lm_license_file>* `] [-local_remote_path_map` *<local_remote_path_map>* `] [-output_file` *<output_file>* `] [-priority` *<priority>* `] [-processor_limit` *<processor_limit>* `] [-quartus_rootdir` *<quartus_rootdir>* `] [-queue` *<queue>* `] [-resource_requirements` *<resource_requirements>* `]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exe` *<quartus_sta|quartus_cdb|quartus_sh>*	Executable to open the project with. Dictates the packages and commands that will be available to run.
	`-exe_options` *<exe_options>*	List of options to pass to executable.
	`-env` *<env>*	Comma-separated list of environment variables to pass to job
	`-error_file` *<error_file>*	File to store standard error output. Use %J or %I to include job id or index id in the directory or filename

continued...

	`-initial_work_dir` *<initial_work_dir>*	Initial working directory on remote host. The remote host will start in this directory.
	`-lm_license_file` *<lm_license_file>*	LM_LICENSE_FILE environment setting
	`-local_remote_path_map` *<local_remote_path_map>*	The shared directory mapping between local and remote host
	`-output_file` *<output_file>*	File to store standard output. You can include Job id using %J or Index id %I in the directory or filename
	`-priority` *<priority>*	Job priority on your LSF farm. The higher the number, the higher the priority
	`-processor_limit` *<processor_limit>*	Minimum number of processors, and optionally, a maximum number of processors for the job
	`-quartus_rootdir` *<quartus_rootdir>*	Appropriate Quartus Prime Pro installation folder to access the project
	`-queue` *<queue>*	One or more comma-separated queues. Check with your administrator for a list of valid queues for your LSF farm
	`-resource_requirements` *<resource_requirements>*	Required resources. Used to restrict to machines with specific memory requirements
	<object>	Identifier associated with the object, must be unique

Description

```
Sets properties on the specified object to cause background Quartus
Prime Pro software instances to execute through the LSF workload
management platform.

Use the following options to configure the LSF settings:
 * -resource_requirements
 * -queue
 * -initial_work_dir
 * -env
 * -local_remote_path_map
 * -priority
 * -output_file
 * -error_file
 * -processor_limit

The "-exe" option specifies the Quartus Prime Pro executable to run.
The supported options are quartus_sta, quartus_cdb, and quartus_sh.

The "-exe_options" option specifies any options passed to the
executable that is specified with the "-exe" option.

The "-quartus_rootdir" option specifies the directory location of the
desired Quartus Prime Pro software installation. This is the root directory
of the software installation, not the bin directory.

The "-lm_license_file" option specifies the LM_LICENSE_FILE value
that will be passed to the Quartus Prime Pro software executable. If you
don't know the value, look in the Tools > License Setup dialog in
the Quartus Prime Pro software GUI.
```

Example Usage

```
qed::configure_lsf_compute_spec workspace_id -exe quartus_sta
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.31.7. qed::configure_pbspro_compute_spec (::quartus::qed)

The following table displays information for the `qed::configure_pbspro_compute_spec` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::configure_pbspro_compute_spec [-h | -help] [-long_help] [-exe <quartus_sta|quartus_cdb|quartus_sh>] [-exe_options <exe_options>] [-additional_args <additional_args>] [-email <email>] [-env <env>] [-initial_work_dir <initial_work_dir>] [-lm_license_file <lm_license_file>] [-priority <priority>] [-quartus_rootdir <quartus_rootdir>] [-queue <queue>] [-resources <resources>] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exe <quartus_sta|quartus_cdb|quartus_sh>`	Executable to open the project with. Dictates the packages and commands that will be available to run.
	`-exe_options <exe_options>`	List of options to pass to executable.
	`-additional_args <additional_args>`	Additional arguments to pass to the pbspro client when connecting to remote host
	`-email <email>`	List of email addresses to receive job status updates. Comma separated.
	`-env <env>`	A comma separated list of environment variables to pass to job
	`-initial_work_dir <initial_work_dir>`	Initial working directory on remote host. The remote host will start in this directory.
	`-lm_license_file <lm_license_file>`	LM_LICENSE_FILE environment setting
	`-priority <priority>`	Job priority. The higher the number, the higher the priority.
	`-quartus_rootdir <quartus_rootdir>`	Appropriate Quartus Prime Pro installation folder to access the project
	`-queue <queue>`	Name of the job queue to use
	`-resources <resources>`	Architecture, license, and/or memory requirement
	`<object>`	Identifier associated with the object, must be unique
Description		

```
Sets properties on the specified object to cause background Quartus
Prime Pro software instances to execute through the PBSPro workload
management platform.

Use the following options to configure the PBSPro settings:
 * -queue
 * -priority
 * -resources
 * -email
 * -additional_args
 * -env
 * -initial_work_dir

The "-exe" option specifies the Quartus Prime Pro executable to run.
The supported options are quartus_sta, quartus_cdb, and quartus_sh.

The "-exe_options" option specifies any options passed to the
executable that is specified with the "-exe" option.

The "-quartus_rootdir" option specifies the directory location of the
desired Quartus Prime Pro software installation. This is the root directory
of the software installation, not the bin directory.
```

continued...

	The "-lm_license_file" option specifies the LM_LICENSE_FILE value that will be passed to the Quartus Prime Pro software executable. If you don't know the value, look in the Tools > License Setup dialog in the Quartus Prime Pro software GUI.		
Example Usage	`qed::configure_pbspro_compute_spec workspace_id -exe quartus_sta`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.8. qed::configure_slurm_compute_spec (::quartus::qed)

The following table displays information for the `qed::configure_slurm_compute_spec` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::configure_slurm_compute_spec [-h | -help] [-long_help] [-exe <quartus_sta|quartus_cdb|quartus_sh>] [-exe_options <exe_options>] [-additional_args <additional_args>] [-cluster <cluster>] [-constraint <constraint>] [-cores_per_socket <cores_per_socket>] [-email <email>] [-env <env>] [-initial_work_dir <initial_work_dir>] [-licenses <licenses>] [-lm_license_file <lm_license_file>] [-memory <memory>] [-output <output>] [-partition <partition>] [-priority <priority>] [-quartus_rootdir <quartus_rootdir>] [-resources <resources>] [-sockets_per_node <sockets_per_node>] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exe <quartus_sta|quartus_cdb|quartus_sh>`	Executable to open the project with. Dictates the packages and commands that will be available to run.
	`-exe_options <exe_options>`	List of options to pass to executable.
	`-additional_args <additional_args>`	Additional arguments to pass to the slurm client when connecting to remote host
	`-cluster <cluster>`	Comma separated list of clusters to use
	`-constraint <constraint>`	Node features required as defined by Slurm administrator
	`-cores_per_socket <cores_per_socket>`	Number of CPU cores per socket
	`-email <email>`	Email address to receive job status updates. Defaults to submitting user.
	`-env <env>`	Comma-separated list of environment variables to pass to job
	`-initial_work_dir <initial_work_dir>`	Initial working directory on remote host. The remote host will start in this directory.
	`-licenses <licenses>`	Licenses that need to be allocated for this job
	`-lm_license_file <lm_license_file>`	LM_LICENSE_FILE environment setting
	`-memory <memory>`	Memory requirement per node. Default unit is megabytes
	`-output <output>`	Filename to store standard output
	`-partition <partition>`	Comma separated list of partitions to use

continued...

	`-priority` <*priority*>	Job priority. The higher the number, the higher the priority
	`-quartus_rootdir` <*quartus_rootdir*>	Appropriate Quartus Prime Pro installation folder to access the project
	`-resources` <*resources*>	Architecture, license, and/or memory requirement
	`-sockets_per_node` <*sockets_per_node*>	Number of CPUs per node
	<*object*>	Identifier associated with the object, must be unique

Description

```
Sets properties on the specified object to cause background Quartus
Prime Pro software instances to execute through the SLURM workload
management platform.

Use the following options to configure the SLURM settings:
 * -cluster
 * -partition
 * -memory
 * -sockets_per_node
 * -cores_per_socket
 * -priority
 * -resources
 * -constraint
 * -email
 * -additional_args
 * -env
 * -initial_work_dir
 * -output
 * -licenses

The "-exe" option specifies the Quartus Prime Pro executable to run.
The supported options are quartus_sta, quartus_cdb, and quartus_sh.

The "-exe_options" option specifies any options passed to the
executable that is specified with the "-exe" option.

The "-quartus_rootdir" option specifies the directory location of the
desired Quartus Prime Pro software installation. This is the root directory
of the software installation, not the bin directory.

The "-lm_license_file" option specifies the LM_LICENSE_FILE value
that will be passed to the Quartus Prime Pro software executable. If you
don't know the value, look in the Tools > License Setup dialog in
the Quartus Prime Pro software GUI.
```

Example Usage

```
qed::configure_slurm_compute_spec workspace_id -exe quartus_sta
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.9. qed::configure_ssh_compute_spec (::quartus::qed)

The following table displays information for the `qed::configure_ssh_compute_spec` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493
Syntax	`qed::configure_ssh_compute_spec [-h | -help] [-long_help] [-exe` <*quartus_sta|quartus_cdb|quartus_sh*> `] [-exe_options` <*exe_options*> `] [-additional_args` <*additional_args*> `] [-custom_ssh_port` <*custom_ssh_port*> `] -hostnames` <*hostnames*> `[-initial_work_dir` <*initial_work_dir*> `] [-jobs_per_host` <*jobs_per_host*> `] [-lm_license_file` <*lm_license_file*> `] [-local_remote_path_map` <*local_remote_path_map*> `] [-private_key` <*private_key*> `] [-quartus_rootdir` <*quartus_rootdir*> `] [-reverse_tunnel_port` <*reverse_tunnel_port*> `] [-ssh_client` <*ssh_client*> `] [-ssh_server` <*ssh_server*> `] [-user` <*user*> `]` <*object*>
Arguments	`-h | -help` — Short help

continued...

	`-long_help`	Long help with examples and possible return values
	`-exe` *<quartus_sta|quartus_cdb|quartus_sh>*	Executable to open the project with. Dictates the packages and commands that will be available to run.
	`-exe_options` *<exe_options>*	List of options to pass to executable.
	`-additional_args` *<additional_args>*	Additional arguments to pass to the SSH client when connecting to remote host
	`-custom_ssh_port` *<custom_ssh_port>*	Custom port to use when making the SSH connection. This is only needed if the remote host(s) uses a port other than port 22 for SSH
	`-hostnames` *<hostnames>*	Select servers to use. Supports a comma-separated list of hostnames, or a value of file:filename, where the named file contains newline separated hostnames.
	`-initial_work_dir` *<initial_work_dir>*	Initial working directory on remote host. The remote host will start in this directory.
	`-jobs_per_host` *<jobs_per_host>*	Number of jobs each host should run in parallel. Default is 1
	`-lm_license_file` *<lm_license_file>*	LM_LICENSE_FILE environment setting
	`-local_remote_path_map` *<local_remote_path_map>*	Shared directory mapping between local and remote host. Format is local_path;remote_path.
	`-private_key` *<private_key>*	Private key associated with the remote host
	`-quartus_rootdir` *<quartus_rootdir>*	Appropriate Quartus Prime Pro installation folder to access the project
	`-reverse_tunnel_port` *<reverse_tunnel_port>*	Enable port forwarding and specify a port on the remote host to be forwarded to the local host. Remote host will connect to DSE server using localhost:<reverse_tunnel_port> over the secured ssh channel.
	`-ssh_client` *<ssh_client>*	Path to the ssh client used to connect to remote host
	`-ssh_server` *<ssh_server>*	Windows only. Specify SSH server used by remote host. openssh or cygwin
	`-user` *<user>*	Username to login to remote machine
	<object>	Identifier associated with the object, must be unique
Description		

```
Sets properties on the specified object to cause background Quartus
Prime Pro software instances to connect through the ssh protocol.

The "-hostnames" option specifies a list of hostnames that can be
connected to through the ssh protocol that will execute the background
Quartus Prime Pro software instances. Entries in the list must be
comma-separated and may be formatted as "file:<filename>", where each
file contains a list of newline separated hostnames to consider alongside
any other entries specified.

The "-jobs_per_host" option limits the number of jobs a given host
can be assigned.

The "-custom_ssh_port" option overries the port to use when making
the SSH connection. Only required if the remote host(s) uses a port
other than port 22 for SSH.

The "-private_key" option specifies the private SSH key file to authenticate
and log in to remote host. Use this option if the remote host prompts user for
password when connecting using the ssh client. The corresponding public
key should have been appended to the remote host's
 ~/.ssh/authorized_keys or ~/.ssh/authorized_keys2 files.

The "-private_key" option specifies a private key that is associated
with the remote host. The corresponding pulib key should have been
```

continued...

	appending to the remote host's ~/.ssh/authorized_keys or ~/.ssh/authorized_keys2 files. The "-user" option specifies the user or login name to use when logging in to remote host. Use this option if the user name on remote host is different from the current user. The "-ssh_client" option specifies the path to the SSH client to use when launching the connection. On Linux, it defaults to 'ssh'. On Windows, it defaults to Cygwin ssh. Windows also supports plink from the PuTTY package. The "-additional_arguments" option specifies additional commandline arguments to pass to the SSH client. The "-local_remote_path_map" specifies the shared directory mapping between local and remote host. Format is local_path;remote_path. Windows to Linux Example: [drive:\windows path];[linux path] s:\designs\two_reg;/data/usr/designs/two_reg Linux to Windows Example: /data/usr;s:\data\usr The "-ssh_server" option specifies the server used by the remote host. Examples include openssh or cygwin. The "-initial_work_dir" option specifies the directory that the remote job should start in. The "-reverse_tunnel_port" option Enables port forwarding and specifies a port on the remote host to be forwarded to the local host. Use this if there is a firewall that separates the local host and the remote host. Remote host will connect to QED server using localhost:<reverse_tunnel_port> over the secured ssh channel. The port can be any ephemeral port that is not used by another process. Set value to 'auto' to let QED to automatically find a free network port on remote host. Omit this setting to disable port forwarding. The "-exe" option specifies the Quartus Prime Pro executable to run. The supported options are quartus_sta, quartus_cdb, and quartus_sh. The "-exe_options" option specifies any options passed to the executable that is specified with the "-exe" option. The "-quartus_rootdir" option specifies the directory location of the desired Quartus Prime Pro software installation. This is the root directory of the software installation, not the bin directory. The "-lm_license_file" option specifies the LM_LICENSE_FILE value that will be passed to the Quartus Prime software executable. If you don't know the value, look in the Tools > License Setup dialog in the Quartus Prime Pro software GUI.
Example Usage	qed::configure_ssh_compute_spec workspace_id -exe quartus_sta -hostnames hostname - quartus_rootdir /path/to/acds/quartus qed::configure_ssh_compute_spec workspace_id -exe quartus_sta -hostnames file:/path/to/ hostnames.txt -quartus_rootdir /path/to/acds/quartus qed::configure_ssh_compute_spec workspace_id -exe quartus_sta -hostnames host1,host2 - quartus_rootdir /path/to/acds/quartus qed::configure_ssh_compute_spec workspace_id -exe quartus_sta -hostnames hostname - quartus_rootdir /path/to/acds/quartus -user username -private_key ~/.ssh/id_rsa

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.10. qed::create_object (::quartus::qed)

The following table displays information for the `qed::create_object` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::create_object [-h | -help] [-long_help] -type <project|group> [-user_data <user_data> ] [-groups <groups> ] [-qpf_path <qpf_path> ] [-revision <revision> ] [-projects <projects> ] [-default_group_id <default_group_id> ] [ <id> ]`	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`	Long help with examples and possible return values
	`-type` *<project|group>*	Type of QED Workspace object
	`-user_data` *<user_data>*	Freetext field to store any interesting metadata on the object. Use qed::set_user_data and qed::get_user_data to interact with the value as a dict instead of a string.
	`-groups` *<groups>*	Set of group IDs this project is a member of (Valid only for -type project)
	`-qpf_path` *<qpf_path>*	Full path to a .qpf file to open (Valid only for -type project)
	`-revision` *<revision>*	Name of the revision to open (Valid only for -type project)
	`-projects` *<projects>*	Set of projects belonging to the group (Valid only for -type group)
	`-default_group_id` *<default_group_id>*	Identifier used for the 'default group' that's created to house ungrouped projects during sanitize_workspace (Valid only for -type workspace)
	<id>	Identifier associated with the object, must be unique

Description

```
Creates project or project group objects. The command returns the
ID of the newly created object.

The "-type" option is required, and must specify one of project
or group.

Other supported options depend on the type of object you are
creating. The options correspond to property names that are valid
for the type of object you are creating.
```

Example Usage

```
    qed::create_object -type project project_A     -qpf_path /path/to/project_A/project.qpf -
revision project_A
    qed::create_object -type project project_B     -qpf_path /path/to/project_B/project.qpf -
revision project_B
    qed::create_object -type group   both_projects -projects {project_A project_B}
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.11. qed::delete_object (::quartus::qed)

The following table displays information for the `qed::delete_object` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::delete_object [-h | -help] [-long_help]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<object>	Identifier associated with the object, must be unique

Description

```
Deletes project or project group objects.

The most common reason to delete an object is if you have created
it by mistake. If you added the wrong Quartus Prime Pro project to the
workspace, you can edit its properties to point to the correct
project, or delete the wrong one and add the right one.

Deleting a project object also removes any and all reports you imported
from the project.

Deleting a project group object also removes any and all reports
```

continued...

	you created for that project group. Typically those reports combine or compare data from project reports. Deleting a project group object does not delete the project objects that were part of the project group.		
Example Usage	set id [qed:create_object -type project] qed::delete_object $id		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.12. qed::delete_object_report_panel (::quartus::qed)

The following table displays information for the `qed::delete_object_report_panel` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::delete_object_report_panel [-h | -help] [-long_help] -panel_name` *<panel_name>* *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-panel_name` *<panel_name>*	Name of the report panel to delete
	<object>	Identifier associated with the object, must be unique
Description	Deletes the specified report panel associated with the given object ID. Specify the panel name relative to the folder of the object ID.	
Example Usage	qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf qed::launch_connection project_A -open_project set imported_name [qed::import_report_panel project_A -panel_name "Timing Analyzer||Setup Summary"] qed::delete_object_report_panel project_A -panel_name $imported_name	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.13. qed::disconnect (::quartus::qed)

The following table displays information for the `qed::disconnect` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::disconnect [-h | -help] [-long_help] [-async] [-timeout` *<timeout>* `] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-async`	Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)

continued...

	`-timeout` *<timeout>*	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<object>	Identifier associated with the object, must be unique
Description		

```
Closes background instances of the Quartus Prime Pro software associated
with a project object, or projects in a project group.

If you invoke the "qed::disconnect" command and want to extract more
data from remote projects, or execute commands in them, you must use
the "qed::launch_connection" command to re-initiate the connection.
```

Example Usage

```
    qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf

    qed::launch_connection project_A

    qed::disconnect project_A
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.14. qed::find_projects_under_directory (::quartus::qed)

The following table displays information for the `qed::find_projects_under_directory` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::find_projects_under_directory [-h | -help] [-long_help] [-project_match` *<project_match>* `] [-directory_match` *<directory_match>* `] [-include_all_revisions]` *<search_dir>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-project_match` *<project_match>*	Pattern to match against the QPF file basenames
	`-directory_match` *<directory_match>*	Pattern to match against the QPF file directories
	`-include_all_revisions`	Flag to return all revisions of detected projects instead of the first
	<search_dir>	Directory to search
Description		

```
Recursively searches a directory for Quartus Prime Pro projects
and returns a list of Tcl dicts, where each dict contains
metadata about the projects that were found. The metadata is
typically used when you add projects to the workspace.

If no Quartus Prime Pro projects are found, the command returns an empty
list.

A Quartus Prime Pro project is considered to be a file with a .qpf
extension.

The "-project_match" and "-directory_match" options use Tcl glob-style
name matching semantics. Supported wildcard characters include * and ?.
The default value of the "-project_match" and "-directory_match"
options is *.

The command recursively finds all files with names matching *.qpf,
starting in the directory specified with the "-search_dir" option.
You must specify a directory with the "-search_dir" option, even
if it is "." (the current directory); there is no default value.
The command applies the directory_match wildcard to the name of each
directory containing a .qpf file, and it applies the project_match
wildcard to the file name of the .qpf file.
```

continued...

	The command extracts metadata for any Quartus Prime Pro projects that match the "-project_match" and "-directory_match" options. If you specify the "-include_all_revisions" option, metadata is extracted for all revisions of each project. If you do not specify the option, metadata is extracted for only the active revision of each project. The dict that is returned for each project includes the following properties: qpf_path is the absolute path of the Quartus Prime Pro project file, revision is the name of the project revision, version is the version of the Quartus Prime Pro software last used to open the project, and id is an automatically-generated unique name for the combination of that project and revision.		
Example Usage	`qed::find_projects_under_directory "."`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.15. qed::fork_new_revision (::quartus::qed)

The following table displays information for the `qed::fork_new_revision` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::fork_new_revision [-h | -help] [-long_help] [-add_to_groups` *<add_to_groups>* `] [-async] [-copy_results] [-id` *<id>* `] -revision` *<revision>* `[-timeout` *<timeout>* `]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add_to_groups` *<add_to_groups>*	List of one or more groups to add the forked project object to. Defaults to the same group(s) as the source project object.
	`-async`	Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	`-copy_results`	Flag to copy the source revision's compilation results into the new revision
	`-id` *<id>*	Identifier to associate with the new project object, must be unique
	`-revision` *<revision>*	Name of the new revision to create
	`-timeout` *<timeout>*	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<object>	Identifier associated with the object, must be unique
Description	Creates a new revision within a connected project and initializes a new project object that may be used to connect to that revision. Revisions within a Quartus Prime Pro project may be used to isolate and interact in parallel with multiple different versions of a design. These differences may include settings or assignments, which includes referencing different versions of source files to use during compilation. The project object returned by this utility will not yet be connected to the remote revision, and the database associated with the revision will not be complete. Run "qed::launch_connection" and "qed::compile" to generate a complete database for the new revision.	

continued...

	The "-revision" option must specify a unique and new name for a revision to generate within the remote project. This command will return an error if the revision cannot be created successfully. The new revision will be based on the revision that the connected project is accessing, which means it inherits all settings and assignments from that source revision and with no further changes will produce identical compilation results as compared to that source revision. The "-id" option controls the "id" property of the project object which is created when this command completes. If unspecified, a unique ID will be generated and assigned to the new project. The "-add_to_groups" option will initialize the new project object's "groups" property to be a list of pre-existing group(s). If unspecified, the forked project object will inherit its 'groups' property from the source project object. The "-copy_results" option initiates a file copy of the compilation results from the source revision into the new revision. This means the new revision appears as being fully compiled and prepared for analysis but identical to the source revision until some future change is made and the revision is recompiled.
Example Usage	`qed::create_object -type group my_group` `qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf -revision rev_A -groups my_group` `qed::launch_connection project_A -open_project` `qed::run project_A -cmd "set_global_assignment -name SEED 1"` `qed::fork_new_revision project_A -revision rev_B -id project_B -inherit_groups` `qed::launch_connection project_B -open_project` `qed::run project_B -cmd "set_global_assignment -name SEED 2"` `# Returns: {project_A 1 project_B 2}` `qed::run my_group -cmd "get_global_assignment -name SEED"`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.16. qed::fork_new_seeds (::quartus::qed)

The following table displays information for the `qed::fork_new_seeds` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::fork_new_seeds [-h | -help] [-long_help] [-id_pattern` *<id_pattern>* `] [-launch_compiles] -num_seeds` *<num_seeds>* `[-revision_pattern` *<revision_pattern>* `]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id_pattern` *<id_pattern>*	Pattern used to generate revision names. \"%ORIGINAL_ID%\" will be replaced with the current ID of the source project. \"%SEED%\" will be replaced with the seed value being assigned to the new revision.
	`-launch_compiles`	Flag to compile the forked seeds after changing their SEED assignment value
	`-num_seeds` *<num_seeds>*	Number of new seeds to create
	`-revision_pattern` *<revision_pattern>*	Pattern used to generate revision names. \"%ORIGINAL_REVISION%\" will be replaced with the current revision of the source project. \"%SEED%\" will be replaced with the seed value being assigned to the new revision.
	<object>	Identifier associated with the object, must be unique

continued...

Description

```
Creates a set of new revisions within a connected project and initializes a new
project object for each one. The new objects are then connected and the value of
the "SEED" global assignment is modified accordingly. Each new object will inherit
its "groups" property from the source project object.

The "-num_seeds" argument dictates how many revisions to create and connect. The seed
values will be the next in line after the value of the source project's seed. For
example, if the source project used a seed of 4 and -num_seeds 3 is specified, the
returned projects will have seeds 5, 6, and 7 set, respectively.

The compilation database associated with the returned projects will not be
complete. Run "qed::compile" to generate a complete database for each
new revision.

The "-revision_pattern" option must specify a unique and new name for each revision
to generate within the remote project. This command will return an error if
a revision cannot be created successfully. In order to guarantee this uniqueness, the
seed value may be incorporated into the generated revision by including the substring
"%SEED%" into the -revision_pattern value. This substring will be replaced with the unique
seed that will be set once the revision is created. For example, if the original project
had a seed value of 4, -num_seeds 3 is specified, and the -revision_pattern argument is
set to "seed_%SEED%", the resulting revisions will be named "seed_5", "seed_6" and "seed_7"
respectively. Note that the revision names are visible via the returned project objects'
"revision" property. To name the new revisions after the original project's revision,
include the substring "%ORIGINAL_REVISION%" in the -revision_pattern argument.

Each new revision will be based on the revision that the connected project is accessing,
which means it inherits all settings and assignments from that source revision except
that it will have a different value set for the "SEED" global assignment.

The "-id_pattern" option controls each "id" property of the project objects which are created
when this command completes. These IDs must be unique within the workspace. In order to
guarantee this uniqueness, the seed value may be incorporated into the generated ID by including
the substring "%SEED%" into the -id_pattern value. This substring will be replaced with the
unique
seed that will be set once the revision is created. For example, if the original project
had a seed value of 4, -num_seeds 3 is specified, and the -id_pattern argument is
set to "seed_%SEED%", the resulting projects will have IDs of "seed_5", "seed_6" and "seed_7"
respectively. To derive the IDs of the new projects after the original project's revision,
include the substring "%ORIGINAL_ID%" in the -id_pattern argument.

The "-launch_compiles" option will automatically launch Quartus Prime Pro compiles on just the
forked seeds after their "SEED" global assignments have been updated. This is equivalent to
creating
a new group that just contains the new project objects and running qed::compile on that group.
```

Example Usage

```
    qed::create_object -type group my_group
    qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf -revision
rev_A -groups my_group
    qed::launch_connection project_A -open_project
    qed::run project_A -cmd "set_global_assignment -name SEED 1"

    # Fork one new seed with a specific ID and revision name
    qed::fork_new_seeds project_A -num_seeds 1 -revision_pattern rev_B -id project_B -
inherit_groups
    qed::launch_connection project_B -open_project

    # Fork several new seeds and derive IDs and revision names from patterns
    qed::fork_new_seeds project_B -num_seeds 3 -revision_pattern rev_B_%SEED% -id
project_B_seed_%SEED% -inherit_groups

    # Returns: {project_A 1 project_B 2 project_B_seed_3 3 project_B_seed_4 4 project_B_seed_5
5}
    qed::run_command my_group -cmd "get_global_assignment -name SEED"
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.31.17. qed::generate_report (::quartus::qed)

The following table displays information for the `qed::generate_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::generate_report [-h | -help] [-long_help] [-arguments <arguments> ] [-async] [-timeout <timeout> ] -type <type> <object>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-arguments <arguments>`		Arguments that customize each report
	`-async`		Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	`-timeout <timeout>`		Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	`-type <type>`		Type of report to generate
	<object>		Identifier associated with the object, must be unique
Description	`This command currently contains no help description.`		
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf` `qed::launch_connection project_A -open_project` `qed::generate_report project_A -type report_timing -arguments "-npaths 100 -from_clock sys_clk"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.18. qed::get_all_properties_dict (::quartus::qed)

The following table displays information for the `qed::get_all_properties_dict` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::get_all_properties_dict [-h | -help] [-long_help] [-set_only] [-writeable_only] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-set_only`	Flag indicating that any unset property names should be filtered out.
	`-writeable_only`	Flag indicating that read-only property names should be filtered out.
	<object>	Identifier associated with the object, must be unique

continued...

Description	Returns a Tcl dict representing the properties of the given object. Keys of the Tcl dict represent the property names valid for the given object, and values of the Tcl dict represent the associated values of each property. When you create an object, some of its properties may not be set; they may have no value and no default value. When you use the "-set_only" option, the list of properties returned is limited to properties that are set. Some object properties are read-only, and the rest are read/write. When you use the "-writeable_only" option, the list of properties returned is limited to properties that are read/write.		
Example Usage	`qed::create_object -type project project_A -user_data {this is some data!}` `qed::get_all_properties_dict project_A -writeable_only` `qed::create_object -type group group_one -projects {project_A}` `qed::get_all_properties_dict group_one -set_only`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.19. qed::get_default_group_id (::quartus::qed)

The following table displays information for the `qed::get_default_group_id` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::get_default_group_id [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	Returns the project group ID reserved for the default group. All projects are required to be in at least one project group. This requirement is not enforced by the project or project group objects themselves, but by the "qed::sanitize_workspace" command. To simplify the experience for new users, creating a project group is an optional step. However, to satisfy the requirement that all projects are in at least one project group, a project group may be automatically created by the "qed::sanitize_workspace" command if no project group exists. When a project group is created by the "qed::sanitize_workspace" command, its ID is the default group ID.		
Example Usage	`qed::get_default_group_id`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.20. qed::get_object_report_panel_contents (::quartus::qed)

The following table displays information for the `qed::get_object_report_panel_contents` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::get_object_report_panel_contents [-h | -help] [-long_help] -panel_name <panel_name> <object>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-panel_name <panel_name>`		Name of the report panel to retrieve
	<object>		Identifier associated with the object, must be unique
Description	`Returns the serialized content of an object panel if it exists.`		
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf` `qed::launch_connection project_A -open_project` `set imported_name [qed::import_report_panel project_A -panel_name "Timing Analyzer||Setup Summary"]` `qed::get_object_report_panel_contents project_A -panel_name $imported_name`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.21. qed::get_object_report_panel_names (::quartus::qed)

The following table displays information for the `qed::get_object_report_panel_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::get_object_report_panel_names [-h | -help] [-long_help] [-include_generated_panels] [-regexp_match <regexp_match> ] [-string_match <string_match> ] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-include_generated_panels`	Flag to include QED-generated panels in the output
	`-regexp_match <regexp_match>`	Regex to filter all panels, in the style of the \[regexp\] command
	`-string_match <string_match>`	Pattern to filter all panels, in the style of the \[string match \] command
	<object>	Identifier associated with the object, must be unique
Description	`Returns a Tcl list of report panel names associated with the given object.` `The "-string_match" option supports Tcl glob-style pattern matching.` `The "-regexp_match" option supports Tcl regexp pattern matching. You should enclose your regular expression in curly braces.` `The "-include_generated_panels" option control whether report panels that were generated by the Quartus Exploration Dashboard are included`	

continued...

	in the list. If you call the "qed::get_object_report_panel_names" command with the ID of a project, and do not specify the "-include_generated_panels" option, the list of names includes only report panels that were imported from the project. Typically, report panels associated with a project group are generated by the Quartus Exploration Dashboard, so you should use the "-include_generated_panels" option when you run the command with a project group ID.
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf` `qed::launch_connection project_A -open_project` `qed::import_report_panel project_A -panel_name "Timing Analyzer||Setup Summary"` `qed::import_report_panel project_A -panel_name "Timing Analyzer||Hold Summary"` `qed::get_object_report_panel_names`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.22. qed::get_objects (::quartus::qed)

The following table displays information for the `qed::get_objects` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::get_objects [-h | -help] [-long_help] [-type <project|group|workspace> ] [-user_data <user_data> ] [-groups <groups> ] [-qpf_path <qpf_path> ] [-revision <revision> ] [-projects <projects> ] [-default_group_id <default_group_id> ] [-checks <checks> ] [ <id> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-type <project|group|workspace>`	Type of QED Workspace object
	`-user_data <user_data>`	Freetext field to store any interesting metadata on the object. Use qed::set_user_data and qed::get_user_data to interact with the value as a dict instead of a string.
	`-groups <groups>`	Set of group IDs this project is a member of (Valid only for -type project)
	`-qpf_path <qpf_path>`	Full path to a .qpf file to open (Valid only for -type project)
	`-revision <revision>`	Name of the revision to open (Valid only for -type project)
	`-projects <projects>`	Set of projects belonging to the group (Valid only for -type group)
	`-default_group_id <default_group_id>`	Identifier used for the 'default group' that's created to house ungrouped projects during sanitize_workspace (Valid only for -type workspace)
	`-checks <checks>`	List of Tcl expressions that evaluate to true for objects to retrieve. Expressions may use properties as variables of the same name during the check.
	`<id>`	Identifier associated with the object, must be unique
Description	Returns a Tcl list of project or project group objects that match the options you specify. If no objects match, the command returns an empty list.	

continued...

Example Usage	<pre>qed::get_objects -type project project_A -user_data "hello world!" -qpf_path /path/to/project_A/project.qpf -revision project_A qed::get_objects -type group my_group -user_data "hello neighbour!" -projects {project_A} # Returns: {project_A my_group} qed::get_objects # Returns: {project_A} qed::get_objects -type project # Returns: {my_group} qed::get_objects -projects project_A # Returns: {project_A my_group} qed::get_objects -checks [list {"hello" in $user_data}]</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.23. qed::get_project_report_panel_names (::quartus::qed)

The following table displays information for the `qed::get_project_report_panel_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::get_project_report_panel_names [-h | -help] [-long_help] [-include_generated_panels] [-regexp_match` *<regexp_match>* `] [-string_match` *<string_match>* `] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-include_generated_panels`	Flag to include QED-generated panels in the output
	`-regexp_match` *<regexp_match>*	Regex to filter all panels, in the style of the \[regexp\] command
	`-string_match` *<string_match>*	Pattern to filter all panels, in the style of the \[string match \] command
	<object>	Identifier associated with the object, must be unique
Description	<pre>Returns a multi-level Tcl dict of report panel names associated with the projects of the given group object. In the event that certain project report panels across the group only differ by seed-dependent values (e.g. slack values, Design Assistant violation counts, etc.) they will be grouped together in the returned dictionary based on a sanitized version of the panel names. The primary keys in the returned dictionary correspond to these sanitized names, and each sanitized name maps to a sub-dictionary where the key is a project that maps to the original, un-sanitized name of the panel associated with that project. If a panel was imported or generated on some of the group's projects but not others, only the projects with that panel will appear in the values associated with that panel's name. Each un-sanitized panel name associated with a project can be passed to the "qed::get_object_report_panel_contents" command alongside the project ID the panel name maps to. There are several ways to control the panel names which are returned by this command. The "-string_match" option supports Tcl glob-style pattern matching. The "-regexp_match" option supports Tcl regexp pattern matching. You should enclose your regular expression in curly braces. The "-include_generated_panels" option control whether report panels that were generated by the Quartus Exploration Dashboard are included</pre>	

continued...

	in the list. If you call the "qed::get_project_report_panel_names" command, and do not specify the "-include_generated_panels" option, the dict returned includes only report panels that were imported from each project.
Example Usage	qed::create_object -type project project_A -qpf_path /file/path/to/A/project.qpf qed::launch_connection project_A -open_project qed::create_object -type project project_B -qpf_path /file/path/to/B/project.qpf qed::launch_connection project_B -open_project qed::create_object -type group both_projects -projects {project_A project_B} qed::import_report_panel both_projects -panel_name "Timing Analyzer||Setup Summary" qed::import_report_panel project_A -panel_name "Timing Analyzer||Hold Summary" # Returns: # { # "Timing Analyzer||Setup Summary" { # project_A "Timing Analyzer||Setup Summary" # project_B "Timing Analyzer||Setup Summary" # } # "Timing Analyzer||Hold Summary" { # project_B "Timing Analyzer||Hold Summary" # } # } qed::get_project_report_panel_names both_projects

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.24. qed::get_property (::quartus::qed)

The following table displays information for the `qed::get_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::get_property [-h | -help] [-long_help] -property` *<property>* *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-property` *<property>*	Name of the property to query
	<object>	Identifier associated with the object, must be unique
Description	Returns the value of the given property of the object. If the object does not have the property (the property does not exist), the command returns a Tcl error. If the object has the property (the property exists) but the property is not set, the command returns a Tcl error. Refer to help for the "qed::has_property" command for how to check whether a property exists or is set.	
Example Usage	qed::create_object -type project project_A -user_data {this is some data!} # Returns: {this is some data!} qed::get_property project_A -property user_data	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.25. qed::get_property_of_projects (::quartus::qed)

The following table displays information for the `qed::get_property_of_projects` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::get_property_of_projects [-h | -help] [-long_help] -property` *<property>* *<object>*		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-property` *<property>*	Name of the property to query	
	<object>	Identifier associated with the object, must be unique	
Description	Gets the value of the given property for all projects in the given project group and returns the information in a Tcl dict. The keys of the Tcl dict are the IDs of the projects in the group. The values are two element lists of the form { <code> <value> }. If the <code> value is 1, an error occurred reading the property. The <result> contains the received error message. If the <code> value is 0, the <result> value contains the value of the property. Refer to help for the "qed::get_property" command for more information.		
Example Usage	`qed::create_object -type project project_A -user_data {this is some data!}` `qed::create_object -type group group_one -projects {project_A}` `# Returns: {project_A {0 {this is some data!}}}` `qed::get_property_of_projects group_one -property user_data`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.26. qed::get_return_value (::quartus::qed)

The following table displays information for the `qed::get_return_value` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::get_return_value [-h | -help] [-long_help] [-return_token` *<return_token>* `] [-timeout` *<timeout>* `]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-return_token` *<return_token>*	Token returned by a previous asynchronous run invocation
	`-timeout` *<timeout>*	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<object>	Identifier associated with the object, must be unique
Description	Returns data from a command that was run without blocking (using the "-async" option). You must specify the token returned from another command such as "qed::launch_connection" or "qed::run_command". Tokens are not interchangeable between objects. The "qed::get_return_value"	

continued...

	command returns a Tcl error if the given token was not issued by the given object. The "qed::get_return_value" command blocks until the return value		
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf` `qed::launch_connection project_A` `set tok [qed::run_command project_A -cmd "after 1000; expr {100 * 100}"]` `# Waits for ~1 second and returns 10000` `qed::get_return_value project_A -return_token $tok`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.27. qed::get_user_data (::quartus::qed)

The following table displays information for the `qed::get_user_data` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::get_user_data [-h | -help] [-long_help] -key` *<key>* *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-key` *<key>*	Arbitrary key to look up inside the user_data property of the selected object
	<object>	Identifier associated with the object, must be unique
Description	Reads the "user_data" property of an object as if it is a dict, and looks up the value stored at the given key. It is possible to use this command in conjunction with "qed::set_property -property user_data", but whenever this method is invoked, any existing value must be a legal Tcl dictionary (which includes the empty string). "qed::get_property -property user_data" returns the entire value of the property instead of looking up a single key.	
Example Usage	`qed::create_object -type project project_A` `qed::set_property -property user_data -value [dict create key_one value_1] project_A` `qed::set_user_data -key key_two -value value_2 project_A` `# Returns: {key_one value_1 key_two value_2}` `qed::get_property -property user_data project_A` `# Returns: value_1` `qed::get_user_data -key key_one project_A` `# Returns: value_2` `qed::get_user_data -key key_two project_A`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.28. qed::has_property (::quartus::qed)

The following table displays information for the `qed::has_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::has_property [-h | -help] [-long_help] -property` *<property>* `[-is_set]` *<object>*	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`	Long help with examples and possible return values
	`-property` *<property>*	Name of the property to check
	`-is_set`	Flag indicating that this should only return true if the property is set to something (including a default value)
	<object>	Identifier associated with the object, must be unique
Description		

```
Returns a boolean value indicating whether the given property exists
on the specified object.

A property can exist without being set. The "-is_set" option causes
the command to return true only when the given property exists and
is set to any value (including a default value).

You should check for the existence of an object's property before
attempting to modify it, because getting an object property that does
not exist returns a Tcl error.
```

Example Usage

```
    qed::create_object -type project project_A -user_data {this is some data!}

    # Returns: True
    qed::has_property project_A -property qpf_path

    # Returns: False
    qed::has_property project_A -property qpf_path -is_set

    # Returns: True
    qed::has_property project_A -property user_data

    # Returns: True
    qed::has_property project_A -property user_data -is_set
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.29. qed::import_report_panel (::quartus::qed)

The following table displays information for the `qed::import_report_panel` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::import_report_panel [-h | -help] [-long_help] [-async] [-panel_name` *<panel_name>* `] [-regexp_match` *<regexp_match>* `] [-string_match` *<string_match>* `] [-timeout` *<timeout>* `] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-async`	Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	`-panel_name` *<panel_name>*	Name of the panel to retrieve
	`-regexp_match` *<regexp_match>*	Regex to match against all panels (first match is selected)
	`-string_match` *<string_match>*	Pattern to match against all panels (first match is selected)
	`-timeout` *<timeout>*	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<object>	Identifier associated with the object, must be unique

continued...

Description	``` Imports a report panel from a project or projects in a project group into the workspace. When you use the "-panel_name" option, you must specify the path to the report panel, separating report folder names with the "||" separator. For example, the panel name of the RAM summary report panel is "Fitter||Place Stage||Fitter RAM Summary". The "-string_match" option supports Tcl glob-style pattern matching. When multiple report panels match the "-string_match" pattern, only the first matching report panel is imported. The "-regexp_match" option supports Tcl regexp pattern matching. You should enclose your regular expression in curly braces. When multiple report panels match the "-regexp_match" option, only the first matching report panel is imported. Before you import a report panel, you must have successfully used the "qed::launch_connection" command to initiate background instances of the Quartus Prime Pro software, and you must have opened the projects. ```		
Example Usage	``` qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf qed::launch_connection project_A -open_project qed::import_report_panel project_A -panel_name "Timing Analyzer||Setup Summary" ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.30. qed::import_report_panel_names (::quartus::qed)

The following table displays information for the `qed::import_report_panel_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::import_report_panel_names [-h | -help] [-long_help] [-async] [-folders <folders> ] [-regexp_match <regexp_match> ] [-string_match <string_match> ] [-timeout <timeout> ] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-async`	Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	`-folders <folders>`	List of top-level folders to import panels from
	`-regexp_match <regexp_match>`	Regexp to match against remote panel names before returning them
	`-string_match <string_match>`	Pattern to match against remote panel names before returning them
	`-timeout <timeout>`	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<object>	Identifier associated with the object, must be unique
Description	``` Imports a list of report panel names from a project or projects in a project group. By default, all report panel names are imported. The "-string_match" option limits the panel names that are imported to ```	

continued...

	panel names that match a Tcl glob-style pattern. The "-regexp_match" option limits the panel names that are imported to panel names that match the given regular expression. Enclose the regular expression in curly braces. The "-folders" option limits the panel names that are imported to panel names that are contained in the given top-level folder names. Top-level folder names correspond to the folder names in the compilation report, such as Synthesis, Fitter, and Timing Analyzer. Specify multiple folder names as a Tcl list, such as { Synthesis Fitter "Timing Analyzer" }. When you call import_report_panel_names with a project object, the command returns a list of report panel names that match the filters or folder names. Any errors that occur are returned with standard Tcl error semantics. Before you import a report panel, you must have successfully used the "qed::launch_connection" command to initiate background instances of the Quartus Prime Pro software, and you must have opened the projects.
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf` `qed::launch_connection project_A -open_project` `qed::import_report_panel_names project_A`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.31. qed::is_connected (::quartus::qed)

The following table displays information for the `qed::is_connected` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::is_connected [-h | -help] [-long_help] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<object>	Identifier associated with the object, must be unique
Description	When you call the command with a project object, the return value is a boolean value indicating whether the given project is open in a background Quartus Prime Pro software instance. When you call the command with a project group object, the return value is a Tcl dict where the keys are IDs of the projects in the project group, and the values are the boolean values indicating whether each separate project is open in a background Quartus Prime Pro software instance. To determine whether the project group is connected, you must iterate through the Tcl dict and check each value.	
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf` `# Returns: false` `qed::is_connected project_A` `qed::launch_connection project_A` `# Returns: true` `qed::is_connected project_A`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.32. qed::is_workspace_open (::quartus::qed)

The following table displays information for the `qed::is_workspace_open` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::is_workspace_open [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Checks whether a QED workspace is currently open. Returns 1, if a workspace is currently open; returns 0, otherwise.`		
Example Usage	`qed::is_workspace_open`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.33. qed::launch_connection (::quartus::qed)

The following table displays information for the `qed::launch_connection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::launch_connection [-h | -help] [-long_help] [-open_project] [-open_timeout` *<open_timeout>* `] [-force_open] [-load_db_state] [-async] [-timeout` *<timeout>* `] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-open_project`	Flag to open the project once the connection is established
	`-open_timeout` *<open_timeout>*	Timeout for opening the project once the connection is established (0 = no timeout)
	`-force_open`	Flag to use -force when running project_open on the remote side
	`-load_db_state`	Flag to load the necessary compilation database state for the accessor executable once the connection is established and the project is open. For example, if the accessor executable is "quartus_sta", this initializes the timing netlist
	`-async`	Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	`-timeout` *<timeout>*	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<object>	Identifier associated with the object, must be unique

continued...

Description	

```
Starts one or more instances of the Quartus Prime Pro software running
in the background, to receive commands from the shell where the
workspace is open.

In most cases, you should use the "-open_project" option so that the
Quartus Prime Pro project is opened automatically after the software
instance has started. By default, the command does not open the
Quartus Prime Pro project after starting the instance of the software.

Use the "-timeout" option to specify the maximum time to wait for
the background instance of the Quartus Prime Pro software to start. The
timeout value is specified in seconds, and its default is 0, which
means to wait forever. Specifying "-timeout 0" is equivalent to
not using the "-timeout" option.

Use the "-load_db_state" option to prepare the project for
analysis after it opens. This option has no effect if you do not also
use the "-open_project" option, or if the project has no complete
compliation database. The initialization steps depend on the executable
that is started.

For example, when you use the Timing Analyzer
executable (quartus_sta), loading the compilation database follows the
initialization procedure in the Quartus Prime Timing Analyzer GUI:
 * create the timing netlist
 * read SDC files
 * update the timing netlist

The "-async" option causes the command to returns a token for use
with the "qed::get_return_value" command. By default the command will
wait, or block, until the background instances of the Quartus Prime
software have started, and any project opening or timing initialization
options have completed.

When you call "qed::launch_connection" with a project object without the
"-async" option, the command returns the value of whatever parts
of the initialization flow you have specified. Any errors that occur
starting the background software instance, opening the project, and
initializing timing, are returned with standard Tcl error semantics.

When you call "qed::launch_connection" with a project group object
without specifying the "-async" option, the command returns a Tcl dict where
keys are IDs of the projects in the project group, and the values are
two-element lists of the form { <code> <result> }. If the <code> value
is 1, the "launch_connection" command received an error starting
the background software instance, opening the project, or initializing
timing. When the <code> value is 1, the <result> contains the
received error message. If the <code> value is 0, the
"launch_connection" command was successful for the project, and
the <result> value contains the return value of whatever parts of the
initialization flow you have specified.

Information needed to start the Quartus Prime Pro background
software instances is set through the compute spec, which
is stored in an object property called compute_spec_args.
See qed::get_inherited_compute_spec_args for more details.

You can set the compute spec on the workspace object,
on a project group object, and on individual
project objects. The "qed::launch_connection" command reads
the compute specs with a defined priority to determine how to
start the background software instances. First the
workspace compute spec is read, then a project group compute
spec, then a project compute spec.
If a conflict of compute specs occurs, the compute spec of a
project object overrides the compute spec of the project group
it is in, and the compute spec of a project group overrides the
compute spec of the workspace. This priority allows you to specify
a default compute spec at the workspace level and override or
customize it if necessary for project groups or projects.

Because the Quartus Exploration Dashboard supports a variety of
compute farms, and each type of compute farm has unique,
related requirements, you must use a command unique to each
type of compute farm to configure it. Refer to help for the
appropriate commands described below for details.

 * To execute on the local machine running the Quartus
    Exploration Dashboard, use the "qed::configure_local_compute_spec"
    command
 * To execute through the LSF workload management software,
    use the "qed::configure_lsf_compute_spec" command
 * To execute by connecting to remote machines with the ssh command,
    use the "qed::configure_ssh_compute_spec" command
```

continued...

Example Usage	<pre>qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf qed::launch_connection project_A</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.34. qed::list_properties (::quartus::qed)

The following table displays information for the `qed::list_properties` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::list_properties [-h | -help] [-long_help] [-set_only] [-writeable_only] <object>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-set_only`	Flag indicating that any unset property names should be filtered out.	
	`-writeable_only`	Flag indicating that read-only property names should be filtered out.	
	<object>	Identifier associated with the object, must be unique	
Description	<pre>Returns a list of valid properties for the given object. When you create an object, some of its properties may not be set; they may have no value and no default value. When you use the "-set_only" option, the list of properties returned is limited to properties that are set. Some object properties are read-only, and the rest are read/write. When you use the "-writeable_only" option, the list of properties returned is limited to properties that are read/write. This is a utility command that makes it easier to find out what properties are valid for the given object.</pre>		
Example Usage	<pre>qed::create_object -type project project_A -user_data {this is some data!} qed::list_properties project_A -writeable_only qed::create_object -type group group_one -projects {project_A} qed::list_properties group_one -set_only</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.35. qed::load_db_state (::quartus::qed)

The following table displays information for the `qed::load_db_state` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::load_db_state [-h | -help] [-long_help] [-async] [-timeout <timeout> ] <object>`	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`	Long help with examples and possible return values
	`-async`	Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	`-timeout` <*timeout*>	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<*object*>	Identifier associated with the object, must be unique

Description

```
Prepares an opened project for deeper analysis. The specific operations
performed will depend on the executable used to access the project.
For example, if the project is accessed with "quartus_sta", timing
analysis preparation follows the initialization procedure in the
Quartus Prime Pro Timing Analyzer: creating the timing netlist, reading
any SDC files, and updating the timing netlist.

If the opened project does not have a completed compilation database,
this will return an error.

Use the "-timeout" option to specify the maximum time to wait for
the background instance of the Quartus Prime Pro software to initialize the
executable state on an opened project. The timeout value is specified in seconds,
and its default is 0, which means to wait forever. Specifying "-timeout 0" is
equivalent to not using the "-timeout" option.

The "-async" option causes the command to returns a token for use
with the "get_return_value" command. By default the command will
wait, or block, until the background instances of the Quartus Prime
software have started, and any project opening or timing initialization
options have completed.

When you call "qed::load_db_state" with a project group object
without specifying the "-async" option, the command returns a Tcl dict where
keys are IDs of the projects in the project group, and the values are
two-element lists of the form { <code> <result> }. If the <code> value
is 1, the "load_db_state" command received an error loading the compilation
database state. When the <code> value is 1, the <result> contains
the received error message. If the <code> value is 0, the
"load_db_state" command was successful for the project, and
the <result> value contains a message describing that the process succeeded.
```

Example Usage

```
    qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf

    qed::launch_connection project_A
    qed::open_project project_A
    qed::load_db_state project_A
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.31.36. qed::open_project (::quartus::qed)

The following table displays information for the `qed::open_project` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::open_project [-h | -help] [-long_help] [-force_open] [-load_db_state] [-async] [-timeout` <*timeout*> `]` <*object*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-force_open`	Flag to use -force when running project_open on the remote side

continued...

	`-load_db_state`	Flag to load the necessary compilation database state for the accessor executable once the connection is established and the project is open. For example, if the accessor executable is "quartus_sta", this initializes the timing netlist
	`-async`	Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	`-timeout` *<timeout>*	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	<object>	Identifier associated with the object, must be unique

Description

```
Use the "-timeout" option to specify the maximum time to wait for
the background instance of the Quartus Prime Pro software to open the project.
The timeout value is specified in seconds, and its default is 0, which
means to wait forever. Specifying "-timeout 0" is equivalent to
not using the "-timeout" option.

Use the "-load_db_state" option to prepare the project for deeper
analysis after it opens. The specific operations performed will depend
on the executable used to access the project. For example, if the
project is accessed with "quartus_sta", timing analysis preparation
follows the initialization procedure in the Quartus Prime Pro Timing
Analyzer: creating the timing netlist, reading any SDC files, and
updating the timing netlist.

The "-async" option causes the command to returns a token for use
with the "get_return_value" command. By default the command will
wait, or block, until the background instances of the Quartus Prime
software have started, and any project opening or timing initialization
options have completed.

When you call "qed::open_project" with a project group object without
specifying the "-async" option, the command returns a Tcl dict where
keys are IDs of the projects in the project group, and the values are
two-element lists of the form { <code> <result> }. If the <code> value
is 1, the "open_project" command received an error opening the project
or initializing the executable state. When the <code> value is 1,
the <result> contains the received error message. If the <code> value is 0,
the "open_project" command was successful for the project, and
the <result> value contains the return value of whatever parts of the
initialization flow you have specified.
```

Example Usage

```
qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf

qed::launch_connection project_A
qed::open_project project_A
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.31.37. qed::pop_from_property (::quartus::qed)

The following table displays information for the `qed::pop_from_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::pop_from_property [-h | -help] [-long_help] -property` *<property>* `-value` *<value>* `[-all] [-allow_missing]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-property` *<property>*	Name of the property to set

continued...

	`-value <value>`	Value to set the property to
	`-all`	Flag indicating that multiple instances of the value should all be removed (default: only the first is removed)
	`-allow_missing`	Flag to bypass the usual error if the value wasn't already present in the property's value
	<object>	Identifier associated with the object, must be unique
Description		

```
Specialized version of the "qed::set_property" command to modify
properties that have list values.

It is equivalent to retrieving the existing property value with the
"qed::get_property" command, removing the value from the returned list,
then calling "qed::set_property" to update the entire list value.

The "-all" argument removes all instances of the value from the list. Without
this flag, only the first instance of a potentially repeated value will be
removed.

The "-allow_missing" argument bypasses the error which is normally thrown
if the value to pop is not present in the list.

The command returns the updated value of the property.

The command returns a Tcl error if the given property does not exist.
```

Example Usage

```
    qed::create_object -type group   group_one
    qed::create_object -type group   group_two

    # Result: project_A is in groups {group_one}
    qed::create_object -type project project_A -groups {group_one}

    # Result: project_A is in groups {group_one group_two}
    qed::push_to_property project_A -property groups -value group_two

    # Result: project_A is in groups {group_two}
    qed::pop_from_property project_A -property groups -value group_one
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.38. qed::push_to_property (::quartus::qed)

The following table displays information for the `qed::push_to_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::push_to_property [-h | -help] [-long_help] -property <property> -value <value> [-unique] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-property <property>`	Name of the property to set
	`-value <value>`	Value to set the property to
	`-unique`	Flag indicating that the value shouldn't be appended if already present in the property
	<object>	Identifier associated with the object, must be unique
Description		

```
Specialized version of the "qed::set_property" command to modify
properties that have list values.
```

continued...

	It is equivalent to retrieving the existing property value with the "qed::get_property" command, appending the value to the returned list, then calling "qed::set_property" to update the entire list value. The "-unique" argument prevents the value from being added to the list if it is already present. Without this flag, the value will be added to the end of the list regardless of the other values already present. The command returns the updated value of the property. The command returns a Tcl error if the given property does not exist.		
Example Usage	`qed::create_object -type group group_one` `qed::create_object -type group group_two` `# Result: project_A is in groups {group_one}` `qed::create_object -type project project_A -groups {group_one}` `# Result: project_A is in groups {group_one group_two}` `qed::push_to_property project_A -property groups -value group_two` `# Result: project_A is in groups {group_two}` `qed::pop_from_property project_A -property groups -value group_one`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.39. qed::refresh_reports (::quartus::qed)

The following table displays information for the `qed::refresh_reports` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::refresh_reports [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	Causes the report window to update to reflect any changed project or project group reports.		
Example Usage	`qed::refresh_reports`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.40. qed::run_analysis (::quartus::qed)

The following table displays information for the `qed::run_analysis` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::run_analysis [-h | -help] [-long_help] [-arguments` *<arguments>* `] [-list_types] [-type` *<type>* `] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-arguments` *<arguments>*	Arguments that customize each report

continued...

	-list_types		Bypass running any analysis and return a list of valid types to run
	-type <*type*>		Type of analysis to run
	<*object*>		Identifier associated with the object, must be unique
Description	This command currently contains no help description.		
Example Usage	# Inside an opened workspace with a compute specification configured: qed::create_object -type project project_A -qpf_path /file/path/to/A/project.qpf qed::create_object -type project project_B -qpf_path /file/path/to/B/project.qpf qed::create_object -type group both_projects -projects {project_A project_B} qed::launch_connection group -open_project -load_db_state qed::generate_report group -type report_timing -arguments "-npaths 100 -from_clock sys_clk" qed::run_analysis group -type aggregate_report_timing_tables		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.41. qed::run_command (::quartus::qed)

The following table displays information for the qed::run_command Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	qed::run_command [-h | -help] [-long_help] [-async] [-timeout <*timeout*>] -cmd <*cmd*> <*object*>	
Arguments	-h | -help	Short help
	-long_help	Long help with examples and possible return values
	-async	Flag indicating not to wait for any issued remote commands to complete. With this flag, the return value is a token that can be passed to qed::get_return_value. By default, the return value of the remote command is reproduced (including any errors)
	-timeout <*timeout*>	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)
	-cmd <*cmd*>	Command to issue to the remote executable
	<*object*>	Identifier associated with the object, must be unique
Description	Communicates with background instances of the Quartus Prime Pro software to run commands and receive return values. When you call the "qed::run_command" command with a project object, the command is evaluated by the background instance of the Quartus Prime Pro software associated with that project. When you call the "qed::run_command" command with a project group object, the command is evaluated by the background instances of the Quartus Prime Pro software for all projects that are in the project group. Remember Tcl evaluation and variable substitution rules when you specify the value of the "-cmd" option. Enclosing the command value in quotes ("<command value>") allows the command to be interpreted by the Tcl interpreter of the workspace before it is sent to the background software instances. Enclosing the command value in curly braces ({<command value>}) delays interpretation until it is received by the background software instances.	

continued...

Example Usage	<pre>qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf qed::launch_connection project_A qed::run_command project_A -cmd "info hostname"</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.42. qed::sanitize_workspace (::quartus::qed)

The following table displays information for the `qed::sanitize_workspace` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::sanitize_workspace [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	<pre>Runs a variety of checks on projects and project groups to ensure they are in a "clean" state. After you run the "qed::sanitize_workspace" command, the following conditions will be true: * Every project will be part of at least one project group. If you have not created a project group, a default project group is created. * If any project objects exist in the workspace, at least one project group will exist. If you have not created a project group object, a default project group will be created. * No empty project groups will exist. Any project group that does not contain any projects will be deleted. You should use the "qed::sanitize_workspace" command after you add projects to the workspace, create a project group, edit properties of projects or project groups, or delete objects.</pre>	
Example Usage	<pre>qed::sanitize_workspace</pre>	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.43. qed::set_properties (::quartus::qed)

The following table displays information for the `qed::set_properties` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::set_properties [-h | -help] [-long_help] [-user_data` *<user_data>* `] [-groups` *<groups>* `] [-qpf_path` *<qpf_path>* `] [-revision` *<revision>* `] [-projects` *<projects>* `] [-default_group_id` *<default_group_id>* `] [-id` *<id>* `] [-from_dict` *<from_dict>* `]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-user_data` *<user_data>*	Freetext field to store any interesting metadata on the object. Use qed::set_user_data and qed::get_user_data to interact with the value as a dict instead of a string.

continued...

	`-groups` *<groups>*	Set of group IDs this project is a member of (Valid only for -type project)
	`-qpf_path` *<qpf_path>*	Full path to a .qpf file to open (Valid only for -type project)
	`-revision` *<revision>*	Name of the revision to open (Valid only for -type project)
	`-projects` *<projects>*	Set of projects belonging to the group (Valid only for -type group)
	`-default_group_id` *<default_group_id>*	Identifier used for the 'default group' that's created to house ungrouped projects during sanitize_workspace (Valid only for -type workspace)
	`-id` *<id>*	New identifier to associate with the object, must be unique
	`-from_dict` *<from_dict>*	Dictionary used to set the properties (key = property name, value = property value)
	<object>	Identifier associated with the object, must be unique
Description		

```
Allows you to modify multiple properties of an object, with a single
command, after the object has been created.

You must specify the multiple properties and new values in a Tcl dict.

Valid property and value strings depend on the type of object you
are updating.

Calling the "qed::set_properties" command with a dict of size 1 is
equivalent to calling the "qed::set_property" command.

The command returns a Tcl error if any of the dictionary keys do not
exist as properties of the given object.
```

Example Usage

```
    qed::create_object -type project project_A
    qed::set_properties project_a -qpf_path /path/to/project_B/project.qpf -revision project_B
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.44. qed::set_property (::quartus::qed)

The following table displays information for the `qed::set_property` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::set_property [-h | -help] [-long_help] -property` *<property>* `-value` *<value>* *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-property` *<property>*	Name of the property to set
	`-value` *<value>*	Value to set the property to
	<object>	Identifier associated with the object, must be unique
Description		

```
Allows you to modify a property of an object after it is created.

Valid property and value strings depend on the type of object you
are updating.

The "qed::set_properties" command supports setting multiple properties
of one object in a single command.
```

continued...

	The command returns the updated value of the property. The command returns a Tcl error if the given property does not exist.		
Example Usage	`qed::create_object -type project project_A` `qed::set_property project_a -property qpf_path -value /path/to/project_B/project.qpf`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.45. qed::set_user_data (::quartus::qed)

The following table displays information for the `qed::set_user_data` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::set_user_data [-h | -help] [-long_help] -key` *<key>* `-value` *<value>* *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-key` *<key>*	Arbitrary key to store inside the user_data property of the selected object
	`-value` *<value>*	Arbitrary value to store inside the user_data property of the selected object
	<object>	Identifier associated with the object, must be unique
Description	Modifies the "user_data" property of an object as if it is a dict, inserting or overwriting the value stored at the given key with the given value. It is possible to use this command in conjunction with "qed::set_property -property user_data", but whenever this method is invoked any existing value must be a legal Tcl dictionary (which includes the empty string).	
Example Usage	`qed::create_object -type project project_A` `qed::set_property -property user_data -value [dict create key_one value_1] project_A` `qed::set_user_data -key key_two -value value_2 project_A` `# Returns: {key_one value_1 key_two value_2}` `qed::get_property -property user_data project_A`	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.46. qed::wait_for_ready (::quartus::qed)

The following table displays information for the `qed::wait_for_ready` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::wait_for_ready [-h | -help] [-long_help] [-timeout` *<timeout>* `]` *<object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-timeout` *<timeout>*	Optional timeout for waiting for a return value in ms (default = 0 = no timeout)

continued...

	<object>	Identifier associated with the object, must be unique
Description	Blocks further script execution until all commands associated with the given object have completed or timed out.	
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf` `qed::launch_connection project_A` `qed::run_command project_A -cmd "after 1000; expr {100 * 100}"` `# Waits for ~1 second` `qed::wait_for_ready project_A`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful

3.1.31.47. qed::workspace_close (::quartus::qed)

The following table displays information for the `qed::workspace_close` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::workspace_close [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Closes an open QED workspace. Data in the workspace, such as projects, groups, and imported report panels is automatically saved in a database when the workspace closes.	
Example Usage	`qed::workspace_close`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful

3.1.31.48. qed::workspace_new (::quartus::qed)

The following table displays information for the `qed::workspace_new` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493	
Syntax	`qed::workspace_new [-h | -help] [-long_help] [-directory` *<directory>* `] [-overwrite] [` *<id>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-directory` *<directory>*	Directory to cd into before creating the workspace. This directory must exist prior to invoking this command.
	`-overwrite`	Overwrite any pre-existing workspace database and ensure a fresh workspace is created
	<id>	Identifier associated with the object, must be unique

continued...

Description	Creates and opens a new QED workspace with the specified workspace name. The command returns the ID of the new QED workspace object. The "qed::workspace_new" command returns an error if a QED workspace with the same name exists in the directory. Use the "-overwrite" option to overwrite the existing workspace. No QED workspace can be open when you create a new QED workspace; it is an error if a QED workspace is open when you use the "qed::workspace_new" command.		
Example Usage	`if { ! [qed::is_workspace_open] } {` `  qed::workspace_new my_workspace` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.49. qed::workspace_open (::quartus::qed)

The following table displays information for the `qed::workspace_open` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::workspace_open [-h | -help] [-long_help] [-force] [ <id> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-force`	Overwrite any pre-existing, non-compatible workspace database and ensure the workspace opens	
	<id>	Identifier associated with the object, must be unique	
Description	Opens an existing QED workspace. To create a new QED workspace, use the workspace_new command. A QED workspace is built on a Quartus Prime Pro project, so the workspace_open command gives an error when the compilation database version is not compatible with the current version of the Quartus Prime software. Use the "-force" option to overwrite the compilation database. If the specified workspace does not exist, the "qed::workspace_open" command returns a Tcl error.		
Example Usage	`# my_workspace must exist` `qed::workspace_open my_workspace`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.31.50. qed::write_object_reports_to_file (::quartus::qed)

The following table displays information for the `qed::write_object_reports_to_file` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qed on page 493		
Syntax	`qed::write_object_reports_to_file [-h | -help] [-long_help] [-append] -file` *<output file name>* `[-include_generated_panels]` *<object>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-append`		If output is sent to an ASCII file, this option appends the result to that file. Otherwise, the file will be overwritten.
	`-file` *<output file name>*		Name of output file to be generated
	`-include_generated_panels`		Flag to include QED-generated panels in the output
	<object>		Identifier associated with the object, must be unique
Description	`Writes all the report panels associated with the given object to an ASCII file.` `If you specify a project object, and do not use the "-include_generated_panels" option, only the report panels you imported from the project are written to the file.` `If you have created any new report panels based on any analysis in the Quartus Exploration Dashboard, you should use the "-include_generated_panels" option so those generated panels are written to the file.`		
Example Usage	`qed::create_object -type project project_A -qpf_path /file/path/to/project.qpf` `qed::launch_connection project_A -open_project` `qed::import_report_panel project_A -panel_name "Timing Analyzer||Setup Summary"` `qed::write_object_reports_to_file project_A -file project_A_reports.rpt`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.32. ::quartus::qmtf

The following table displays information for the **::quartus::qmtf** Tcl package:

Tcl Package and Version	::quartus::qmtf 1.0
Description	`This package contains no general description.`
Availability	`This package is not available in any Quartus Prime executable.`
Tcl Commands	`mtf::test (::quartus::qmtf)` on page 536

3.1.32.1. mtf::test (::quartus::qmtf)

The following table displays information for the `mtf::test` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qmtf on page 535		
Syntax	`mtf::test [-h | -help] [-long_help] -action <action>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-action <action>`		Action to test
Description	`Test MTF functions`		
Example Usage	`mtf::test -action say_hello`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.33. ::quartus::qshm

The following table displays information for the **::quartus::qshm** Tcl package:

Tcl Package and Version	::quartus::qshm 1.0
Description	`This package contains no general description.`
Availability	`This package is loaded by default in the following executables:` `qpro_sh` `quartus_sh`
Tcl Commands	qshm_connect_to_quartus (::quartus::qshm) on page 536 qshm_disconnect_from_quartus (::quartus::qshm) on page 537 qshm_dispose_client (::quartus::qshm) on page 537 qshm_get_hub_key_prefix (::quartus::qshm) on page 538 qshm_get_parent_hub_key (::quartus::qshm) on page 538 qshm_obtain_client (::quartus::qshm) on page 539 qshm_send_request (::quartus::qshm) on page 539 qshm_send_server_state_query (::quartus::qshm) on page 539 qshm_set_context (::quartus::qshm) on page 540

3.1.33.1. qshm_connect_to_quartus (::quartus::qshm)

The following table displays information for the `qshm_connect_to_quartus` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536	
Syntax	`qshm_connect_to_quartus [-h | -help] [-long_help] -hub_key_prefix <hub_key_prefix> -name <name> [-project <project> ] [-revision <revision> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-hub_key_prefix <hub_key_prefix>`	hub key prefix

continued...

	`-name <name>`	client name
	`-project <project>`	Quartus Prime project
	`-revision <revision>`	Quartus Prime project revision
Description	`Connects a client with the provided name to a Quartus Prime hub and returns the key of the dedicated server created by the hub for that client.`	
Example Usage	`## Connect a client with name client1 to a Quarts Prime hub with hub key prefix hub1` `qshm_connect_to_quartus -name client1 -hub_key_prefix hub1 -project <project> -revision <revision>`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful

3.1.33.2. qshm_disconnect_from_quartus (::quartus::qshm)

The following table displays information for the `qshm_disconnect_from_quartus` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536	
Syntax	`qshm_disconnect_from_quartus [-h | -help] [-long_help] -name <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	client name
Description	`Disconnect a client with the specified name from the Quarts Prime it is connected to.`	
Example Usage	`## Disconnect a client with name client1 from the Quarts Prime` `qshm_disconnect_from_quartus -name client1`	
Return Value	**Code Name**	**Code** / **String Return**
	TCL_OK	0 / INFO: Operation successful

3.1.33.3. qshm_dispose_client (::quartus::qshm)

The following table displays information for the `qshm_dispose_client` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536	
Syntax	`qshm_dispose_client [-h | -help] [-long_help] [-name <name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	Client name
Description	`Dispose and reclaim resources by the client with the specified name.`	

continued...

Example Usage	```## Dispose of a client with name client1 qshm_dispose_client -name client1```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.33.4. qshm_get_hub_key_prefix (::quartus::qshm)

The following table displays information for the `qshm_get_hub_key_prefix` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536	
Syntax	`qshm_get_hub_key_prefix [-h | -help] [-long_help] [-edition <edition> ] [-qroot_dir <qroot_dir> ] [-version <version> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-edition <edition>`	Quartus Prime edition
	`-qroot_dir <qroot_dir>`	Quartus Prime root directory
	`-version <version>`	Quartus Prime version
Description	```Returns the key prefix for a Quartus Prime hub based on the specified configuration arguments.```	
Example Usage	```## Create an SHM-based server type socket with the port name "trip" qshm_get_hub_key_prefix -qroot_dir /a/b/c -version 16.1 -edition "Quartus Prime Pro"```	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.33.5. qshm_get_parent_hub_key (::quartus::qshm)

The following table displays information for the `qshm_get_parent_hub_key` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536	
Syntax	`qshm_get_parent_hub_key [-h | -help] [-long_help] -name <name>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name <name>`	client name
Description	```Obtains the complete SHM key of the parent hub of the server that is connected to the client specified by the name argument```	
Example Usage		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.33.6. qshm_obtain_client (::quartus::qshm)

The following table displays information for the `qshm_obtain_client` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536		
Syntax	`qshm_obtain_client [-h | -help] [-long_help] [-name <name> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-name <name>`	Client name	
Description	`Creates and returns a client for the specified client name.`		
Example Usage	`## Create an SHM-based client type socket with the name "client1"` `qshm_obtain_client -name client1`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.33.7. qshm_send_request (::quartus::qshm)

The following table displays information for the `qshm_send_request` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536		
Syntax	`qshm_send_request [-h | -help] [-long_help] -cmd <cmd> -name <name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-cmd <cmd>`	command string	
	`-name <name>`	client name	
Description	`Sends the a request to the client with the specified name and waits to receive a response from the dedicated server the client is connected to.`		
Example Usage	`## Send a Tcl command to the client with the specified name. The context should have been set to include the request_type=tcl key value pair.` `qshm_send_request -name client1 -cmd "set x 5"`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.33.8. qshm_send_server_state_query (::quartus::qshm)

The following table displays information for the `qshm_send_server_state_query` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536
Syntax	`qshm_send_server_state_query [-h | -help] [-long_help] -name <name> -state_key <state_key>`

continued...

Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <name>`		client name
	`-state_key <state_key>`		state key string
Description			
	`Sends the a server state request to the client with the specified name and waits to receive a response from the state socket of the dedicated server the client is connected to.`		
Example Usage			
	`## Send a server state command using the client with specified name.` `qshm_send_server_state_query -name client1 -state_key device_family`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.33.9. qshm_set_context (::quartus::qshm)

The following table displays information for the `qshm_set_context` Tcl command:

Tcl Package and Version	Belongs to ::quartus::qshm on page 536		
Syntax	`qshm_set_context [-h | -help] [-long_help] -key <key> -name <name> -value <value>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-key <key>`		A context key
	`-name <name>`		client name
	`-value <value>`		A context value
Description			
	`Sets the a context key value pair for the client with the specified name.`		
Example Usage			
	`## Set a key value pair request_type=tcl for the client with the specified name.` `qshm_set_context -name client1 -key request_type -value tcl`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.34. ::quartus::report

The following table displays information for the **::quartus::report** Tcl package:

Tcl Package and Version	::quartus::report 2.1
Description	
	`This package contains a set of Tcl functions for accessing and updating information in the report database.`
Availability	
	`This package is loaded by default in the following executables:` `qacv` `quartus_fit` `quartus_ipgenerate` `quartus_sim`

continued...

	quartus_stp This package is available for loading in the following executables: qpro qpro_sh quartus quartus_cdb quartus_drc quartus_eda quartus_map quartus_sh quartus_si quartus_sta quartus_syn
Tcl Commands	add_row_to_table (::quartus::report) on page 541 create_report_panel (::quartus::report) on page 542 delete_report_panel (::quartus::report) on page 544 get_fitter_resource_usage (::quartus::report) on page 545 get_number_of_columns (::quartus::report) on page 547 get_number_of_rows (::quartus::report) on page 548 get_report_panel_column_index (::quartus::report) on page 549 get_report_panel_data (::quartus::report) on page 550 get_report_panel_id (::quartus::report) on page 552 get_report_panel_names (::quartus::report) on page 553 get_report_panel_row (::quartus::report) on page 554 get_report_panel_row_index (::quartus::report) on page 555 load_report (::quartus::report) on page 556 read_xml_report (::quartus::report) on page 557 refresh_report_window (::quartus::report) on page 558 save_report_database (::quartus::report) on page 558 unload_report (::quartus::report) on page 559 write_report_panel (::quartus::report) on page 560 write_xml_report (::quartus::report) on page 561

3.1.34.1. add_row_to_table (::quartus::report)

The following table displays information for the `add_row_to_table` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`add_row_to_table [-h | -help] [-long_help] [-id` *<table_id>* `] [-name` *<table_name>* `] <row>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id` *<table_id>*	Id of table to update
	`-name` *<table_name>*	Name of table to update
	<row>	Tcl list of strings to add to table
Description	Adds the list of strings to the table as a row. Using the panel id provides faster data access than using the panel name. Panel ids that you have cached may become outdated or invalid if the report gets unloaded or reloaded. This error occurs after compilation or with calls to the "project_close", "unload_report", and "load_report" commands. Panel names support wildcards. The table of content portion of the UI Compilation Report window shows short panel names for better readability. However, the panel name used by this command is the full panel name as shown in the right-hand side frame of the UI Compilation Report window or the .rpt file of the corresponding command-line executable. For example, the	

continued...

```
table of content shows the path "Analysis & Synthesis||Summary".
However, the corresponding full path used by this Tcl command
is "Analysis & Synthesis||Analysis & Synthesis Summary".
```

Example Usage

```
load_package report
project_open chiptrip
load_report

# Set panel name and id
set panel {Fitter||Fitter Settings}
set id [get_report_panel_id $panel]

if {$id != -1} {
    # If panel exists, add a row to it
    add_row_to_table -id $id {{New Field} Yes No}
    # Save the changes to the report database
    save_report_database
} else {
    # Otherwise print an error message
    puts "Error: Table $panel does not exist."
}

unload_report
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Number of elements conflict. The number of elements in the added row must be <*string*>.
TCL_ERROR	1	ERROR: Illegal color: <*string*>. Specify a color that is currently supported by add_row_to_table.
TCL_ERROR	1	ERROR: You specified <*string*> colors for the -<*string*> option. However, you must specify <*string*> colors to match the number of elements specified for the <<*string*>> argument.
TCL_ERROR	1	ERROR: Value specified for -<*string*> option is not a valid Tcl list: <*string*>. Specify a valid Tcl list.
TCL_ERROR	1	ERROR: You must open a project before you can use this command.
TCL_ERROR	1	ERROR: Can't find panel: <*string*>. Specify an existing report panel name.
TCL_ERROR	1	ERROR: Report not loaded for revision name: <*string*>. Type load_report to load the report.
TCL_ERROR	1	ERROR: Can't access report panel. Use this command only for report panels with rows or columns.
TCL_ERROR	1	ERROR: Value specified for <row> is not a valid Tcl list: <*string*>. Specify <row> as a valid Tcl list.
TCL_ERROR	1	ERROR: add_row_to_table command is not allowed for this report panel. You cannot add rows to this report panel.

3.1.34.2. create_report_panel (::quartus::report)

The following table displays information for the `create_report_panel` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`create_report_panel [-h | -help] [-long_help] [-folder] [-table] <panel_name>`	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`	Long help with examples and possible return values
	`-folder`	Option to create folder
	`-table`	Option to create table
	<panel_name>	Name of the panel to create

Description

```
Creates a new report panel with the specified name.

If -table option is specified, a table is created.
If -folder option is specified, a folder is created.

The name must be the full path to the new report panel, such as
"Fitter||My Table". If the specified panel is created successfully,
the corresponding panel id is returned.

The table of contents portion of the Compilation Report
window shows short panel names for better readability.
However, the panel name used by this command is the full
panel name as shown in the right-hand side frame of the
Compilation Report window or the .rpt file of the
corresponding command-line executable. For example, the
table of contents shows the path "Analysis & Synthesis||Summary".
However, the corresponding full path used by this Tcl command
is "Analysis & Synthesis||Analysis & Synthesis Summary".
```

Example Usage

```
load_package report
project_open chiptrip
load_report

# Set folder name and id
set folder    "My Folder"
set folder_id [get_report_panel_id $folder]

# Check if specified folder exists. If not, create it.
if {$folder_id == -1} {
    set folder_id [create_report_panel -folder $folder]
}

# Set table name and id
set table    "$folder||My Table"
set table_id [get_report_panel_id $table]

# Check if specified table exists. If so, delete it.
if {$table_id != -1} {
    delete_report_panel -id $table_id
}

# Create the specified table and get its id
set table_id [create_report_panel -table $table]

# Add Timing Analyzer Summary to the table
add_row_to_table -id $table_id {{Name} {Value}}
add_row_to_table -id $table_id {{Number of Registers} {100}}

# Save the changes to the report database
save_report_database

unload_report
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: The database file was not found at: <*string*>.
TCL_ERROR	1	ERROR: Report folder <*string*> already exists -- specify a different report folder name.
TCL_ERROR	1	ERROR: Can't find folder: <*string*>. Specify an existing report folder name.
TCL_ERROR	1	ERROR: Illegal color: <*string*>. Specify a color that is currently supported by add_row_to_table.
TCL_ERROR	1	ERROR: The input report object is not a table to be the master table of master details object

continued...

	TCL_ERROR	1	ERROR: The parent folder is not master details type <*string*>.
	TCL_ERROR	1	ERROR: The master details object already has the master table
	TCL_ERROR	1	ERROR: The -orientation option is only supported for -master_folder panels.
	TCL_ERROR	1	ERROR: Options -table, -tcl_table, -sql_table and -folder are mutually exclusive -- specify only one of the options.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Report object is not a folder: <*string*>. Specify an existing report folder name.
	TCL_ERROR	1	ERROR: Specify one of the options: -<*string*> or -<*string*>.
	TCL_ERROR	1	ERROR: Report not loaded for revision name: <*string*>. Type load_report to load the report.
	TCL_ERROR	1	ERROR: Report panel already exists: <*string*>. Specify a different report panel name.

3.1.34.3. delete_report_panel (::quartus::report)

The following table displays information for the `delete_report_panel` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`delete_report_panel [-h | -help] [-long_help] [-id` <*panel_id*> `] [-name` <*panel_name*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id` <*panel_id*>	Id of panel to delete
	`-name` <*panel_name*>	Name of panel to delete
Description		

```
Deletes the report panel with the specified id or name.
The panel can either be a report table or report folder.

Using the panel id provides faster data access than using the
panel name.

Panel ids that you have cached may become outdated or invalid
if the report is unloaded or reloaded. This error occurs
after compilation or with calls to the "project_close",
"unload_report", and "load_report" commands.

Panel names support wildcards.

The table of contents portion of the Compilation Report
window shows short panel names for better readability.
However, the panel name used by this command is the full
panel name as shown in the right-hand side frame of the
Compilation Report window or the .rpt file of the
corresponding command-line executable. For example, the
table of contents shows the path "Analysis & Synthesis||Summary".
However, the corresponding full path used by this Tcl command
is "Analysis & Synthesis||Analysis & Synthesis Summary".
```

Example Usage

```
load_package report
project_open chiptrip
load_report
```

continued...

```
# Set table name and id
set table    "Fitter||My Table"
set table_id [get_report_panel_id $table]

# Delete the table if the it already exists
if {$table_id != -1} {
    delete_report_panel -id $table_id
}

# Re-create the table
create_report_panel -table $table
add_row_to_table -name $table {{Name} {Value}}
add_row_to_table -name $table {{Number of Registers} {100}}

# This time, use table name instead of table id to delete it.
delete_report_panel -name $table

# Now, delete a folder
set folder    "My Folder"
set folder_id [get_report_panel_id $folder]

# Delete it if the specified folder already exists
if {$folder_id != -1} {
    delete_report_panel -id $folder_id
}

# Save the changes to the report database
save_report_database

unload_report
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Found conflicting panel options <*string*> and <*string*>. Specify only one of the options: -name or -id. If the panel name was not specified, then an unknown option was detected.
	TCL_ERROR	1	ERROR: The delete_report_panel command is not allowed for this report panel. You cannot delete a report panel folder.
	TCL_ERROR	1	ERROR: Illegal panel id: <*string*>. Specify a legal panel id.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: The option specified <*string*> , is unknown. Recheck the command options.
	TCL_ERROR	1	ERROR: Specify one of the options: -name or -id.
	TCL_ERROR	1	ERROR: Can't find panel: <*string*>. Specify an existing report panel name.
	TCL_ERROR	1	ERROR: Report not loaded for revision name: <*string*>. Type load_report to load the report.

3.1.34.4. get_fitter_resource_usage (::quartus::report)

The following table displays information for the `get_fitter_resource_usage` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`get_fitter_resource_usage [-h | -help] [-long_help] [-alm] [-alut] [-available] [-io_pin] [-lab] [-le] [-mem_bit] [-percentage] [-reg] [-used] [-utilization]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-alm`	Get total adaptive logic modules
	`-alut`	Get total Combinational ALUTs
	`-available`	Get available resource summary
	`-io_pin`	Get total I/O pins
	`-lab`	Get total logic array blocks
	`-le`	Get total logic elements
	`-mem_bit`	Get total memory bits
	`-percentage`	Get used resource summary in percentage
	`-reg`	Get total registers
	`-used`	Get used resource summary
	`-utilization`	Get total logic utilization

Description

```
Gets the Fitter resource usage results.

You must use one of the following options: "-alut", "-reg", "-le",
"-alm", "-lab", "-io_pin", "-mem_bit" or "-resource".

If the above option is not "-resource", you may also optionally use
one of the following options: "-used", "-available" or "-percentage".

Option "-resource" takes resource name as parameter, which supports wildcards.
```

Example Usage

```
load_package report
project_open chiptrip
load_report

# Shortcut of get_fitter_resource_usage command
set cmd     get_fitter_resource_usage

# Get total registers, logic elements, and DSP block 9-bit elements.
set registers [$cmd -reg]
set alms      [$cmd -alm]
set io_pin    [$cmd -io_pin -available]
set mem_bit   [$cmd -mem_bit -percentage]
set dsp       [$cmd -resource "DSP block 9*"]
puts "Registers usage: $registers"
puts "Total used ALMs: $alms"
puts "Total available I/O pins: $io_pin"
puts "Total used memory bits in percentage: ${mem_bit}%"
puts "DSP block 9-bit elements: $dsp"

unload_report
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Command requires one of the following options: -alut, -reg, -le, -alm, -lab, -io_pin, mem_bit or -resource. Specify one of the options.
TCL_ERROR	1	ERROR: Options -used, -available and -percentage can't be used together. Specify either one or none of the options.
TCL_ERROR	1	ERROR: Option -resource was used with option -used, -available or -percentage. Use option -resource only.
TCL_ERROR	1	ERROR: Can't find panel: <*string*>. Make sure the project was compiled by quartus_fit and the panel was not deleted.

3.1.34.5. get_number_of_columns (::quartus::report)

The following table displays information for the `get_number_of_columns` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540		
Syntax	`get_number_of_columns [-h | -help] [-long_help] [-id <table_id> ] [-name <table_name> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-id <table_id>`	Id of panel from which to get data	
	`-name <table_name>`	Name of panel from which to get data	
Description	`Returns the number of columns for the specified panel.` `Using the panel id provides faster data access than using the panel name.` `Panel ids that you have cached may become outdated or invalid if the report is unloaded or reloaded. This error occurs after compilation or with calls to the "project_close", "unload_report", and "load_report" commands.` `Panel names support wildcards.` `The table of contents portion of the Compilation Report window shows short panel names for better readability. However, the panel name used by this command is the full panel name as shown in the right-hand side frame of the Compilation Report window or the .rpt file of the corresponding command-line executable. For example, the table of contents shows the path "Analysis & Synthesis||Summary". However, the corresponding full path used by this Tcl command is "Analysis & Synthesis||Analysis & Synthesis Summary".`		
Example Usage	`## Loop through a panel row and print out all its element information` `load_package report` `project_open chiptrip` `load_report` `# Set panel name and id` `set panel   {*Input Pins}` `set id      [get_report_panel_id $panel]` `if {$id == -1} {` `error( "Panel not found")` `}` `# Get the number of columns` `set col_cnt [get_number_of_columns -id $id]` `# Set row name and get row index` `set rname   {*clk1*}` `set rindex  [get_report_panel_row_index -id $id $rname]` `set rname   [get_report_panel_data -row $rindex -col 0 -id $id]` `puts "\[Input Pins - $rname\]"` `for {set i 1} {$i < $col_cnt} {incr i} {` `set result "[get_report_panel_data -row 0 -col $i -id $id]: "` `append result [get_report_panel_data -row $rindex -col $i -id $id]` `puts $result` `}` `unload_report` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_OK	0	INFO: Report automatically reloaded because it was not up-to-date after use of Tcl command execute_flow or execute_module (which belong to ::quartus::flow package). No action is required.
	TCL_ERROR	1	ERROR: Can't find the top-level panel. Make sure the loaded report is not empty. You can run a successful compilation to rebuild the report.
	TCL_ERROR	1	ERROR: Can't find panel: <*string*>. Specify an existing report panel name.
	TCL_ERROR	1	ERROR: Report not loaded for revision name: <*string*>. Type load_report to load the report.
	TCL_ERROR	1	ERROR: Can't access report panel. Use this command only for report panels with rows or columns.

3.1.34.6. get_number_of_rows (::quartus::report)

The following table displays information for the `get_number_of_rows` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`get_number_of_rows [-h | -help] [-long_help] [-id` <*table_id*> `] [-name` <*table_name*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id` <*table_id*>	Id of panel from which to get data
	`-name` <*table_name*>	Name of panel from which to get data
Description		

```
Returns the number of rows for the specified panel.

Using the panel id provides faster data access than using the
panel name.

Panel ids that you have cached may become outdated or invalid
if the report is unloaded or reloaded. This error occurs
after compilation or with calls to the "project_close",
"unload_report", and "load_report" commands.

Panel names support wildcards.

The table of contents portion of the Compilation Report
window shows short panel names for better readability.
However, the panel name used by this command is the full
panel name as shown in the right-hand side frame of the
Compilation Report window or the .rpt file of the
corresponding command-line executable. For example, the
table of contents shows the path "Analysis & Synthesis||Summary".
However, the corresponding full path used by this Tcl command
is "Analysis & Synthesis||Analysis & Synthesis Summary".
```

Example Usage

```
## Loop through a panel and print out all its row information

load_package report
project_open chiptrip
load_report

# Set panel name and id
set panel   {*Fitter Settings}
set id      [get_report_panel_id $panel]

# Get the number of rows
set row_cnt [get_number_of_rows -id $id]

for {set i 0} {$i < $row_cnt} {incr i} {
    puts [get_report_panel_row -row $i -id $id]
}
```

continued...

	`` unload_report project_close ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Report automatically reloaded because it was not up-to-date after use of Tcl command execute_flow or execute_module (which belong to ::quartus::flow package). No action is required.
	TCL_ERROR	1	ERROR: Can't find the top-level panel. Make sure the loaded report is not empty. You can run a successful compilation to rebuild the report.
	TCL_ERROR	1	ERROR: Can't find panel: <*string*>. Specify an existing report panel name.
	TCL_ERROR	1	ERROR: Report not loaded for revision name: <*string*>. Type load_report to load the report.
	TCL_ERROR	1	ERROR: Can't access report panel. Use this command only for report panels with rows or columns.

3.1.34.7. get_report_panel_column_index (::quartus::report)

The following table displays information for the `get_report_panel_column_index` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`get_report_panel_column_index [-h | -help] [-long_help] [-id` <*table_id*> `] [-name` <*table_name*> `] <`*col_name*`>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id` <*table_id*>	Id of panel from which to get column index
	`-name` <*table_name*>	Name of panel from which to get column index
	<*col_name*>	Column name
Description	``` Gets the column index of the specified panel column name. Column name refers to the specified name in the header row. The column index is a non-negative integer for an existing column. The column index is -1 if the column name is not found in the panel. Using the column index and panel id provides faster data access than using column name and panel name. Column indices and panel ids that you have cached may become outdated or invalid if the report is unloaded or reloaded. This error occurs after compilation or with calls to the "project_close", "unload_report", and "load_report" commands. Column and panel names support wildcards. The table of contents portion of the Compilation Report window shows short panel names for better readability. However, the panel name used by this command is the full panel name as shown in the right-hand side frame of the Compilation Report window or the .rpt file of the corresponding command-line executable. For example, the table of content shows the path "Analysis & Synthesis||Summary". However, the corresponding full path used by this Tcl command is "Analysis & Synthesis||Analysis & Synthesis Summary". ```	

continued...

Example Usage	<pre>load_package report project_open chiptrip load_report # Set panel name and id set panel {*Input Pins} set id [get_report_panel_id $panel] # Set column name and index set cname {Pin #} set cindex [get_report_panel_column_index -id $id $cname] # Get data out of the specified panel set clock [get_report_panel_data -id $id -row_name clk1 -col $cindex] set enable [get_report_panel_data -id $id -row_name enable -col $cindex] # Output results puts "$cname of clock: $clock" puts "$cname of enable: $enable" unload_report project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.34.8. get_report_panel_data (::quartus::report)

The following table displays information for the `get_report_panel_data` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`get_report_panel_data [-h | -help] [-long_help] [-col` *<column>* `] [-col_name` *<column_name>* `] [-id` *<table_id>* `] [-name` *<table_name>* `] [-row` *<row>* `] [-row_name` *<row_name>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-col` *<column>*	column (or X) coordinate
	`-col_name` *<column_name>*	column (or X) name
	`-id` *<table_id>*	id of panel from which to get data
	`-name` *<table_name>*	Name of panel from which to get data
	`-row` *<row>*	row (or Y) coordinate
	`-row_name` *<row_name>*	row (or Y) name
Description	<pre>Returns non-empty data for the specified row and column of the specified panel. If the data is empty or if the row, column, or panel do not exist, an error will be generated. To properly handle the error, make sure to catch the result as in the following example: if [catch {set data [get_report_panel_data ...]} result] { puts "No data found" } else { puts "Got $data" } Using the panel id and row and column indices provides faster data access than using panel, row, and column names. Panel ids and row and column indices that you have cached may become outdated or invalid if the report is unloaded or reloaded. This error occurs after compilation or with calls to the "project_close", "unload_report", and "load_report" commands.</pre>	

continued...

```
Panel, row, and panel names support wildcards.

The table of contents portion of the Compilation Report
window shows short panel names for better readability.
However, the panel name used by this command is the full
panel name as shown in the right-hand side frame of the
Compilation Report window or the .rpt file of the
corresponding command-line executable. For example, the
table of contents shows the path "Analysis & Synthesis||Summary".
However, the corresponding full path used by this Tcl command
is "Analysis & Synthesis||Analysis & Synthesis Summary".
```

Example Usage

```
load_package report
project_open chiptrip
load_report

# Set panel name and id
set panel {Flow Settings}
set id    [get_report_panel_id $panel]
# Set row name
set rname {Revision Name}

# Get row 2 - column 1 data
get_report_panel_data -row 2 -col 1 -id $id
# Get row {Revision Name} - column 1 data
get_report_panel_data -row_name $rname -col 1 -id $id
# Get row {Revision Name} - column {Setting} data
get_report_panel_data -row_name $rname -col_name Setting -id $id
# If unsure the case of a row or column name, use glob-style pattern
get_report_panel_data -row_name {[Rr]evision*} -col_name {[Ss]etting} -id $id

unload_report
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_OK	0	INFO: Report automatically reloaded because it was not up-to-date after use of Tcl command execute_flow or execute_module (which belong to ::quartus::flow package). No action is required.
TCL_ERROR	1	ERROR: Found conflicting column options. Specify either the -col or -col_name option.
TCL_ERROR	1	ERROR: Found conflicting row options. Specify either the -row or -row_name option.
TCL_ERROR	1	ERROR: Table is empty. Add rows to the table before accessing its data.
TCL_ERROR	1	ERROR: Illegal column name: <*string*>.
TCL_ERROR	1	ERROR: Illegal column number: <*string*>. Specify a legal column number from <*string*> to <*string*>.
TCL_ERROR	1	ERROR: Illegal row name: <*string*>.
TCL_ERROR	1	ERROR: Illegal row number: <*string*>. Specify a legal row number from <*string*> to <*string*>.
TCL_ERROR	1	ERROR: No column option specified.
TCL_ERROR	1	ERROR: Can't retrieve data for row and column number. Specify a legal row and column number for the command get_report_panel_data.
TCL_ERROR	1	ERROR: No row option specified.
TCL_ERROR	1	ERROR: Can't access report panel. Use this command only for report panels with rows or columns.

3.1.34.9. get_report_panel_id (::quartus::report)

The following table displays information for the `get_report_panel_id` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540		
Syntax	`get_report_panel_id [-h | -help] [-long_help] <name>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<name>	Name of panel for which to get id	
Description			
Example Usage			
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

```
Gets the id of a panel with the specified name.

The panel id is a non-negative integer for an existing panel.
If the specified panel cannot be found, the id is -1.

Using the panel id provides faster data access than using the
panel name.

You can use the "get_report_panel_id" command to check for the
existence of a panel.(Refer to the example under the
"get_report_panel_id" command.)

Panel ids that you have cached may become outdated or invalid
if the report is unloaded or reloaded. This error occurs
after compilation or with calls to the "project_close",
"unload_report", and "load_report" commands.

Panel names support wildcards.

The table of contents portion of the Compilation Report
window shows short panel names for better readability.
However, the panel name used by this command is the full
panel name as shown in the right-hand side frame of the
Compilation Report window or the .rpt file of the
corresponding command-line executable. For example, the
table of contents shows the path "Analysis & Synthesis||Summary".
However, the corresponding full path used by this Tcl command
is "Analysis & Synthesis||Analysis & Synthesis Summary".
```

```
load_package report
project_open chiptrip
load_report

# Set panel name
set panel {Fitter||Fitter Settings}
# Get the panel id
set id [get_report_panel_id $panel]

if {$id != -1} {
    # If panel exists, add a row to it
    add_row_to_table -id $id {{New Field} Yes No}
    # Save the changes to the report database
    save_report_database
} else {
    # Otherwise print an error message
    puts "Error: Table $panel does not exist."
}

unload_report
project_close
```

3.1.34.10. get_report_panel_names (::quartus::report)

The following table displays information for the `get_report_panel_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540		
Syntax	`get_report_panel_names [-h | -help] [-long_help] [-filter_name <filter_name> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-filter_name <filter_name>`	filter the returned list to match the specified panel names	
Description	`Returns a list of panel names for the current report.` `The table of contents portion of the Compilation Report window shows short panel names for better readability. However, each panel name returned by this command is the full panel name as shown in the right-hand side frame of the Compilation Report window or the .rpt file of the corresponding command-line executable. For example, the table of contents shows the path "Analysis & Synthesis||Summary". However, the corresponding full path returned by this Tcl command is "Analysis & Synthesis||Analysis & Synthesis Summary".`		
Example Usage	`## Load report database and write Timing Analyzer Summary` `## panel to file in HTML format` `load_package report` `project_open chiptrip` `load_report` `# Set panel name` `set fmax_panel "Timing Analyzer Summary"` `# Iterate through all the accessable panels` `foreach panel [get_report_panel_names] {` `  # If find the panel '*Timing Analyzer Summary',` `  # write its content to file fmax.htm` `  if {[string match "*$fmax_panel" $panel] == 1} {` `  write_report_panel -file fmax.htm -html $panel` `  break` `  }` `}` `unload_report` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Report automatically reloaded because it was not up-to-date after use of Tcl command execute_flow or execute_module (which belong to ::quartus::flow package). No action is required.
	TCL_ERROR	1	ERROR: Illegal report panel type.
	TCL_ERROR	1	ERROR: Can't find panel: <*string*>. Specify an existing report panel name.
	TCL_ERROR	1	ERROR: Report not loaded for revision name: <*string*>. Type load_report to load the report.

3.1.34.11. get_report_panel_row (::quartus::report)

The following table displays information for the `get_report_panel_row` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540		
Syntax	`get_report_panel_row [-h | -help] [-long_help] [-id <table_id> ] [-name <table_name> ] [-row <row> ] [-row_name <row_name> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-id <table_id>`	Id of panel from which to get data	
	`-name <table_name>`	Name of panel from which to get data	
	`-row <row>`	Row (or Y) coordinate	
	`-row_name <row_name>`	Row (or Y) name	
Description	`Reports data of the specified panel row.` `Using the panel id and row index provides faster data access` `than using panel name and row name, respectively.` `Panel ids and row indices that you have cached may become outdated` `or invalid if the report gets unloaded or reloaded. This error` `occurs after compilation or with calls to the "project_close",` `"unload_report", and "load_report" commands.` `Panel and row names support wildcards.` `The table of content portion of the UI Compilation Report` `window shows short panel names for better readability.` `However, the panel name used by this command is the full` `panel name as shown in the right-hand side frame of the` `UI Compilation Report window or the .rpt file of the` `corresponding command-line executable. For example, the` `table of content shows the path "Analysis & Synthesis||Summary".` `However, the corresponding full path used by this Tcl command` `is "Analysis & Synthesis||Analysis & Synthesis Summary".`		
Example Usage	`load_package report` `project_open chiptrip` `load_report` `# Set panel name and id` `set panel {Flow Settings}` `set id    [get_report_panel_id $panel]` `# Set row name` `set rname {Revision Name}` `# Get row {Revision Name} data (use panel id)` `get_report_panel_row -row_name $rname -id $id` `# Get row 2 data (use panel id)` `get_report_panel_row -row 2 -id $id` `# Get row 3 data (use panel name)` `get_report_panel_row -row 3 -name $panel` `## Get the last row data` `set row_cnt     [get_number_of_rows -id $id]` `set last_rindex [expr $row_cnt - 1]` `get_report_panel_row -row $last_rindex -id $id` `unload_report` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_OK	0	INFO: Report automatically reloaded because it was not up-to-date after use of Tcl command execute_flow or execute_module (which belong to ::quartus::flow package). No action is required.
	TCL_ERROR	1	ERROR: Found conflicting row options. Specify either the -row or -row_name option.
	TCL_ERROR	1	ERROR: Table is empty. Add rows to the table before accessing its data.
	TCL_ERROR	1	ERROR: Illegal row name: <*string*>.
	TCL_ERROR	1	ERROR: Illegal row number: <*string*>. Specify a legal row number from <*string*> to <*string*>.
	TCL_ERROR	1	ERROR: No row option specified.
	TCL_ERROR	1	ERROR: The option specified <*string*> , is unknown. Recheck the command options.
	TCL_ERROR	1	ERROR: Can't access report panel. Use this command only for report panels with rows or columns.

3.1.34.12. get_report_panel_row_index (::quartus::report)

The following table displays information for the `get_report_panel_row_index` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`get_report_panel_row_index [-h | -help] [-long_help] [-id` <*table_id*> `] [-name` <*table_name*> `]` <*row_name*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-id` <*table_id*>	Id of panel from which to get row index
	`-name` <*table_name*>	Name of panel from which to get row index
	<*row_name*>	Row name
Description		

```
Gets the row index of the specified panel row name.

Row name refers to the first element content of the specified
row. The row index is a non-negative integer for an existing row.
Row index is -1 if the row name is not found in the panel.

Using the row index and panel id provides faster data access
than using row name and panel name.

Row indices and panel ids that you have cached may become outdated
or invalid if the report is unloaded or reloaded. This error
occurs after compilation or with calls to the "project_close",
"unload_report", and "load_report" commands.

Row and panel names support wildcards.

The table of contents portion of the Compilation Report
window shows short panel names for better readability.
However, the panel name used by this command is the full
panel name as shown in the right-hand side frame of the
Compilation Report window or the .rpt file of the
corresponding command-line executable. For example, the
table of contents shows the path "Analysis & Synthesis||Summary".
However, the corresponding full path used by this Tcl command
is "Analysis & Synthesis||Analysis & Synthesis Summary".
```

continued...

Example Usage	<pre>load_package report project_open chiptrip load_report # Set panel name and id set panel {*Input Pins} set id [get_report_panel_id $panel] # Set row name and index set rname {[Cc]lk1} set rindex [get_report_panel_row_index -id $id $rname] # Get data out of the specified panel set pc_str [get_report_panel_data -id $id -row 0 -col 1] set pin_cnt [get_report_panel_data -id $id -row $rindex -col 1] set iob_str [get_report_panel_data -id $id -row 0 -col 2] set io_bank [get_report_panel_data -id $id -row $rindex -col 2] # Output results puts "$pc_str: $pin_cnt" puts "$iob_str: $io_bank" unload_report project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.34.13. load_report (::quartus::report)

The following table displays information for the `load_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`load_report [-h | -help] [-long_help] [-simulator]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-simulator`	Option to load the Simulation Report. If this option isn't specified, the Compilation Report is loaded instead.
Description	<pre>By default, loads the Compilation Report for the current revision or the specified revision name. If the -simulator option is specified, loads the Simulation Report instead. After the report is loaded or reloaded, the cached panel ids, and row and column indices may become outdated or invalid. Intel recommends that you update them before using them.</pre>	
Example Usage	<pre># Load report package load_package report # Open chiptrip project project_open chiptrip # Load the current revision report load_report # Set panel name and id set panel {Fitter||Fitter Summary} set id [get_report_panel_id $panel] # Get total registers set rname {Total [Rr]egisters} set rindex [get_report_panel_row_index -id $id $rname] set rname [get_report_panel_data -id $id -row $rindex -col 0] set data [get_report_panel_data -id $id -row $rindex -col 1] puts "$rname: $data" # Get total pins set rname {Total [Pp]ins} set rindex [get_report_panel_row_index -id $id $rname] set rname [get_report_panel_data -id $id -row $rindex -col 0] set data [get_report_panel_data -id $id -row $rindex -col 1] puts "$rname: $data"</pre>	

continued...

	`` # Unload the report unload_report # Close the project project_close ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't load report for revision: <*string*>. A different, active revision already exists: <*string*>. To specify another active revision name, type set_current_revision <*string*>.
	TCL_ERROR	1	ERROR: Can't find active revision. Specify an active revision name using set_current_revision <*string*>.
	TCL_ERROR	1	ERROR: Can't find active revision. Specify an active revision name using set_current_revision <revision name>.
	TCL_ERROR	1	ERROR: Revision name does not exist: <*string*>. Run Analysis & Synthesis (quartus_map) with the specified revision name before using this Tcl command.
	TCL_ERROR	1	ERROR: Compilation database is newer than the last call to create_timing_netlist. Use the delete_timing_netlist Tcl command before using <*string*>.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Can't load report data for revision name: <*string*>. Make sure the report database exists for the specified revision name.
	TCL_ERROR	1	ERROR: Existing report was not unloaded. Ensure that your load_report commands have matching unload_report commands.

3.1.34.14. read_xml_report (::quartus::report)

The following table displays information for the `read_xml_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`read_xml_report [-h | -help] [-long_help] <filename>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<*filename*>	Name of XML Report Database File from which to read
Description	``` Creates the current Compilation Report from the specified XML Report Database File (.xml). ```	
Example Usage	``` # Set project name set project_name "chiptrip" load_package report project_open $project_name # Read XML Report Database File (.xml) read_xml_report $project_name.xml load_report # Get all the panel names puts {All Report Panel Names:} puts [get_report_panel_names] ```	

continued...

	`unload_report` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Report read from XML Report Database File: *<string>*. No action is required.
	TCL_ERROR	1	ERROR: Can't read XML Report Database File: *<string>* . Make sure that the file exists and does not contain syntax errors.

3.1.34.15. refresh_report_window (::quartus::report)

The following table displays information for the `refresh_report_window` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`refresh_report_window [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Refresh the Report Window.`	
Example Usage	`refresh_report_window`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.34.16. save_report_database (::quartus::report)

The following table displays information for the `save_report_database` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`save_report_database [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Saves the report database, including any new report panel.`	
Example Usage	`load_package report` `project_open chiptrip` `load_report` `# Set panel name and id` `set panel {Fitter||Fitter Settings}` `set id    [get_report_panel_id $panel]` `# If panel exists, add a row to it. Otherwise, print an error message.` `if {$id != -1} {` `    add_row_to_table -id $id {{New Field} Yes No}` `    # Save the changes to the report database`	

continued...

```
    save_report_database
} else {
    puts "Error: Table $panel does not exist."
}

unload_report
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Report not loaded for revision name: <*string*>. Type load_report to load the report.

3.1.34.17. unload_report (::quartus::report)

The following table displays information for the `unload_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`unload_report [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description		

```
Unloads the report for the current revision or the specified
revision name.

After the report is loaded or reloaded, the cached panel ids,
and row and column indices, may become outdated or invalid.
Intel recommends that you update them before using them.
```

Example Usage

```
# Load report package
load_package report
# Open chiptrip project
project_open chiptrip
# Load the current revision report
load_report

# Set panel name and id
set panel {Fitter||Fitter Summary}
set id    [get_report_panel_id $panel]

# Get total registers
set rname  {Total [Rr]egisters}
set rindex [get_report_panel_row_index -id $id $rname]
set rname  [get_report_panel_data -id $id -row $rindex -col 0]
set data   [get_report_panel_data -id $id -row $rindex -col 1]
puts "$rname: $data"

# Get total pins
set rname  {Total [Pp]ins}
set rindex [get_report_panel_row_index -id $id $rname]
set rname  [get_report_panel_data -id $id -row $rindex -col 0]
set data   [get_report_panel_data -id $id -row $rindex -col 1]
puts "$rname: $data"

# Unload the report
unload_report
# Close the project
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

continued...

	TCL_ERROR	1	ERROR: Can't load report for revision: <*string*>. A different, active revision already exists: <*string*>. To specify another active revision name, type set_current_revision <*string*>.
	TCL_ERROR	1	ERROR: Can't find active revision. Specify an active revision name using set_current_revision <*string*>.
	TCL_ERROR	1	ERROR: Can't find active revision. Specify an active revision name using set_current_revision <revision name>.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.

3.1.34.18. write_report_panel (::quartus::report)

The following table displays information for the `write_report_panel` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`write_report_panel [-h | -help] [-long_help] [-append] -file` <*output file name*> `[-html] [-id` <*table_id*> `] [-name` <*table_name*> `] [-xml] [` <*name*> `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to an ASCII file, this option appends the result to that file. Otherwise, the file will be overwritten. This option is not supported for HTML or XML files.
	`-file` <*output file name*>	Name of output file to be generated
	`-html`	Option to generate output file in HTML format
	`-id` <*table_id*>	id of panel from which to get data
	`-name` <*table_name*>	Name of panel from which to get data
	`-xml`	Option to generate output file in XML format
	<*name*>	Name of panel from which to get data
Description		

```
Writes data from the specified report panel to the specified
output file. If the "-html" option is specified, the output file
is generated in HTML format. If the "-xml" option is specified,
the output file is generated in XML format. Otherwise, the output
file is generated in ASCII format.

For ASCII formatted files, "-append" preserves the existing file
and adds the panel to the end. Otherwise, the file is overwritten.
HTML and XML files are always overwritten.

Using the panel id provides faster data access than using the
panel name.

Panel ids that you have cached may become outdated or invalid
if the report is unloaded or reloaded. This error occurs
after compilation or with calls to the "project_close",
"unload_report", and "load_report" commands.

Panel names support wildcards.

The table of contents portion of the Compilation Report
window shows short panel names for better readability.
However, the panel name used by this command is the full
panel name as shown in the right-hand side frame of the
Compilation Report window or the .rpt file of the
corresponding command-line executable. For example, the
table of contents shows the path "Analysis & Synthesis||Summary".
However, the corresponding full path used by this Tcl command
is "Analysis & Synthesis||Analysis & Synthesis Summary".
```

continued...

Example Usage	```## Load report database and write Timing Analyzer Summary ## panel to file in HTML format load_package report project_open chiptrip load_report # Set panel name and id set panel "*Timing Analyzer Summary" set id [get_report_panel_id $panel] # If the specified panel exists, write it to # fmax.htm and fmax.xml. # Otherwise, print out an error message if {$id != -1} { write_report_panel -file fmax.htm -html -id $id write_report_panel -file fmax.xml -xml -id $id } else { puts "Error: report panel could not be found." } unload_report project_close```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Successfully generated HTML-Format Report File: *<string>*
	TCL_OK	0	INFO: Successfully generated output file: *<string>*
	TCL_OK	0	INFO: Successfully generated XML-Format Report File: *<string>*
	TCL_OK	0	INFO: Report automatically reloaded because it was not up-to-date after use of Tcl command execute_flow or execute_module (which belong to ::quartus::flow package). No action is required.
	TCL_ERROR	1	ERROR: Can't create or overwrite file: *<string>*. Specify a file name that has write permission.
	TCL_ERROR	1	ERROR: Can't find the top-level panel. Make sure the loaded report is not empty. You can run a successful compilation to rebuild the report.
	TCL_ERROR	1	ERROR: Can't find panel: *<string>*. Specify an existing report panel name.
	TCL_ERROR	1	ERROR: Report not loaded for revision name: *<string>*. Type load_report to load the report.

3.1.34.19. write_xml_report (::quartus::report)

The following table displays information for the `write_xml_report` Tcl command:

Tcl Package and Version	Belongs to ::quartus::report on page 540	
Syntax	`write_xml_report [-h | -help] [-long_help] <filename>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<filename>	Name of XML Report Database File to which to write
Description	```Writes the current Compilation Report or Simulation Report to the specified XML Report Database File (.xml).```	

continued...

Example Usage	<pre>load_package report set project_name "chiptrip" project_open $project_name ## Create XML Report Database File (.xml) load_report file delete -force report.xml puts [write_xml_report report.xml] unload_report # Close project project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Report written to XML Report Database File: *<string>*. No action is required.
	TCL_ERROR	1	ERROR: Can't write XML Report Database File: *<string>* -- report does not exist. Make sure that an existing report is loaded. Type load_report -h for more information.

3.1.35. ::quartus::sdc

The following table displays information for the **::quartus::sdc** Tcl package:

Tcl Package and Version	::quartus::sdc 1.5
Description	<pre>Synopsys Design Constraint (SDC) format is used to specify the design intent, including the timing and area constraints of the design. The Timing Analyzer only implements the set of SDC commands required to specify the timing constraints of the design. For area constraints, the QSF file should be used. This package implements the SDC Spec Version 1.5 (June 2005). Any command in this package can be specified in a Timing Analyzer SDC file.</pre>
Availability	<pre>This package is loaded by default in the following executable: quartus_sta This package is available for loading in the following executable: quartus_fit</pre>
Tcl Commands	all_clocks (::quartus::sdc) on page 563 all_inputs (::quartus::sdc) on page 563 all_outputs (::quartus::sdc) on page 564 all_registers (::quartus::sdc) on page 564 create_clock (::quartus::sdc) on page 565 create_generated_clock (::quartus::sdc) on page 566 derive_clocks (::quartus::sdc) on page 568 get_cells (::quartus::sdc) on page 568 get_clocks (::quartus::sdc) on page 570 get_nets (::quartus::sdc) on page 571 get_pins (::quartus::sdc) on page 572 get_ports (::quartus::sdc) on page 573 remove_clock_groups (::quartus::sdc) on page 574 remove_clock_latency (::quartus::sdc) on page 574 remove_clock_uncertainty (::quartus::sdc) on page 575 remove_disable_timing (::quartus::sdc) on page 576 remove_input_delay (::quartus::sdc) on page 576 remove_output_delay (::quartus::sdc) on page 577 reset_design (::quartus::sdc) on page 578 set_clock_groups (::quartus::sdc) on page 578 set_clock_latency (::quartus::sdc) on page 579 set_clock_uncertainty (::quartus::sdc) on page 581 set_disable_timing (::quartus::sdc) on page 582 set_false_path (::quartus::sdc) on page 583 set_input_delay (::quartus::sdc) on page 584 set_input_transition (::quartus::sdc) on page 586

continued...

	set_max_delay (::quartus::sdc) on page 587 set_max_time_borrow (::quartus::sdc) on page 588 set_min_delay (::quartus::sdc) on page 589 set_multicycle_path (::quartus::sdc) on page 591 set_output_delay (::quartus::sdc) on page 593

3.1.35.1. all_clocks (::quartus::sdc)

The following table displays information for the `all_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`all_clocks [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns a collection of all previously defined clocks in the design.`		
Example Usage	`project_open chiptrip` `create_timing_netlist` `foreach_in_collection clk [all_clocks] {` `  puts [get_clock_info -name $clk]` `}` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.2. all_inputs (::quartus::sdc)

The following table displays information for the `all_inputs` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`all_inputs [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns a collection of all input ports in the design.`		
Example Usage	`project_open chiptrip` `create_timing_netlist` `foreach_in_collection in [all_inputs] {` `  puts [get_port_info -name $in]` `}` `set_input_delay -clock clock1 2.0 [all_inputs]` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.3. all_outputs (::quartus::sdc)

The following table displays information for the `all_outputs` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`all_outputs [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns a collection of all output ports in the design.`		
Example Usage	`project_open chiptrip` `create_timing_netlist` `foreach_in_collection out [all_outputs] {` `  puts [get_port_info -name $out]` `}` `set_output_delay -clock clock1 2.0 [all_outputs]` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.4. all_registers (::quartus::sdc)

The following table displays information for the `all_registers` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`all_registers [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns a collection of all registers in the design.`		
Example Usage	`project_open chiptrip` `create_timing_netlist` `foreach_in_collection reg [all_registers] {` `  puts [get_register_info -name $reg]` `}` `report_timing -from [all_registers] -to [all_registers]` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.5. create_clock (::quartus::sdc)

The following table displays information for the `create_clock` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`create_clock [-h | -help] [-long_help] [-add] [-name` *<clock_name>* `] -period` *<value>* `[-waveform` *<edge_list>* `] [` *<targets>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add`	Adds clock to a node with an existing clock
	`-name` *<clock_name>*	Clock name of the created clock
	`-period` *<value>*	Speed of the clock in terms of clock period
	`-waveform` *<edge_list>*	List of edge values
	<targets>	List or collection of targets
Description		

```
Defines a clock. If the -name option is not used, the clock name is
the same as the first target in the list or collection. The clock name
is used to refer to the clock in other commands.

The -period option specifies the clock period. It is also possible to
use this option to specify a frequency to define the clock period.
This can be done by using -period option followed by either
<frequency>MHz or "<frequency> MHz". However, this is a Timing
Analyzer-only extension and makes the SDC syntax non-standard.

The -waveform option specifies the rising and falling edges (duty
cycle) of the clock, and is specified as a list of two time values:
the first rising edge and the next falling edge. The rising edge must
be within the range [0, period]. The falling edge must be within one
clock period of the rising edge. The waveform defaults to (0,
period/2).

If a clock with the same name is already assigned to a given target,
the create_clock command will overwrite the existing clock.  If a clock
with a different name exists on the given target, the create_clock
command will be ignored unless the -add option is used.  The -add
option can be used to assign multiple clocks to a pin or port.

If the target of the clock is internal (i.e. not an input port), the
source latency is zero by default.

If a clock is on a path after another clock, then it blocks or
overwrites the previous clock from that point forward.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.  The
values used must follow standard Tcl or Timing Analyzer-extension
substitution rules.  See the help for use_timing_analyzer_style_escaping for
details.
```

Example Usage

```
# Create a simple 10ns with clock with a 60% duty cycle
create_clock -period 10 -waveform {0 6} -name clk [get_ports clk]

# Create a clock with a falling edge at 2ns, rising edge at 8ns,
# falling at 12ns, etc.
create_clock -period 10 -waveform {8 12} -name clk [get_ports clk]

# Assign two clocks to an input port that are switched externally
create_clock -period 10 -name clk100Mhz [get_ports clk]
create_clock -period 6.667 -name clk150Mhz -add [get_ports clk]

# Two ways to use MHz to define clock period (Timing Analyzer only)
create_clock -period 250MHz -name clk250MHz [get_ports clk]
create_clock -period "250 MHz" -name clk250MHz [get_ports clk]
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.6. create_generated_clock (::quartus::sdc)

The following table displays information for the `create_generated_clock` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`create_generated_clock [-h | -help] [-long_help] [-add] [-divide_by <factor>] [-duty_cycle <percent>] [-edge_shift <shift_list>] [-edges <edge_list>] [-invert] [-master_clock <clock>] [-multiply_by <factor>] [-name <clock_name>] [-offset <time>] [-phase <degrees>] [-source <clock_source>] [<targets>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add`	Add clock to existing clock node
	`-divide_by <factor>`	Division factor
	`-duty_cycle <percent>`	Specifies the duty cycle as a percentage of the clock period--accepts floating point values
	`-edge_shift <shift_list>`	List of edge shifts
	`-edges <edge_list>`	List of edge values
	`-invert`	Invert the clock waveform
	`-master_clock <clock>`	Specifies clock of the source node
	`-multiply_by <factor>`	Multiplication factor
	`-name <clock_name>`	Name of generated clock
	`-offset <time>`	Specifies the offset as an absolute time shift
	`-phase <degrees>`	Specifies the phase shift in degrees
	`-source <clock_source>`	Source node for the generated clock
	<targets>	List or collection of targets
Description		

```
Defines an internally generated clock. If -name is not specified,
the clock name is the same as the first target in the list or
collection. The clock name is used to refer to the clock in other
commands.

If a clock with the same name is already assigned to a given
target, the create_generated_clock command overwrites the
existing clock. If a clock with a different name exists on the
given target, the create_generated_clock command is ignored
unless the -add option is used. The -add option can be used to
assign multiple clocks to a pin or port, and is recommended be
used with -master_clock option.

The source of the generated clock, specified by -source, is a
port, pin, register, or net in the design. All waveform
modifications are relative to this point. If more than one clock
feeds the source node, the -master_clock option must be used to
specify which clock to modify.

The source latency of the generated clock is based on the clock
```

continued...

```
network of the generated clock, and not the clock network of the
node specified using -source. This latency is added to any
source latency of the master clock.

If no target is specified, the clock is treated as a virtual clock.
In that case, the source latency of the generated clock will be
equal to the source latency of the master clock, plus any added
latency specified with set_clock_latency.

The -divide_by, -multiply_by, -invert, -duty_cycle, -edges, and
-edge_shift options modify the waveform relative to the waveform
at the source node.

Clock division and multiplication, using -divide_by and
-multiply_by, is performed relative to the first rising
edge. Clock division is based on edges in the master clock
waveform, and scaled if the division is an odd number. Use the
-duty_cycle option to specify the new duty cycle for clock
multiplication. Use the -phase option to specify any phase shift
relative to the new clock period. Use the -offset option to
specify an arbitrary offset or time shift. Use the -invert
option to invert the generated waveform.  The -phase and -duty_cycle
options may be specified as a decimal value (e.g. 22.5) or
as a ratio of two numbers (e.g. 45/2).  The latter form may
improve Timing Analyzer accuracy when detecting relationships between
related clocks.

Clock generation can also be specified with the -edges and
-edge_shift options. The -edges option accepts a list of three
numbers specifying the master clock edges to use for the first
rising edge, the next falling edge, and next rising edge. Edges
of the master clock are labeled according to the first rising
edge (1), next falling edge (2), next rising edge (3), etc. For
example, a basic clock divider can be specified equivalently
with -divide_by 2 or -edges {1 3 5}. The -edge_shift option
accepts a list of three time values, the amount to shift each of
the three edges.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate
type. The values used must follow standard Tcl or
Timing Analyzer-extension substitution rules. See the help for
use_timing_analyzer_style_escaping for details.
```

Example Usage

```
# Create a clock and a divide-by-2 generated clock
create_clock -period 10 [get_ports clk]
create_generated_clock -divide_by 2 -source [get_ports clk] -name clkdiv [get_registers clkdiv]

# An equivalent generated clock
create_generated_clock -edges {1 3 5} -source [get_ports clk] -name clkdiv [get_registers
clkdiv]

# Specify a clock multipler with a 60% duty cycle
create_generated_clock -multiply_by 2 -source [get_ports clk] -duty_cycle 60 [get_pins clkmult|
combout]

# Specify an inverted divide-by-2 clock relative to the output of the source clock
create_generated_clock -divide_by 2 -invert -source [get_ports clk] -name nclkdiv
[get_registers clkdiv]

# Specify a divide-by-2 clock with a 90-degree phase shift
create_generated_clock -divide_by 2 -phase 90 -source [get_ports clk] -name clkdiv
[get_registers clkdiv]

# Create a divide-by-2 generated clock generated off the falling edge of the source clock
create_generated_clock -edges {2 4 6} -source [get_ports clk] -name clkfall_div [get_registers
clkfall_div]

# Assign two clocks to an input port that are switched externally,
# along with an internal clock divider.
create_clock -period 10 -name clk100Mhz [get_ports clk]
create_clock -period 6.667 -name clk150Mhz -add [get_ports clk]
create_generated_clock -divide_by 2 -name clk50Mhz -source [get_ports clk] -master_clock
clk100Mhz -add [get_registers clkdiv]
create_generated_clock -divide_by 2 -name clk75Mhz -source [get_ports clk] -master_clock
clk150Mhz -add [get_registers clkdiv]

# Create a virtual clock, and two generated clocks that derive from it.
# This makes the generated clocks related, so crossings between them are not asynchronous.
create_clock -period 10 -name virtual_base
create_generated_clock -master_clock virtual_base -divide_by 2 [get_ports clka]
create_generated_clock -master_clock virtual_base -divide_by 4 [get_ports clkb]
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.7. derive_clocks (::quartus::sdc)

The following table displays information for the `derive_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`derive_clocks [-h | -help] [-long_help] [-no_black_box_sources] -period <period_value> [-suffix <suffix_string> ] [-waveform <edge_list> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-no_black_box_sources`	Option to disable considering periphery nodes of black box atoms as possible clock sources when deriving clocks	
	`-period <period_value>`	Speed of the default clock in terms of clock period	
	`-suffix <suffix_string>`	Suffix of derived clock names	
	`-waveform <edge_list>`	List of edge values	
Description			

```
Creates a clock on sources of clock pins in the design that do not
already have at least one clock sourcing the clock pin. This command
is equivalent to calling create_clock on each clock source in the
design that does not already have a clock assigned to it.

See the help for create_clock for more information.

Intel does not recommend using this command during final sign-off
analysis of a design. derive_clocks should only be used early in the
design phase when the clocks are not completely known. When possible,
create_clock and create_generated_clock should be used instead.
```

Example Usage

```
# Automatically create a 10ns, 60% duty cycle clock on all
# unconstrained clock sources.
derive_clocks -period 10 -waveform {0 6}

# Append a suffix to all derived clock names using the "-suffix" option.
# All derived clock names will end with "~my_suffix".
derive_clocks -period 10 -suffix "my_suffix"
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.35.8. get_cells (::quartus::sdc)

The following table displays information for the `get_cells` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`get_cells [-h | -help] [-long_help] [-compatibility_mode] [-hierarchical] [-no_duplicates] [-nocase] [-nowarn] [ <filter> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-compatibility_mode`	Use simple Tcl matching
	`-hierarchical`	Specifies use of a hierarchical searching method
	`-no_duplicates`	Do not match duplicated cell names
	`-nocase`	Specifies case insensitive node name matching
	`-nowarn`	Do not issue warning messages about unmatched patterns
	<filter>	Valid destinations (string patterns are matched using Tcl string matching)

Description

```
Returns a collection of cells in the design. All cell names in the
collection match the specified pattern. Wildcards can be used to
select multiple cells at once.

There are three Tcl string matching schemes available with this
command: the default method, the -hierarchical option, and the
-compatibility_mode option.

When you use the default matching scheme, use pipe characters to
separate one hierarchy level from the next. They are treated as
special characters and are taken into account when string matching
with wildcards is performed. When this matching scheme is enabled, the
specified pattern is matched against absolute cell names: the names
that include the entire hierarchical path. A full cell name can
contain multiple pipe characters in it to reflect the hierarchy. All
hierarchy levels in the pattern are matched level by level. Any
included wildcards refer to only one hierarchical level. For example,
"*" and "*|*" produce different collections since they refer to the
highest hierarchical level and second highest hierarchical level
respectively.

When using the -hierarchical matching scheme, pipe characters are
treated as special characters and are taken into account when string
matching with wildcards is performed. This matching scheme forces the
search to proceed recursively down the hierarchy. The specified
pattern is matched against the relative cell names: the immediate
names that do not include any of the hierarchy information. Note that
a short cell name cannot contain pipe characters in it. Any included
wildcards are expanded to match the relative cell names.

The -compatibility_mode matching scheme uses simple Tcl string matching
on full, absolute cell names. Pipe characters are not treated as special
characters when used with wildcards.

The default matching scheme returns cells whose names match the
specified filter and also cells automatically generated by the Quartus
II software from these cells). Use -no_duplicates option to not
include duplicated cells.

The filter for the collection is a Tcl list of wildcards, and must
follow standard Tcl or Timing Analyzer-extension substitution rules. See
help for the use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
# Find a cell called "reg" using case insensitive search
get_cells -nocase reg
# Create a collection of all cells whose names start with "reg"
get_cells reg*
# Create a collection of all cells on the highest hierarachical level
set mycollection [get_cells *]
# Create a collection of all cells in the design
# Output cell names.
foreach_in_collection cell $mycollection {
    puts [get_cell_info -name $cell]
}
set fullcollection [get_cells -hierarchical *]
# Output cell IDs and names.
foreach_in_collection cell $fullcollection {
    puts -nonewline $cell
    puts -nonewline ": "
    puts [get_cell_info -name $cell]
}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.9. get_clocks (::quartus::sdc)

The following table displays information for the `get_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`get_clocks [-h | -help] [-long_help] [-include_generated_clocks] [-nocase] [-nowarn] [-of_objects <object_collection> ] [ <filter> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-include_generated_clocks`	Includes generated clocks derived from the matched clocks	
	`-nocase`	Specifies the matching of node names to be case-insensitive	
	`-nowarn`	Do not issue warning messages when querying for clocks	
	`-of_objects <object_collection>`	Returns all clocks that target (defined on) or drive (determine data frequency of) the nodes in the collection.	
	<filter>	Valid destinations (string patterns are matched using Tcl string matching)	
Description			
Example Usage			
Return Value	**Code Name**	**Code**	**String Return**

Description

```
Returns a collection of previously defined clocks in the design. Use a clock
collection as the -from/to argument of a command (such as set_multicycle_path)
to refer to all nodes driven by the clocks in the collection.

# The following multicycle constraint applies to all paths ending at registers
# driven by clk
set_multicycle_path -to [get_clocks clk] 2

If a filter, which is a Tcl list of wildcards and must follow
standard Tcl or Timing Analyzer-extension substitution rules, is specified, then
get_clocks returns all previously defined clocks whose names match the filter.
See the help for use_timing_analyzer_style_escaping for filter rules.

If you use the -of_objects option, you must provide a collection of registers, ports,
pins, or cells. The get_clocks command returns a collection of all the previously defined
clocks that target these nodes, or if these nodes are not clock targets, all the previously
defined clocks that drive these nodes. You cannot use the -of_objects option with the clock
name filter. Refer to the long help for examples of using the -of_objects option.

Tip: the get_clocks command can be used as part of SDC commands, as well as for reporting.
When it is used as part of SDC commands and includes the -include_generated_clocks option,
the Timing Analyzer needs to analyze the design and may issue warnings about missing clocks.
The missing clocks might still be created later on in the SDC file and are not
necessarily indicative of a problem. Use the -nowarn option in this case to suppress
those warnings. When you use the -nowarn option, pass a node collection generated by another
get_*
        collection command as the filter. This way, warnings directly related to the filter
resolution are
        still posted.
```

Example Usage

```
# get clocks that begin with 'c' or 'C', and print out their names and periods:
set clocks [get_clocks c* -nocase]
foreach_in_collection clk $clocks {
    set name [get_clock_info -name $clk]
    set period [get_clock_info -period $clk]
    puts "$name: $period"
}

# getting the clock that targets a port, and its generated clock:
create_clock -name my_clock -period 10.000 [get_ports CLK_100]
create_generated_clock -name my_gen_clock -divide_by 2 -source [get_ports CLK_100]
[get_registers clk_div_reg]
get_clocks -nowarn -of_objects [get_ports CLK_100] -include_generated_clocks

# display the name of the clocks that drive registers beginning with 'reg_':
foreach_in_collection clk_id [get_clocks -nowarn -of_objects [get_registers reg_*]] {
    puts [get_clock_info -name $clk_id]
}
```

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Use the -clock_networks_only option only with the -of_objects option.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.10. get_nets (::quartus::sdc)

The following table displays information for the `get_nets` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`get_nets [-h | -help] [-long_help] [-no_duplicates] [-nocase] [-nowarn] [ <filter> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-no_duplicates`	Do not match duplicated net names	
	`-nocase`	Specifies case-insensitive node name matching	
	`-nowarn`	Do not issue warning messages about unmatched patterns	
	<filter>	Valid destinations (string patterns are matched using Tcl string matching)	
Description			
Example Usage			
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

```
Returns a collection of nets in the design. All net names in the
collection match the specified pattern. Wildcards can be used to
select multiple nets at once.

The default matching scheme returns nets whose names match the
specified filter and nets that are automatically generated by the
Quartus Prime software from these nets. Use the -no_duplicates option to
exclude duplicated nets.

The filter for the collection is a Tcl list of wildcards, and must
follow standard Tcl or Timing Analyzer-extension substitution rules.  See
help for the use_timing_analyzer_style_escaping command for details.
```

```
# Find a net called "reg" using case insensitive search
get_nets -nocase reg
# Create a collection of all nets whose names start with "reg"
get_nets reg*
# Create a collection of all nets in the design
set mycollection [get_nets *]
# Output net names.
foreach_in_collection net $mycollection {
    puts [get_net_info -name $net]
}
```

3.1.35.11. get_pins (::quartus::sdc)

The following table displays information for the `get_pins` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`get_pins [-h | -help] [-long_help] [-compatibility_mode] [-hierarchical] [-no_duplicates] [-nocase] [-nowarn] [ <filter> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-compatibility_mode`	Use simple Tcl matching
	`-hierarchical`	Specifies use of a hierarchical searching method
	`-no_duplicates`	Do not match duplicated pin names
	`-nocase`	Specifies case-insensitive node name matching
	`-nowarn`	Do not issue warning messages about unmatched patterns
	<filter>	Valid destinations (string patterns are matched using Tcl string matching)
Description		

```
Returns a collection of pins in the design. All pin names in the
collection match the specified pattern. Wildcards can be used to
select multiple pins at once.

There are three Tcl string matching schemes available with this
command: the default method, the -hierarchical option, and the
-compatibility_mode option.

By default, pipe characters are used to separate one hierarchy level
from the next. They are treated as special characters and are taken
into account when string matching with wildcards is performed. When
the default matching scheme is enabled, the specified pattern is
matched against absolute pin names: the names that include the entire
hierarchical path. All hierarchy levels in the pattern are matched
level by level. Pin names of the form <absolute full cell name>|<pin
suffix> are used for matching. Note that a full cell name can contain
multiple pipe characters in it to reflect the hierarchy. Any included
wildcards refer to only one hierarchical level. For example, "*|*" and
"*|*|*" produce different collections since they refer to the highest
hierarchical level and second highest hierarchical level respectively.

When using the -hierarchical matching scheme, pipe characters are
treated as special characters and are taken into account when string
matching with wildcards is performed. This matching scheme forces the
search to proceed recursively through the hierarchy. The specified
pattern is matched against the relative pin names: the immediate names
that do not include any of the hierarchy information.  Pin names of
the form <relative short cell name>|<pin suffix> are used for
matching. Note that a short cell name cannot contain pipe
characters. Any included wildcards are expanded to match the relative
pin names.  For example, "*" and "*|*" match exactly the same pins
since the former is expanded into the latter.

The -compatibility_mode matching scheme uses simple Tcl string matching
on full, absolute cell names. Pipe characters are not treated as special
characters when used with wildcards.

The default matching scheme returns not only pins whose names match
the specified filter, but also pins duplicated from these pins (refers
to pins are automatically generated by Quartus from the pins).  Use
-no_duplicates option to not include duplicated pins.

The filter for the collection is a Tcl list of wildcards, and must
follow standard Tcl or Timing Analyzer-extension substitution rules. See
help for the use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
# Get regout pin of "reg" cell
get_pins -nocase reg|regout
# Create a collection of all pins of "reg" cell
get_pins reg|*
# Create a collection of all pins on the highest hierarachical level
```

continued...

```
set mycollection [get_pins *]
# Output pin names.
foreach_in_collection pin $mycollection {
    puts [get_pin_info -name $pin]
}
# Create a collection of all pins in the design
set fullcollection [get_pins -hierarchical *]
# Output pin IDs and names.
foreach_in_collection pin $fullcollection {
    puts -nonewline $pin
    puts -nonewline ": "
    puts [get_pin_info -name $pin]
}
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.12. get_ports (::quartus::sdc)

The following table displays information for the `get_ports` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`get_ports [-h | -help] [-long_help] [-nocase] [-nowarn] [ <filter> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-nocase`	Specifies case-insensitive node name matching
	`-nowarn`	Do not issue warning messages about unmatched patterns
	<filter>	Valid destinations (string patterns are matched using Tcl string matching)

Description

```
Returns a collection of ports (design inputs and outputs) in the
design.

The filter for the collection is a Tcl list of wildcards, and must
follow standard Tcl or Timing Analyzer-extension substitution rules. See
help for the use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
project_open chiptrip
create_timing_netlist

# Get all ports starting with "In".
set ports [get_ports In*]
foreach_in_collection port $ports {
    puts [get_port_info -name $port]
}

delete_timing_netlist
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.13. remove_clock_groups (::quartus::sdc)

The following table displays information for the `remove_clock_groups` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`remove_clock_groups [-h | -help] [-long_help] -all`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-all`	Specify remove all clock group settings	
Description	`Remove all clock group assignments. This command removes any clock groups that have been previously set. There is no way to remove specific groups.`		
Example Usage	`project_open top` `create_timing_netlist` `create_clock -period 10.000 -name clkA [get_ports sysclk[0]]` `create_clock -period 10.000 -name clkB [get_ports sysclk[1]]` `# Set clkA and clkB to be mutually exclusive clocks.` `set_clock_groups -exclusive -group {clkA} -group {clkB}` `set_clock_groups -exclusive -group {clkC} -group {clkD}` `# Remove clock groups A, B, C, and D. Result is that there` `# are no longer any mutually exclusive clocks.` `remove_clock_groups -all`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.35.14. remove_clock_latency (::quartus::sdc)

The following table displays information for the `remove_clock_latency` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`remove_clock_latency [-h | -help] [-long_help] -source` *<targets>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-source`	Specifies the source clock latency
	<targets>	Valid destinations (string patterns are matched using Tcl string matching)
Description	`Removes clock latency for a given clock or clock target.` `There are two types of latency: network and source. Network latency is the clock network delay between the clock and register clock pins. Source latency is the clock network delay between the clock and its source (e.g., a system clock or a base clock of a generated clock).` `The Timing Analyzer automatically computes network latencies for all register and generated clocks. Overriding clock network latencies is not supported by the Timing Analyzer. Therefore, the -source option must always be specified. Remove_clock_latency requires this option as well.` `You can apply clock latency to a clock, which affects all targets of`	

continued...

	the clock, or to a specific clock target. Therefore, you can remove clock latency from a collection of clocks, or from a collection of target nodes. remove_clock_latency removes all latencies from a clock or node, so removing a node's clock latency with respect to a particular clock, or removing only latencies with particular conditions is not supported. The value of the targets is either a collection or a Tcl list of wildcards used to create a collection of the appropriate type. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See the help for use_timing_analyzer_style_escaping for details.
Example Usage	`create_clock -name SYSCLK -period 10.000 [get_ports inclk]` `create_generated_clock -name OUTCLK -divide_by 1 -source [get_ports inclk] [get_ports outclk]` `create_generated_clock -name FDBKCLK -divide_by 1 -source [get_ports outclk] [get_ports fdbkclk]` `# Apply a simple 2.000 ns source latency to the system clock.` `set_clock_latency -source 2.000 [get_clocks SYSCLK]` `# Specify feedback clock latencies between output port outclk` `# and the output port fdbkclk.` `set_clock_latency -source -late -rise 0.800 [get_clocks FDBKCLK]` `set_clock_latency -source -late -fall 0.750 [get_clocks FDBKCLK]` `set_clock_latency -source -early -rise 0.500 [get_clocks FDBKCLK]` `set_clock_latency -source -early -fall 0.460 [get_clocks FDBKCLK]` `# Remove all clock latency from FDBKCLK` `remove_clock_latency -source [get_clocks FDBKCLK]`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.15. remove_clock_uncertainty (::quartus::sdc)

The following table displays information for the `remove_clock_uncertainty` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`remove_clock_uncertainty [-h | -help] [-long_help] -from` *<from_clock>* `-to` *<to_clock>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-from` *<from_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-to` *<to_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
Description		

```
Removes clock uncertainty from a collection of clocks to a collection
of clocks. The source and destination clocks can be any arbitrary
collection of clocks. This command removes all uncertainty between two
clocks. If there does not exist uncertainty between two clocks
specified in remove_clock_uncertainty, the command does nothing for
those two clocks but continues to attempt to remove uncertainty
between other clocks specified.

The values of the -from and -to options are either collections or a
Tcl list of wildcards used to create collections of appropriate types.
The values used must follow standard Tcl or Timing Analyzer-extension
substitution rules.  See the help for use_timing_analyzer_style_escaping for
details.
```

continued...

Example Usage	`set_clock_uncertainty -setup -rise_from {clk1 clk2} -fall_to {clk3 clk4} 200ps` `set_clock_uncertainty -from {clk5 clk6} -to {clk7 clk8} 300ps` `remove_clock_uncertainty -from {clk3 clk5} -to {clk4 clk7}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.35.16. remove_disable_timing (::quartus::sdc)

The following table displays information for the `remove_disable_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`remove_disable_timing [-h | -help] [-long_help] [-from` *<name>* `] [-to` *<name>* `]` *<cells>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-from` *<name>*	Valid source pin suffix
	`-to` *<name>*	Valid destination pin suffix
	<cells>	List of cells
Description	`Adds a previously disabled edge (arc) back to a given cell(s). If no -from/-to value is specified, the missing value is substituted by a "*".` `The values of the -from and -to are valid pin suffixes. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See the help for use_timing_analyzer_style_escaping for details.`	
Example Usage	`remove_disable_timing -from datain -to combout A|B` `remove_disable_timing -from carryin *`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.17. remove_input_delay (::quartus::sdc)

The following table displays information for the `remove_input_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`remove_input_delay [-h | -help] [-long_help] [-blackbox]` *<targets>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-blackbox`	Removes an input delay that was assigned to a partition boundary port.
	<targets>	Collection or list of input ports

continued...

Description	<pre>Removes input delay from a port. For each input port specified, removes all input delays for that port. This means that rise, fall, max, and min delays for each clock and reference pin on the input port are all removed. The value of the targets is either a collection or a Tcl list of wildcards used to create a collection of the appropriate type. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See help for the use_timing_analyzer_style_escaping command for details.</pre>		
Example Usage	<pre># Simple input delay with the same value for min/max and rise/fall set_input_delay -clock clk 1.5 [get_ports {in1 in2}] set_input_delay -clock clk2 1.5 [get_ports {in1 in2}] set_input_delay -clock clk 1.6 [get_ports {in3 in4}] # Remove input delay on ports in1 and in4, # for all flags and reference ports and flags remove_input_delay [get_ports {in1 in4}]</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Incorrect collection type. Expected a collection of type <*string*>.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.18. remove_output_delay (::quartus::sdc)

The following table displays information for the `remove_output_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`remove_output_delay [-h | -help] [-long_help] [-blackbox]` <*targets*>		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-blackbox`		Removes an output delay that was assigned to a partition boundary port.
	<*targets*>		Collection or list of output ports
Description	<pre>Removes output delay from a port. For each output port specified, removes all output delays for that port. Rise, fall, max, and min delays for each clock and reference pin on the output port are all removed. The value of the targets is either a collection or a Tcl list of wildcards used to create a collection of the appropriate type. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See help for the use_timing_analyzer_style_escaping command for details.</pre>		
Example Usage	<pre># Simple output delay with the same value for min/max and rise/fall set_output_delay -clock clk 1.5 [get_ports {out1 out2}] set_output_delay -clock clk2 1.5 [get_ports {out1 out2}] set_output_delay -clock clk 1.6 [get_ports {out3 out4}] # Remove input delay on ports out1 and out4, # for all flags and reference ports and flags remove_output_delay [get_ports {out1 out4}]</pre>		
Return Value	**Code Name**	**Code**	**String Return**

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Incorrect collection type. Expected a collection of type <*string*>.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.19. reset_design (::quartus::sdc)

The following table displays information for the `reset_design` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`reset_design [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Removes all assignments from the design. This includes clocks, generated clocks, derived clocks, input delays, output delays, clock latency, clock uncertainty, clock groups, false paths, multicycle paths, min delays, and max delays. After reset_design is called, the design should be in the same state as it would be if create_timing_netlist was just called.`		
Example Usage	`# Constrain design` `create_clock -name clk -period 4.000 -waveform { 0.000 2.000 } [get_ports clk]` `set_input_delay -clock clk2 1.5 [get_ports in*]` `set_output_delay -clock clk 1.6 [get_ports out*]` `set_false_path -from [get_keepers in] -through [get_nets r1] -to [get_keepers out]` `# Reset the design to the state that it was in before any constraints were entered` `reset_design`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.20. set_clock_groups (::quartus::sdc)

The following table displays information for the `set_clock_groups` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_clock_groups [-h | -help] [-long_help] [-asynchronous] [-exclusive] -group <names> [-logically_exclusive] [-physically_exclusive]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-asynchronous`	Specify mutually exclusive clocks (such as groups of primary clocks)
	`-exclusive`	Specify mutually exclusive clocks (an alias for the -logically_exclusive option). Exists for backwards compatibility.
	`-group <names>`	Valid destinations (string patterns are matched using Tcl string matching)

continued...

	-logically_exclusive	Specify logically exclusive clocks (meaning they are not actively used at the same time)
	-physically_exclusive	Specify physically exclusive clocks (meaning they are not physically present at the same time)
Description		

```
Clock groups provide a quick and convenient way to specify which
clocks are not related.  Asynchronous clocks are those that are
completely unrelated (e.g., have different ideal clock sources).
Logically exclusive clocks are not actively used in the design at
the same time (e.g., multiplexed clocks), but the clock signals may
physically exist on-chip at the same time and therefore may still
influence each other through crosstalk effects.  Physically exclusive
clocks, in contrast, cannot be physically present in the device at
the same time (e.g., multiple clocks defined on the same clock pin).

The Timing Analyzer does not currently analyze crosstalk explicitly.
Instead, the timing models use extra guard bands to account for any
potential crosstalk-induced delays. As a result, the Timing Analyzer
currently treats asynchronous, logically_exclusive, and physically_exclusive
clock groups the same.  However, different parts of the Timing Analyzer may
treat asynchronous and exclusive groups differently. Any commands that are
affected by clock groups will say so in their help text.  But, no
distinction is made between logically and physically exclusive clock groups,
since the only difference between them is how they affect crosstalk.

The result of set_clock_groups is that all clocks in any group are cut
from all clocks in every other group. The use of a single -group option
tells the Timing Analyzer to cut this group of clocks from all other
clocks in the design, including clocks that are created in the future.
This command is similar to calling set_false_path from each clock in every
group to each clock in every other group and vice versa, making
set_clock_groups easier to specify for cutting clock domains.  However,
cutting clocks with set_clock_groups also affects the results of some other
commands.  Any commands that are affected by clock groups will say so in
their help text.
```

Example Usage

```
project_open top
create_timing_netlist
create_clock -period 10.000 -name clkA [get_ports sysclk[0]]
create_clock -period 10.000 -name clkB [get_ports sysclk[1]]

# Set clkA and clkB to be mutually exclusive clocks.
set_clock_groups -logically_exclusive -group {clkA} -group {clkB}

# The previous line is equivalent to the following two commands.
set_false_path -from [get_clocks clkA] -to [get_clocks clkB]
set_false_path -from [get_clocks clkB] -to [get_clocks clkA]
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.35.21. set_clock_latency (::quartus::sdc)

The following table displays information for the `set_clock_latency` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_clock_latency [-h | -help] [-long_help] [-clock <clock_list> ] [-early] [-fall] [-late] [-rise] -source <delay> <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clock <clock_list>`	Valid clock destinations (string patterns are matched using Tcl string matching)
	`-early`	Specifies the early clock latency
	`-fall`	Specifies the falling transition clock latency

continued...

	`-late`	Specifies the late clock latency
	`-rise`	Specifies the rising transition clock latency
	`-source`	Specifies the source clock latency
	<delay>	Latency delay value
	<targets>	Valid destinations (string patterns are matched using Tcl string matching)

Description

```
Specifies clock latency for a given clock or clock
target.

There are two types of latency: network and
source. Network latency is the clock network delay
between the clock and register clock pins. Source
latency is the clock network delay between the clock
and its source (e.g., the system clock or base clock
of a generated clock).

The Timing Analyzer automatically computes
network latencies for all register and generated
clocks. Overriding clock network latencies is not
supported by the Timing Analyzer. Therefore, the
-source option must always be specified.

You can apply clock latency to a clock, which affects
all targets of the clock, or to a specific clock
target. If you specify a specific clock target that is
driven by more than one clock, use the -clock option
to specify which clock to use.Latencies assigned to a
clock target override any latencies assigned to a
clock.

Different clock latencies can be specified for early
(-early) and late (-late) latencies, as well as for
rising edges (-rise) and falling edges (-fall). If
only some combinations are specified, the other
combinations are used by default. For example, if only
a -rise -early latency and a -fall -early latency are
specified, then the -rise -late latency is assumed to
be the same as the -rise -early latency and the -fall
-late latency is assumed to be the same as the -fall
-early latency. If neither -rise nor -fall are used or
neither -early nor -late are used, then the latency
applies to both conditions.

Source latency can also be assigned to generated
clocks. This may be useful for specifying board level
delays from a clock output port to a clock input port
when the clock input port is acting as a feedback
clock.

The value of the targets is either a collection or a
Tcl list of wildcards used to create a collection of
the appropriate type. The values used must follow
standard Tcl or Timing Analyzer-extension substitution
rules. See help for the use_timing_analyzer_style_escaping
command for details.
```

Example Usage

```
create_clock -name SYSCLK -period 10.000 [get_ports inclk]
create_generated_clock -name OUTCLK -divide_by 1 -source [get_ports inclk] [get_ports outclk]
create_generated_clock -name FDBKCLK -divide_by 1 -source [get_ports outclk] [get_ports
fdbkclk]

# Apply a simple 2.000 ns source latency to the system clock.
set_clock_latency -source 2.000 [get_clocks SYSCLK]

# Specify feedback clock latencies between output port outclk
# and the input port fdbkclk.
set_clock_latency -source -late -rise 0.800 [get_clocks FDBKCLK]
set_clock_latency -source -late -fall 0.750 [get_clocks FDBKCLK]
set_clock_latency -source -early -rise 0.500 [get_clocks FDBKCLK]
set_clock_latency -source -early -fall 0.460 [get_clocks FDBKCLK]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.22. set_clock_uncertainty (::quartus::sdc)

The following table displays information for the `set_clock_uncertainty` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_clock_uncertainty [-h | -help] [-long_help] [-add] [-enable_same_physical_edge] [-fall_from` *<fall_from_clock>* `] [-fall_to` *<fall_to_clock>* `] [-from` *<from_clock>* `] [-hold] [-rise_from` *<rise_from_clock>* `] [-rise_to` *<rise_to_clock>* `] [-setup] [-to` *<to_clock>* `] <uncertainty>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add`	Specifies that this assignment is an addition to the clock uncertainty derived by derive_clock_uncertainty call
	`-enable_same_physical_edge`	Enable setting uncertainty value for same physical clock edge
	`-fall_from` *<fall_from_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-fall_to` *<fall_to_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-from` *<from_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-hold`	Only apply the uncertainty value to hold and removal checks
	`-rise_from` *<rise_from_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-rise_to` *<rise_to_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	`-setup`	Only apply the uncertainty value to setup and recovery checks
	`-to` *<to_clock>*	Valid destinations (string patterns are matched using Tcl string matching)
	<uncertainty>	Uncertainty
Description		

```
Specifies clock uncertainty or skew for clocks for
clock-to-clock transfers. You can specify uncertainty
separately for setup and hold, and you can specify
separate rising and falling clock transitions. If you omit
to specify -setup or -hold, the uncertainty value will be
applied to both analysis types. Similarly, if you omit to
specify rising or falling clock transitions, the uncertainty
value will be applied to both transitions.
The setup uncertainty is subtracted from the data required
time for each applicable path, and the hold
uncertainty is added to the data required time for
each applicable path.

Intel Quartus Prime software computes clock uncertainty for every clock
transfer. For particular transfers, you can use the set_clock_uncertainty
assignment to override the automatically derived value, or you can
specify the -add option to add to the automatically derived value.
Note that the -add option is only relative to the automatically derived
value. If multiple set_clock_uncertainty assignments apply to the same
clock transfer, the later value overrides the earlier ones,
regardless of whether the -add option was used.

Note: The Timing Analyzer does not apply
clock uncertainty to transfers involving the same
physical launch and latch edge (that is, the latch and
```

continued...

	launch edges are the same edge of a clock source and occur at the same time) by default. Such transfers typically occur in hold analysis, but may also occur in setup analysis with a multicycle value of 0. You can use the -enable_same_physical_edge option to override this behavior. The values for -from, -to, and similar options are either collections or a Tcl list of wildcards used to create collections of appropriate types. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See the help for use_timing_analyzer_style_escaping for details.		
Example Usage	`set_clock_uncertainty -setup -rise_from clk1 -fall_to clk2 200ps`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.35.23. set_disable_timing (::quartus::sdc)

The following table displays information for the `set_disable_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562		
Syntax	`set_disable_timing [-h | -help] [-long_help] [-from` *<name>* `] [-to` *<name>* `]` *<cells>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-from` *<name>*		Valid source pin suffix
	`-to` *<name>*		Valid destination pin suffix
	<cells>		List of cells
Description	Disables a timing edge (arc) from inside a given cell or cells. Disabling a timing edge prevents timing analysis through that edge. If either -from or -to (or both) are unspecified, the missing value or values are replaced by a "*" character. The values of the -from and -to are valid pin suffixes. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See the help for use_timing_analyzer_style_escaping for details.		
Example Usage	`set_disable_timing -from datain -to combout A|B` `set_disable_timing -from carryin *`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.24. set_false_path (::quartus::sdc)

The following table displays information for the `set_false_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_false_path [-h | -help] [-long_help] [-fall_from <names> ] [-fall_to <names> ] [-from <names> ] [-hold] [-latency_insensitive] [-no_synchronizer] [-rise_from <names> ] [-rise_to <names> ] [-setup] [-through <names> ] [-to <names> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-hold`	Specifies the false_path value (applies only to clock hold or removal checks)
	`-latency_insensitive`	Mark this false path as one that should still be optimized
	`-no_synchronizer`	Prevent this false path from triggering a synchronizer
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Specifies the false_path value (applies only to clock setup or recovery checks)
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
Description		

```
Specifies a false-path exception, removing (or cutting) paths from
timing analysis.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design.  If the -from or -to values are not
specified, the collection is converted automatically into [get_keepers
*].  It is worth noting that if the counterpart of the unspecified
collection is a clock collection, it is more efficient to explicitly
specify this collection as a clock collection only if the clock
collection also generates the desired assignment.

Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or ports
driven by the -to clock. Applying exceptions between a pair of clocks
is more efficient than for specific node to node or node to clock
paths.

If the -latency_insensitive flag is set, the Fitter will be allowed to
freely insert additional pipelining stages on any paths that are cut by
the exception. These pipelined stages will be retimed to improve performance,
but the timing requirements of the paths will still be ignored, just like
for ordinary false paths. Without this flag, the Fitter will not perform any
optimizations on the cut paths.

If pin names or collections are used, the -from value must be a clock
pin and the -to value must be any non-clock input pin to a
register. Assignments from clock pins or to and from cells applies to
all registers in the cell or driven by the clock pin.
```

continued...

```
The -through values are collections of pins or nets in the design. An
exception applied through a node in the design applies only to paths
through the specified node.
The Timing Analyzer allows you to specify the -through argument multiple times to
describe paths that go through multiple points.
For instance, users can select all paths that go through node X, and then go through node Y.
This helps you narrow down and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
-from destination nodes. The rise or fall value of the option
indicates that the "from" nodes are driven by the rising or falling
edge of the clock that feeds this node, taking into consideration any
logical inversions along the clock path.  The -from option is the
combination of both rising and falling "from" nodes.  If the "from"
collection is a clock collection, the assignment applies to those
nodes that are driven by the respective rising or falling clock edge.

The -rise_to and -fall_to options behave similarly to the "from"
options described previously. These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value, taking into consideration any logical
inversions that are along the clock path.

The -setup and -hold options allow the false path to only be applied
to the corresponding setup/recovery or hold/removal checks. The
default if neither value is specified is to apply the false path to
both -setup and -hold.

The values of the -from, -to, -through, and other similar options are
either collections or a Tcl list of wildcards used to create
collections of appropriate types. The values used must follow standard
Tcl or Timing Analyzer-extension substitution rules.  See help for the
use_timing_analyzer_style_escaping command for details.

See help for the set_clock_groups command for information.
```

Example Usage

```
# Set a false-path between two unrelated clocks
# See also set_clock_groups
set_false_path -from [get_clocks clkA] -to [get_clocks clkB]

# Set a false-path for a specific path
set_false_path -from [get_pins regA|clk] -to [get_pins regB|aclr]

# Set a false-path from a node to a falling clock
set_false_path -from [get_pins regA|clk] -fall_to [get_clocks clkB]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.35.25. set_input_delay (::quartus::sdc)

The following table displays information for the `set_input_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_input_delay [-h | -help] [-long_help] [-add_delay] [-blackbox] -clock <name> [-clock_fall] [-fall] [-max] [-min] [-reference_pin <name> ] [-rise] [-source_latency_included] <delay> <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add_delay`	Create additional delay constraint instead of overriding previous constraints
	`-blackbox`	Create an assignment for a partition boundary port causing it to be treated as a port
	`-clock <name>`	Clock name
	`-clock_fall`	Specifies that input delay is relative to the falling edge of the clock

continued...

	`-fall`	Specifies the falling input delay at the port
	`-max`	Applies value as maximum data arrival time
	`-min`	Applies value as minimum data arrival time
	`-reference_pin` *<name>*	Specifies a pin or port in the design to which the input delay is relative
	`-rise`	Specifies the rising input delay at the port
	`-source_latency_included`	Specifies that input delay includes added source latency
	<delay>	Time value
	<targets>	List of input port type objects

Description

```
Specifies the data arrival times at the specified input ports relative
the clock specified by the -clock option. The clock must refer to a
clock name in the design.

Input delays can be specified relative to the rising edge (default) or
falling edge (-clock_fall) of the clock.

Input delays can be specified relative to a pin or a port (-reference_pin)
in the clock network. Clock arrival times to the reference pin or port
are added to data arrival times.

If no -reference_pin is specified, if the input delay is specified
relative to a generated clock with a single target, the clock arrival
times to the generated clock are added to the data arrival time. If the
generated clock has multiple targets, the worst case arrival time to
those targets will be used.

Input delays can already include clock source latency. By default the
clock source latency of the related clock is added to the input delay
value, but when the -source_latency_included option is specified, the
clock source latency is not added because it was factored into the
input delay value.

The maximum input delay (-max) is used for clock setup checks or
recovery checks and the minimum input delay (-min) is used for clock
hold checks or removal checks. If only -min or -max (or neither) is
specified for a given port, the same value is used for both.

Separate rising (-rise) and falling (-fall) arrival times at the port
can be specified. If only one of -rise and -fall are specified for a
given port, the same value is used for both.

By default, set_input_delay removes any other input delays to the port
except for those with the same -clock, -clock_fall, and -reference_pin
combination. Multiple input delays relative to different clocks, clock
edges, or reference pins can be specified using the -add_delay option.
Future input delays relative to different clocks, clock edges, or
reference pins that do not specify the -add_delay option will still
remove previous input delays that specified the -add_delay option.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.  The
values used must follow standard Tcl or Timing Analyzer-extension
substitution rules. See help for the use_timing_analyzer_style_escaping
command for details.
```

Example Usage

```
# Simple input delay with the same value for min/max and rise/fall:
# 1) set on ports with names of the form myin*
set_input_delay -clock clk 1.5 [get_ports myin*]
# 2) set on all input ports
set_input_delay -clock clk 1.5 [all_inputs]

# Input delay with respect to the falling edge of clock
set_input_delay -clock clk -clock_fall 1.5 [get_ports myin*]

# Input delays for different min/max and rise/fall combinations
set_input_delay -clock clk -max -rise 1.4 [get_ports myin*]
set_input_delay -clock clk -max -fall 1.5 [get_ports myin*]
set_input_delay -clock clk -min -rise 0.7 [get_ports myin*]
set_input_delay -clock clk -min -fall 0.8 [get_ports myin*]

# Adding multiple input delays with respect to more than one clock
set_input_delay -clock clkA -min 1.2 [get_ports myin*]
set_input_delay -clock clkA -max 1.8 [get_ports myin*]
```

continued...

```
set_input_delay -clock clkA -clock_fall 1.6 [get_ports myin*] -add_delay
set_input_delay -clock clkB -min 2.1 [get_ports myin*] -add_delay
set_input_delay -clock clkB -max 2.5 [get_ports myin*] -add_delay

# This is a common mistake where input delays are accidentally removed
set_input_delay -clock clkA -min 0.2 [get_ports myout*]
set_input_delay -clock clkB -min 1.1 [get_ports myout*] -add_delay
# The following removes the clkB entry. You need to always use -add_delay
#     when more than one clock, clock_fall or reference_pin exists
set_input_delay -clock clkA -max 0.8 [get_ports myout*]
# The following removes the clkA entry. You need to always use -add_delay
#     when more than one clock, clock_fall or reference_pin exists
set_input_delay -clock clkB -max 1.5 [get_ports myout*]

# Specifying an input delay relative to an external clock output port
set_input_delay -clock clk -reference_pin [get_ports clkout] 0.8 [get_ports myin*]

# Specifying an input delay relative to the clock pin of a register
set_input_delay -clock clk -reference_pin [get_pins regA|clk] 0.8 [get_ports myin*]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Incorrect collection type. Expected a collection of type <*string*>.
TCL_ERROR	1	ERROR: Options -<*string*> and -<*string*> are mutually exclusive. Specify only one of the two options.
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.26. set_input_transition (::quartus::sdc)

The following table displays information for the `set_input_transition` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_input_transition [-h | -help] [-long_help] [-clock` <*name*> `] [-clock_fall] [-fall] [-max] [-min] [-rise]` <*transition*> <*ports*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-clock` <*name*>	Clock name
	`-clock_fall`	Specifies that input delay is relative to the falling edge of the clock
	`-fall`	Specifies the falling output delay at the port
	`-max`	Applies value as maximum data required time
	`-min`	Applies value as minimum data required time
	`-rise`	Specifies the rising output delay at the port
	<*transition*>	Time value
	<*ports*>	Collection or list of input or bidir ports
Description		

```
This constraint does not affect calculations performed by
the Timing Analyzer. It only affects PrimeTime analysis.
If you set this constraint in the Timing Analyzer the constraint is
written out to the SDC file when you call write_sdc.
```

continued...

Example Usage	`set_input_transition 50 [all_inputs]`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Incorrect collection type. Expected a collection of type <*string*>.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.35.27. set_max_delay (::quartus::sdc)

The following table displays information for the `set_max_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_max_delay [-h | -help] [-long_help] [-fall_from <names> ] [-fall_to <names> ] [-from <names> ] [-no_synchronizer] [-rise_from <names> ] [-rise_to <names> ] [-through <names> ] [-to <names> ] <value>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-no_synchronizer`	Prevent this max delay from triggering a synchronizer
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	<*value*>	Time Value
Description		

```
Specifies a maximum delay exception for a given path.

The maximum delay is similar to changing the setup relationship
(latching clock edge - launching clock edge), except that it can be
applied to input or output ports without input or output delays
assigned to them.  Maximum delays are always relative to any clock
network delays (if the source or destination is a register) or any
input or output delays (if the source or destination is a port).
Therefore, input delays and clock latencies are added to the data
arrival times. Clock latencies also added to data required times and
output delays are subtracted from data required times.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design.  If the -from or -to values are not
specified, the collection is converted automatically into [get_keepers
*].  It is worth noting that if the counterpart to the unspecified
collection is a clock collection, it is more efficient to explicitly
specify this collection as a clock collection but only if the clock
collection also generates the desired assignment.
```

continued...

```
Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or ports
driven by the -to clock. Applying exceptions between a pair of clocks
is more efficient than for specific node to node or node to clock
paths.

If pin names or collections are used, the -from value must be a clock
pin and the -to value must be any non-clock input pin to a register.
Assignments from clock pins or to and from cells applies to all
registers in the cell or driven by the clock pin.

The -through values are collections of pins or nets in the design.  An
exception applied through a node in the design applies only to paths
through the specified node.
The Timing Analyzer allows you to specify the -through argument multiple times to
describe paths that go through multiple points.
For instance, users can select all paths that go through node X, and then go through node Y.
This helps you narrow down and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
-from destination nodes.  The rise or fall value of the option
indicates that the "from" nodes are driven by the rising or falling
edge of the clock that feeds this node taking into consideration any
logical inversions along the clock path.  The "-from" option is the
combination of both rising and falling "from" nodes.  If the "from"
collection is a clock collection, the assignment applies to those
nodes that are driven by the respective rising or falling clock edge.

The -rise_to and -fall_to options behave similarly to the "from1"
options described previously.  These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value taking into consideration any logical
inversions that are along the clock path.

The values of the -from, -to, -through, and other similar options are
either collections or a Tcl list of wildcards used to create
collections of appropriate types. The values used must follow standard
Tcl or Timing Analyzer-extension substitution rules. See help for the
use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
# Apply a 10ns max delay between two unrelated clocks
set_max_delay -from [get_clocks clkA] -to [get_clocks clkB] 10.000

# Apply a 2ns max delay for an input port (TSU)
set_max_delay -from [get_ports in[*]] -to [get_registers *] 2.000

# Apply a 2ns max delay for an output port (TCO)
set_max_delay -from [get_registers *] -to [get_ports out[*]] 2.000

# Apply a 2ns max delay for an input port to an output port (TPD)
set_max_delay -from [get_ports in[*]] -to [get_ports out[*]] 2.000

# Apply a 2ns max delay for an input port only to nodes driven by
# the rising edge of clock CLK
set_max_delay -from [get_ports in[*]] -rise_to [get_clocks CLK] 2.000
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.35.28. set_max_time_borrow (::quartus::sdc)

The following table displays information for the `set_max_time_borrow` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_max_time_borrow [-h | -help] [-long_help] [-exact] <value> <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-exact`	Forces the time borrowed to be the exact value provided (if physically possible)

continued...

	<value>	Time Value
	<targets>	Collection or list of latches
Description		

```
Specifies the maximum borrowed time for level-sensitive latches. The actual borrowed time
will be determined automatically, but will never exceed the amount you specify. For any
latches without a set_max_time_borrow constraint, no limit will apply (except for the physical
limit of what is possible on the device, as described below).

By using the -exact option, you can bypass the automatic algorithm and specify the exact
amount of borrowing at a given latch. For optimum results, using the automatic algorithm
is recommended (ideally, without any set_max_time_borrow constraints).

Time borrowing is specified with respect to the earliest possible time a signal can be
clocked into the latch node. For example, for a positive latch, if the earliest possible
arrival time of the rising clock edge is 1.025ns, then a signal that has an arrival time
of 1.035ns (where this arrival time already includes the micro-setup time of the latch)
will require at least 0.010ns of time borrowing.

Regardless of how the borrowed time is determined (automatically without a limit, automatically
with a set_max_time_borrow constraint, or manually with a set_max_time_borrow -exact constraint),
the borrowed time can never exceed what is physically possible to borrow on the device.
The maximum amount that can be borrowed is the period of time when the latch is open
(e.g. half the clock period if the clock has a 50% duty cycle), but this time is reduced
by clock propagation time spread and clock uncertainty between the latch-opening and
latch-closing clock edges, and is further reduced by the closing-edge setup time of the latch.
Some of these factors vary from corner to corner, as well as from clock to clock
(if multiple clocks drive the latch).

Time borrowing analysis will only occur in the Timing Analysis (Signoff) stage, or when
manually running the Timing Analyzer. The Fitter will not utilize time borrowing information
and will assume zero time borrowed. Thus, the use of level-sensitive latches with high-speed
clocks is not recommended, unless other constraints (such as set_max_delay) are manually set
to ensure optimal Fitter behavior.

The targets of this command must be level-sensitive latches (all other targets will
be ignored).

The targets can be specified as either a collections or a Tcl list of wildcards
used to create collections of appropriate types. The values used must follow
standard Tcl or Timing Analyzer-extension substitution rules. See help for the
use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
# Borrow at most 3ns at all "lat*" latches:
set_max_time_borrow 3 [get_registers lat*]
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Option *<string>* has illegal value: *<string>*. Specify a legal option value.

3.1.35.29. set_min_delay (::quartus::sdc)

The following table displays information for the `set_min_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_min_delay [-h | -help] [-long_help] [-fall_from` *<names>* `] [-fall_to` *<names>* `] [-from` *<names>* `] [-rise_from` *<names>* `] [-rise_to` *<names>* `] [-through` *<names>* `] [-to` *<names>* `]` *<value>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fall_from` *<names>*	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to` *<names>*	Valid destination clocks (string patterns are matched using Tcl string matching)

continued...

	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	<value>	Time Value

Description

```
Specifies a minimum delay exception for a given path.

The minimum delay is similar to changing the hold relationship
(launching clock edge - latching clock edge), except that it can be
applied to input or output ports without input or output delays
assigned to them. Minimum delays are always relative to any clock
network delays (if the source or destination is register) or any input
or output delays (if the source or destination is a port). Therefore,
input delays and clock latencies are added to the data arrival times.
Clock latencies also added to data required times and output delays
are subtracted from data required times.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design. If the -from or -to values are not
specified, the collection is converted automatically into
[get_keepers *]. It is worth noting that if the counterpart of the
unspecified collection is a clock collection, it is more efficient
to explicitly specify this collection as a clock collection, but only
if the clock collection also generates the desired assignment.

Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or
ports driven by the -to clock. Also, applying exceptions between
a pair of clocks is more efficient than for specific node to node or
node to clock paths.

If pin names or collections are used, the -from value must be a
clock pin and the -to value must be any non-clock input pin to
a register. Assignments from clock pins or to and from cells applies
to all registers in the cell or driven by the clock pin.

The -through values are collections of pins or nets in the
design. An exception applied through a node in the design applies
only to paths through the specified node.
The Timing Analyzer allows you to specify the -through argument multiple times to
describe paths that go through multiple points.
For instance, users can select all paths that go through node X, and then go through node Y.
This helps you narrow down and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
destination nodes specified using the -from option. The rise or fall
value of the option indicates that the "from" nodes are driven by the
rising or falling edge of the clock that feeds this node taking into
consideration any logical inversions along the clock path.  The -from
option is the combination of both rising and falling "from" nodes.  If
the -from collection is a clock collection, the assignment applies to
those nodes that are driven by the respective rising or falling clock
edge.

The -rise_to and -fall_to options behave similarly to the "from"
options described previously.  These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value taking into consideration any logical
inversions that are along the clock path.

The values of the -from, -to, -through, and other similar options are
either collections or a Tcl list of wildcards used to create
collections of appropriate types. The values used must follow standard
Tcl or Timing Analyzer-extension substitution rules.  See help for the
use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
# Apply a 0ns min delay between two unrelated clocks
set_min_delay -from [get_clocks clkA] -to [get_clocks clkB] 0.000
```

continued...

```
# Apply a 0ns min delay for an input port (TH)
set_min_delay -from [get_ports in[*]] -to [get_registers *] -.000

# Apply a 0.5ns min delay for an output port (MIN_TCO)
set_min_delay -from [get_registers *] -to [get_ports out[*]] 0.500

# Apply a 0.5ns min delay for an input port to an output port (MIN_TPD)
set_min_delay -from [get_ports in[*]] -to [get_ports out[*]] 0.500

# Apply a 0.5ns min delay for an input port only to nodes driven by
# the falling edge of clock CLK
set_max_delay -from [get_ports in[*]] -fall_to [get_clocks CLK] 0.500
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful

3.1.35.30. set_multicycle_path (::quartus::sdc)

The following table displays information for the `set_multicycle_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_multicycle_path [-h | -help] [-long_help] [-end] [-fall_from <names> ] [-fall_to <names> ] [-from <names> ] [-hold] [-rise_from <names> ] [-rise_to <names> ] [-setup] [-start] [-through <names> ] [-to <names> ] <value>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-end`	Specifies that the multicycle is relative to the destination clock waveform (default)
	`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-hold`	Specifies that the multicycle value applies to clock hold or removal checks
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Specifies that the multicycle value applies to clock setup or recovery checks (default)
	`-start`	Specifies that the multicycle is relative to the source clock waveform
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	<value>	Number of clock cycles

continued...

Description

```
Specifies a multicycle exception for a given set of paths.

Multicycles can be specified relative to the source clock (-start) or
destination clock (-end). This is useful when the source clock and
destination clock are operating at different frequencies.  For
example, if the source clock is twice as fast (half period) as the
destination clock, a -start multicycle of 2 is usually required.

Hold multicycles (-hold) are computed relative to setup multicycles
(-setup).  The value of the hold multicycle represents the number
clock edges away from the default hold multicycle. The default hold
multicycle value is 0.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design. If the -from or -to values are not
specified, the collection is converted automatically into [get_keepers
*].  It is worth noting that if the counterpart of the unspecified
collection is a clock collection, it is more efficient to explicitly
specify this collection as a clock collection but only if the clock
collection also generates the desired assignment.

Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or ports
driven by the -to clock.  Also, applying exceptions between a pair of
clocks is more efficient than for specific node to node or node to
clock paths.

If pin names or collections are used, the -from value must be a clock
pin and the -to value must be any non-clock input pin to a register.
Assignments from clock pins or to and from cells applies to all
registers in the cell or driven by the clock pin.

The -through values are collections of pins or nets in the design.
An exception applied through a node in the design applies only to
paths through the specified node.
The Timing Analyzer allows you to specify the -through argument multiple times to
describe paths that go through multiple points.
For instance, users can select all paths that go through node X, and then go through node Y.
This helps you narrow down and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
"-from" destination nodes. The rise or fall value of the option
indicates that the "from" nodes are driven by the rising or falling
edge of the clock that feeds this node taking into consideration any
logical inversions along the clock path.  The "-from" option is the
combination of both rising and falling "from" nodes.  If the "from"
collection is a clock collection, the assignment applies to those
nodes that are driven by the respective rising or falling clock edge.

The -rise_to and -fall_to options behave similarly to the "from"
options described previously.  These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value taking into consideration any logical
inversions that are along the clock path.

The values of the -from, -to, -through, and similar options are either
collections or a Tcl list of wildcards used to create collections of
appropriate types. The values used must follow standard Tcl or
Timing Analyzer-extension substitution rules. See help for the
use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
create_clock -period 10.000 -name CLK [get_ports clk]
create_generated_clock -divide_by 2 -source [get_ports clk] -name CLKDIV2 [get_registers
clkdiv]

# Apply a source multicycle of 2 with a hold multicycle of 1 for all
# paths from the CLK domain to the CLKDIV2 domain.
set_multicycle_path -start -setup -from [get_clocks CLK] -to [get_clocks CLKDIV2] 2
set_multicycle_path -start -hold -from [get_clocks CLK] -to [get_clocks CLKDIV2] 1

# Apply a multicycle constraint of 3 (with a default hold multicycle of 0) for a
# specific path in the design.
set_multicycle_path -end -setup -from [get_pins rega|clk] -to [get_pins regb|*] 3

# Apply a multicycle constraint of 2 to a given cell, except for the reset pin.
set_multicycle_path -end -setup -to [get_cells regb] 2
set_multicycle_path -end -setup -to [get_pins regb|aclr] 1

#Apply a multicycle constraint of 3 rising from a clock
set_multicycle_path -end -setup -rise_from [get_clocks CLK] 3
```

Return Value	Code Name	Code	String Return

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.
	TCL_ERROR	1	ERROR: The <*string*> value is outside of the legal range.

3.1.35.31. set_output_delay (::quartus::sdc)

The following table displays information for the `set_output_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc on page 562	
Syntax	`set_output_delay [-h | -help] [-long_help] [-add_delay] [-blackbox] -clock <name> [-clock_fall] [-fall] [-max] [-min] [-reference_pin <name> ] [-rise] [-source_latency_included] <delay> <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add_delay`	Create additional delay constraint instead of overriding previous constraints
	`-blackbox`	Create an assignment for a partition boundary port causing it to be treated as a port
	`-clock <name>`	Clock name
	`-clock_fall`	Specifies output delay relative to the falling edge of the clock
	`-fall`	Specifies the falling output delay at the port
	`-max`	Applies value as maximum data required time
	`-min`	Applies value as minimum data required time
	`-reference_pin <name>`	Specifies a pin or port in the design to which the output delay is relative
	`-rise`	Specifies the rising output delay at the port
	`-source_latency_included`	Specifies input delay already includes added source latency
	<*delay*>	Time value
	<*targets*>	Collection or list of output ports
Description		

```
Specifies the data required times at the specified output ports
relative the clock specified by the -clock option. The clock must
refer to a clock name in the design.

Output delays can be specified relative to the rising edge (default)
or falling edge (-clock_fall) of the clock.

Output delays can be specified relative to a pin or a port (-reference_pin)
in the clock network. Clock arrival times to the reference pin or port
are added to the data required time.

If no -reference_pin is specified, if the output delay is specified
relative to a generated clock with a single target, the clock arrival
times to the generated clock are added to the data required time. If the
generated clock has multiple targets, the worst case arrival time to
those targets will be used.

Output delays can include clock source latency. By default the clock
source latency of the related clock is added to the output delay
value, but when the -source_latency_included option is specified, the
clock source latency is not added because it was factored into the
output delay value.
```

continued...

```
The maximum output delay (-max) is used for clock setup checks or
recovery checks and the minimum output delay (-min) is used for clock
hold checks or removal checks. If only one of -min and -max (or
neither) is specified for a given port, the same value is used for
both.

Separate rising (-rise) and falling (-fall) required times at the port
can be specified. If only one of -rise and -fall are specified for a
given port, the same value is used for both.

By default, set_output_delay removes any other output delays to the port
except for those with the same -clock, -clock_fall, and -reference_pin
combination. Multiple output delays relative to different clocks, clock
edges, or reference pins can be specified using the -add_delay option.
Future output delays relative to different clocks, clock edges, or
reference pins that do not specify the -add_delay option will still
remove previous output delays that specified the -add_delay option.

The value of the targets is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.
The values used must follow standard Tcl or Timing Analyzer-extension
substitution rules.  See help for the use_timing_analyzer_style_escaping
command for details.
```

Example Usage

```
# Simple output delay with the same value for min/max and rise/fall:
# 1) set on ports with names of the form myout*
set_output_delay -clock clk 0.5 [get_ports myout*]
# 2) set on all output ports
set_output_delay -clock clk 0.5 [all_outputs]

# Output delay with respect to the falling edge of clock
set_output_delay -clock clk -clock_fall 0.5 [get_ports myout*]

# Output delays for different min/max and rise/fall combinations
set_output_delay -clock clk -max -rise 0.5 [get_ports myout*]
set_output_delay -clock clk -max -fall 0.4 [get_ports myout*]
set_output_delay -clock clk -min -rise 0.4 [get_ports myout*]
set_output_delay -clock clk -min -fall 0.3 [get_ports myout*]

# Adding multiple output delays with respect to more than one clock
set_output_delay -clock clkA -min 0.2 [get_ports myout*]
set_output_delay -clock clkA -max 0.8 [get_ports myout*]
set_output_delay -clock clkA -clock_fall 0.6 [get_ports myout*] -add_delay
set_output_delay -clock clkB -min 1.1 [get_ports myout*] -add_delay
set_output_delay -clock clkB -max 1.5 [get_ports myout*] -add_delay

# This is a common mistake where output delays are accidentally removed
set_output_delay -clock clkA -min 0.2 [get_ports myout*]
set_output_delay -clock clkB -min 1.1 [get_ports myout*] -add_delay
# The following removes the clkB entry. You need to always use -add_delay
#     when more than one clock, clock_fall or reference_pin exists
set_output_delay -clock clkA -max 0.8 [get_ports myout*]
# The following removes the clkA entry. You need to always use -add_delay
#     when more than one clock, clock_fall or reference_pin exists
set_output_delay -clock clkB -max 1.5 [get_ports myout*]

# Specifying an output delay relative to an external clock output port
set_output_delay -clock clk -reference_pin [get_ports clkout] 0.8 [get_ports myout*]

# Specifying an output delay relative to the clock pin of a register
set_output_delay -clock clk -reference_pin [get_pins regA|clk] 0.8 [get_ports myout*]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Incorrect collection type. Expected a collection of type <*string*>.
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36. ::quartus::sdc_ext

The following table displays information for the **::quartus::sdc_ext** Tcl package:

Tcl Package and Version	::quartus::sdc_ext 2.0
Description	Timing Constraints not defined in the SDC Spec Version 1.5 are implemented in this package. Any command in this package can be specified in a Timing Analyzer SDC file.
Availability	This package is loaded by default in the following executable: quartus_sta This package is available for loading in the following executable: quartus_fit
Tcl Commands	derive_clock_uncertainty (::quartus::sdc_ext) on page 595 derive_pll_clocks (::quartus::sdc_ext) on page 596 disable_min_pulse_width (::quartus::sdc_ext) on page 597 get_active_clocks (::quartus::sdc_ext) on page 597 get_fanins (::quartus::sdc_ext) on page 598 get_fanouts (::quartus::sdc_ext) on page 599 get_keepers (::quartus::sdc_ext) on page 601 get_nodes (::quartus::sdc_ext) on page 601 get_partitions (::quartus::sdc_ext) on page 602 get_registers (::quartus::sdc_ext) on page 603 remove_annotated_delay (::quartus::sdc_ext) on page 604 remove_clock (::quartus::sdc_ext) on page 605 reset_timing_derate (::quartus::sdc_ext) on page 605 set_active_clocks (::quartus::sdc_ext) on page 606 set_annotated_delay (::quartus::sdc_ext) on page 606 set_data_delay (::quartus::sdc_ext) on page 608 set_max_skew (::quartus::sdc_ext) on page 610 set_net_delay (::quartus::sdc_ext) on page 612 set_scc_mode (::quartus::sdc_ext) on page 613 set_time_format (::quartus::sdc_ext) on page 614 set_timing_derate (::quartus::sdc_ext) on page 614

3.1.36.1. derive_clock_uncertainty (::quartus::sdc_ext)

The following table displays information for the `derive_clock_uncertainty` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`derive_clock_uncertainty [-h | -help] [-long_help] [-add] [-overwrite]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add`	Adds results to user-defined clock uncertainty assignments
	`-overwrite`	Overwrites user-defined clock uncertainty assignments
Description	Applies inter-clock, intra-clock and I/O interface uncertainties based on timing model characterization. This command calculates and applies setup and hold clock uncertainties for each clock-to-clock transfer found in the design. The calculation of the uncertainties is delayed until the next update_timing_netlist call. To get I/O interface uncertainty in addition to inter-clock and intra-clock uncertainties, create a virtual clock to represent an off-chip clock for input or output delay specification and assign delays to input/output ports with set_input_delay and set_output_delay commands that specify the virtual clock. If set_input_delay and set_output_delay commands specifying a non-	

continued...

```
virtual clock are called, derive_clock_uncertainty applies either
inter-clock or intra-clock uncertainty for that clock transfer since
those transfers represent a clock-to-clock domain rather than an
I/O-to-register clock domain.

The set_clock_uncertainty calls will override the derived values for a
source clock and destination clock pair unless either the
set_clock_uncertainty command or the derive_clock_uncertainty command
specified the -add option, in which case the values are added. Specifying
the -overwrite option will instead cause all set_clock_uncertainty commands
to be ignored. Previous set_clock_uncertainty assignments can also be
manually removed by using the remove_clock_uncertainty command.

Note that this command is called automatically and the user only
needs to manually call it to specify the -add or -overwrite options.
```

Example Usage

```
# create a virtual clock
create_clock -name virtual -period 1

# apply input/output delays with the virtual clock to get
# I/O interface uncertainties
set_input_delay -clock virtual -add_delay 0 [all_inputs]
set_output_delay -clock virtual -add_delay 0 [all_outputs]

# call derive_clock_uncertainty. results will be calculated
# at the next update_timing_netlist call
derive_clock_uncertainty

update_timing_netlist
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.36.2. derive_pll_clocks (::quartus::sdc_ext)

The following table displays information for the `derive_pll_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`derive_pll_clocks [-h | -help] [-long_help] [-create_base_clocks] [-use_net_name]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-create_base_clocks`	Creates base clocks on input clock ports of the design that are feeding the PLL
	`-use_net_name`	Use net names as clock names

Description

```
NOTE: This command is no longer supported for Stratix 10 and later families.

Identifies PLLs or similar resources in the design and creates
generated clocks for their output clock pins. Multiple generated
clocks may be created for each output clock pin if the PLL is using
clock switchover, one for the inclk[0] input clock pin and one for the
inclk[1] input clock pin.

By default this command does not create base clocks on input clock
ports that are driving the PLL. When you use the create_base_clocks
option, derive_pll_clocks also creates the base clock on an input
clock port deriving the PLL. This option does not overwrite an
existing clock.

By default the clock name is the same as the output clock pin name.
To use the net name, use the -use_net_name option.

Note that this command is not supported for Stratix 10 and later device
families. The only families that still have support for this command are
Arria 10 and Cyclone 10GX. The reason why this command has been deprecated
is because all PLL clocks are now automatically generated by the SDC files
generated alongside the PLL IP. No user action is required.
```

continued...

Example Usage	<pre>project_open top create_timing_netlist # Create the base clock for the input clock port driving the PLL create_clock -period 10.0 [get_ports sysclk] # Create the generated clocks for the PLL. derive_pll_clocks update_timing_netlist # Other user actions report_timing delete_timing_netlist project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.3. disable_min_pulse_width (::quartus::sdc_ext)

The following table displays information for the `disable_min_pulse_width` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`disable_min_pulse_width [-h | -help] [-long_help] <targets>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<targets>	Elements to disable MPW checks for
Description	<pre>Allows you to disable minimum pulse width checks for specified targets. If the target is a clock collection, all MPW checks along that clock path will be disabled. Otherwise, MPW checks for the elements in the passed target collection will be disabled.</pre>	
Example Usage	<pre>disable_min_pulse_width -targets reg[*] disable_min_pulse_width -targets [get_clocks {clkA}]</pre>	
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.36.4. get_active_clocks (::quartus::sdc_ext)

The following table displays information for the `get_active_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`get_active_clocks [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

Description	<pre>Returns the collection of active clocks for timing analysis. The active clocks are the clocks specified in the most recent call to set_active_clocks, or all clocks if set_active_clocks has not been called.</pre>		
Example Usage	<pre># Set some active clocks set_active_clocks [get_clocks {clk1 clk2 sysclk*}] # Update the active clocks to exclude clk1 set_active_clocks [remove_from_collection [get_active_clocks] [get_clocks clk1]]</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.5. get_fanins (::quartus::sdc_ext)

The following table displays information for the `get_fanins` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`get_fanins [-h | -help] [-long_help] [-asynch] [-clock] [-inverting_paths] [-no_logic] [-non_inverting_paths] [-stop_at_clocks] [-synch] [-through` *<names>* `] <filter>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-asynch`	Traverse through asynch edges
	`-clock`	Traverse through clock edges
	`-inverting_paths`	Only follow inverting combinational paths
	`-no_logic`	Do not follow combinational paths
	`-non_inverting_paths`	Only follow non-inverting combinational paths
	`-stop_at_clocks`	Return clock targets as fanin/fanouts rather than traversing through them
	`-synch`	Traverse through synch edges
	`-through` *<names>*	Valid through nodes (string patterns are matched using Tcl string matching)
	<filter>	Valid starting nodes (string patterns are matched using Tcl string matching or collection)
Description	<pre>Returns a collection of fanin ports, registers (and optionally clock targets) reachable from the <filter> in the design. When you supply the -no_logic option, get_fanins ignores paths that pass through combinational logic elements other than buffers and inverters. NOTE: the -no_logic option does not consider logic absorbed into the cells of the <filter> nor the cells of fanin registers, ports or clock targets. When you use -synch, -asynch, or -clock options, get_fanins traverses the netlist through corresponding edges. You can specify more than one of these options. If you do not specify any of these three</pre>	

continued...

```
options, the command does not ignore any paths.

When the -non_inverting_paths option is used, no_logic
does not follow any paths that includes odd number of
inverters. Similarly, when the -inverting_paths option
is used, no_logic does not follow any paths that
includes even number of inverters. Both the
-non_inverting_paths and -inverting_paths options
require the -no_logic option and are mutually
exclusive.

When the -through option is used, only the fanins that
can be reached by going through those nodes are
returned.

When -stop_at_clocks is used, combinational clock targets may
be returned (in addition to clock or non-clock registers
and ports), and registers or ports that can only be reached
by traversing through a clock target will not be returned.

The filter for the collection is a Tcl list of
wildcards, and must follow standard Tcl or
Timing Analyzer-extension substitution rules.  See the help
for the use_timing_analyzer_style_escaping command for
details.
```

Example Usage

```
set fanins [get_fanins $item -synch -clock]
foreach_in_collection fanin_keeper $fanins {
    lappend fanin_keeper_list [get_node_info $fanin_keeper -name]
}

set fanins_no_logic [get_fanins $item -no_logic -asynch]
foreach_in_collection fanin_keeper $fanins_no_logic {
    lappend fanin_keeper_list_no_logic [get_node_info $fanin_keeper -name]
}

#-through example
get_fanins inst18 -through inst11
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.36.6. get_fanouts (::quartus::sdc_ext)

The following table displays information for the `get_fanouts` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`get_fanouts [-h | -help] [-long_help] [-asynch] [-clock] [-inverting_paths] [-no_logic] [-non_inverting_paths] [-stop_at_clocks] [-synch] [-through` *<names>* `]` *<filter>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-asynch`	Traverse through asynchronous edges
	`-clock`	Traverse through clock edges
	`-inverting_paths`	Only follow inverting combinational paths
	`-no_logic`	Do not follow combinational paths
	`-non_inverting_paths`	Only follow non-inverting combinational paths
	`-stop_at_clocks`	Return clock targets as fanin/fanouts rather than traversing through them
	`-synch`	Traverse through synchronous edges

continued...

	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	<filter>	Valid starting nodes (string patterns are matched using Tcl string matching or collection)

Description

```
Returns a collection of fanout ports, registers (and
optionally clock targets) reachable from the
<filter> in the design. When the -no_logic option is
used, get_fanouts ignores the paths that pass through
combinational logic elements other than buffers and
inverters.

When you use the -synch, -asynch, or -clock options,
get_fanouts traverses the netlist through corresponding
edges. You can specify more than one of these options.
If you do not specify any of these three
options, the command does not ignore any paths.

When the -non_inverting_paths option is used in
conjunction with the -no_logic option, get_fanouts
does not follow paths that include an odd number
of inverters. Similarly, when the -inverting_paths
option is used in conjunction with the -no_logic
option, get_fanouts does not follow any paths that
include an even number of inverters. Both the
-non_inverting_paths and -inverting_paths options
require the -no_logic option and are mutually
exclusive.

NOTE:
The -no_logic option does not consider logic absorbed
into the cells of the <filter> nor the cells of fanout
registers, ports or clock targets.
The -no_inversion option does not consider inversions
absorbed into synchronous inputs of fanout registers,
as the Intel Quartus Prime software register timing
model always considers these edges as being positive
unate.

When the -through option is used, only the fanouts
that can be reached by going through those nodes are
returned.

When -stop_at_clocks is used, combinational clock targets may
be returned (in addition to clock or non-clock registers
and ports), and registers or ports that can only be reached
by traversing through a clock target will not be returned.

The filter for the collection is a Tcl list of
wildcards, and must follow standard Tcl or
Timing Analyzer-extension substitution rules. See the help
for use_timing_analyzer_style_escaping for details.
```

Example Usage

```
set fanouts [get_fanouts $item]
foreach_in_collection fanout_keeper $fanouts {
    lappend fanout_keeper_list [get_node_info $fanout_keeper -name]
}

set fanouts_no_logic [get_fanouts $item -no_logic]
foreach_in_collection fanout_keeper $fanouts_no_logic {
    lappend fanout_keeper_list_no_logic [get_node_info $fanout_keeper -name]
}

# Using through option to find the fanout registers whose enable input is
# connected to the signal while ignoring the inverting paths.
get_fanouts inst1 -no_logic -non_inverting_paths -through [get_pins -hierarchical *|ena]
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.36.7. get_keepers (::quartus::sdc_ext)

The following table displays information for the `get_keepers` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595		
Syntax	`get_keepers [-h | -help] [-long_help] [-no_duplicates] [-nocase] [-nowarn] [ <filter> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-no_duplicates`	Do not match duplicated keeper names	
	`-nocase`	Specifies the matching of node names to be case-insensitive	
	`-nowarn`	Do not issue warning messages about unmatched patterns	
	<filter>	Valid destinations (string patterns are matched using Tcl string matching)	
Description	`Returns a collection of non-combinational or "keeper" nodes in the design.` `The default matching scheme returns not only non-combinational nodes whose names match the specified filter, but also non-combinational nodes duplicated from these keepers (refers to cells are automatically generated by Quartus from these keepers). Use the -no_duplicates option to exclude duplicated keepers.`  `The filter for the collection is a Tcl list of wildcards, and must follow standard Tcl or Timing Analyzer-extension substitution rules. See help for the use_timing_analyzer_style_escaping command for details.`		
Example Usage	`project_open chiptrip` `create_timimg_netlist` `set kprs [get_keepers *reg*]` `foreach_in_collection kpr $kprs {` `  puts [get_object_info -name $kpr]` `}` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.8. get_nodes (::quartus::sdc_ext)

The following table displays information for the `get_nodes` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`get_nodes [-h | -help] [-long_help] [-include_generated_clocks] [-no_duplicates] [-nocase] [-nowarn] [-of_clocks] [ <filter> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	-include_generated_clocks	Includes generated clocks derived from the matched clocks. This option can only be used in conjunction with the -of_clocks option.
	-no_duplicates	Do not match duplicated node names
	-nocase	Specifies the matching of node names (or clock names if the -of_clocks option is used) to be case-insensitive
	-nowarn	Do not issue warning messages about unmatched patterns
	-of_clocks	Returns nodes that are on the clock domains of the specified clock names
	<filter>	Valid destinations (string patterns are matched using Tcl string matching)

Description

```
Returns a collection of nodes in the design.

The default matching scheme returns not only nodes
whose names match the specified filter, but also nodes
duplicated from these nodes (refers to cells are
automatically generated by Quartus from these
nodes). Use the -no_duplicates option to not include
duplicated nodes. If the -of_clocks option is used, the
nodes returned are limited to those on the clock domains
of clocks whose names match the specified filter.

The filter for the collection is a Tcl list of
wildcards, and must follow standard Tcl or
Timing Analyzer-extension substitution rules. See help for
the use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
project_open chiptrip
create_timimg_netlist

set nodes [get_nodes *name*]
foreach_in_collection node $nodes {
    puts [get_object_info -name $node]
}

delete_timing_netlist
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The -include_generated_clocks option can only be used when the -of_clocks option is used as well
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.9. get_partitions (::quartus::sdc_ext)

The following table displays information for the `get_partitions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`get_partitions [-h | -help] [-long_help] [-cell] [-hierarchical] [-nocase] [ <filter> ]`	
Arguments	-h | -help	Short help
	-long_help	Long help with examples and possible return values
	-cell	Returns a cell collection inside the partitions matching the <filter>

continued...

	-hierarchical		Specifies if hierarchical searching method should be used
	-nocase		Specifies the matching of node names to be case-insensitive
	<*filter*>		Valid partitions (string patterns are matched using Tcl string matching)
Description	Returns a collection of partitions matching the filter by default. All partition names in the collection match the specified pattern. Wildcards can be used to select multiple partitions at once. The -cell option creates and returns the collection of cells found inside the partitions matching the <filter> instead of returning a partition collection. There are three Tcl string matching schemes available with this command: default, -hierarchical, and -no_case. When using the default matching scheme, pipe characters separate one hierarchy level from the next. They are treated as special characters and are taken into account when string matching with wildcards is performed. The default matching scheme does not force the search to proceed recursively down the hierarchy. Using the hierarchical matching scheme forces the search to proceed recursively down the hierarchy. The -nocase matching scheme uses case-insensitive matching behavior. The filter for the collection is a Tcl list of wildcards, and must follow standard Tcl or Timing Analyzer-extension substitution rules. See the help for use_timing_analyzer_style_escaping for details.		
Example Usage	#Get the partitions matching the filter get_partitions * #Get the collection of cells inside partitions matching the filter get_partitions * -cell		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: This command is not supported in this version of the software.

3.1.36.10. get_registers (::quartus::sdc_ext)

The following table displays information for the `get_registers` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`get_registers [-h | -help] [-long_help] [-latches] [-no_duplicates] [-nocase] [-nowarn] [ <filter> ]`	
Arguments	-h | -help	Short help
	-long_help	Long help with examples and possible return values
	-latches	Get only latches that match the filter
	-no_duplicates	Do not match duplicated register names
	-nocase	Specifies the matching of node names to be case-insensitive
	-nowarn	Do not issue warning messages about unmatched patterns

continued...

	<filter>	Valid destinations (string patterns are matched using Tcl string matching)
Description		

```
Returns a collection of registers in the design.

The default matching scheme returns not only registers
whose names match the specified filter, but also
returns registers duplicated from these registers
(cells automatically generated from these registers by
the Quartus Prime software). Use the -no_duplicates
option to exclude duplicated registers.

The filter for the collection is a Tcl list of
wildcards, and must follow standard Tcl or
Timing Analyzer-extension substitution rules. See help for
the use_timing_analyzer_style_escaping command for details.
```

Example Usage

```
project_open chiptrip
create_timing_netlist

set regs [get_registers *reg*]
foreach_in_collection reg $regs {
    puts [get_object_info -name $reg]
}

delete_timing_netlist
project_close
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.11. remove_annotated_delay (::quartus::sdc_ext)

The following table displays information for the `remove_annotated_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`remove_annotated_delay [-h | -help] [-long_help] -all`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all`	Specifies removal of all annotated delays
Description		

```
Removes annotated delays from the design.
```

Example Usage

```
# annotate delay
set_annotated_delay -net -from [get_pins clk] 0.1
update_timing_netlist

# remove all annotated delays
remove_annotated_delay -all
update_timing_netlist
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.12. remove_clock (::quartus::sdc_ext)

The following table displays information for the `remove_clock` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595		
Syntax	`remove_clock [-h | -help] [-long_help] [-all] [ <clock_list> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-all`		Removes all clocks from the design
	<clock_list>		Clock(s) to be removed
Description	`Removes the specified clock(s) from the design.`		
Example Usage	`# Create a clock and then remove it.` `create_clock -period 10 -name CLK [get_ports clk]` `remove_clock CLK`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.36.13. reset_timing_derate (::quartus::sdc_ext)

The following table displays information for the `reset_timing_derate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595		
Syntax	`reset_timing_derate [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Resets all derate factors set on the design.`		
Example Usage	`# set timing derate` `set_timing_derate -late 0.2 [get_cells *]` `update_timing_netlist` `# reset all derate factors` `reset_timing_derate` `update_timing_netlist`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.14. set_active_clocks (::quartus::sdc_ext)

The following table displays information for the `set_active_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`set_active_clocks [-h | -help] [-long_help] <clocks>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<clocks>	List or collection of clocks
Description		

```
Sets the list of active clocks for timing
analysis. All other clocks not in the list or
collection are considered inactive.

Timing analysis is only performed on active
clocks. All clocks are active by default. Generated
clocks that are generated from inactive clocks are
considered inactive. Therefore, to make a generated
clock active, specify both the parent and generated
clock when calling set_active_clocks.

To reset all clocks to active, call "set_active_clocks
*" or "set_active_clocks [all_clocks]".

The set_active_clocks command does not affect all
reports. For example, inactive clocks are still
reported by report_clocks, report_clock_transfers, and
similar commands.
```

Example Usage

```
# Only analyze clk1
set_active_clocks [get_clocks clk1]

# Only analyze clk2
set_active_clocks [get_clocks clk2]

# Analyze all clocks
set_active_clocks [all_clocks]
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.15. set_annotated_delay (::quartus::sdc_ext)

The following table displays information for the `set_annotated_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`set_annotated_delay [-h | -help] [-long_help] [-cell] [-ff] [-fr] [-from <names> ] [-max] [-min] [-net] [-operating_conditions <operating_conditions> ] [-rf] [-rr] [-to <names> ] <delay>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cell`	Specifies that cell delay must be set
	`-ff`	Specifies that FF delay must be set

continued...

	`-fr`		Specifies that FR delay must be set
	`-from` *<names>*		Valid source pins or ports (string patterns are matched using Tcl string matching)
	`-max`		Specifies that only max delay should be set
	`-min`		Specifies that only min delay should be set
	`-net`		Specifies that net delay must be set
	`-operating_conditions` *<operating_conditions>*		Operating conditions Tcl object
	`-rf`		Specifies that RF delay must be set
	`-rr`		Specifies that RR delay must be set
	`-to` *<names>*		Valid destination pins or ports (string patterns are matched using Tcl string matching)
	<delay>		The delay value in default time units
Description			
	``Annotates the cell delay between two or more pins/nodes on a cell, or the interconnect delay between two or more pins on the same net, in the current design. Multiple transition edges (rr, fr, rf, ff) can be specified. If no transition is specified, then the given delay is assigned to all four values. If either -from or -to (or both) values are left unspecified, the missing value or values are substituted by an "*" character. Options -max and -min allow users to specify max or only min delay. If neither -max or -min is specified, both delays are set. Using this command to reduce delay pessimism might lead to optimistic results from timing analysis.

The values for -from and -to are either collections or a Tcl list of wildcards used to create collections of appropriate types. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See help for the use_timing_analyzer_style_escaping command for details.

Delay annotation is deferred until the next time update_timing_netlist is called. To remove annotated delays, use remove_annotated_delay command.

This assignment is for timing analysis only, and is not considered during timing-driven compilation.``` | | |
| **Example Usage** | | | |
| | ```set_annotated_delay -cell 100 -from A\|B\|C\|datain -to A\|B\|C\|combout -rr -ff
set_annotated_delay -net 100 -to A\|carryin
update_timing_netlist

To clear all net delays
set_annotated_delay -net 0
update_timing_netlist

To remove all annotated delay assignments
remove_annotated_delay -all
update_timing_netlist``` | | |
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.16. set_data_delay (::quartus::sdc_ext)

The following table displays information for the `set_data_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`set_data_delay [-h | -help] [-long_help] [-add_latch_clock] [-add_launch_clock] [-allow_destination_borrowing] [-fall_from <names>] [-fall_to <names>] [-from <names>] [-get_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period|max_clock_period>] [-no_synchronizer] [-override] [-rise_from <names>] [-rise_to <names>] [-through <names>] [-to <names>] [-value_multiplier <multiplier>] [<value>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-add_latch_clock`	Include the latch clock path in timing analysis
	`-add_launch_clock`	Include the launch clock path in timing analysis
	`-allow_destination_borrowing`	Allow time borrowing at the destination
	`-fall_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-get_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period|max_clock_period>`	Compute constraint as a multiple of the clock period
	`-no_synchronizer`	Prevent this data delay from triggering a synchronizer
	`-override`	Make this constraint override non-datapath-only setup constraints, instead of applying it in addition to them (equivalent to set_data_delay & set_false_path -setup -no_synchronizer, unless -add_launch_clock is used as well)
	`-rise_from <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-value_multiplier <multiplier>`	Value by which the clock period should be multiplied to compute requirement
	`<value>`	Time Value
Description		

```
Specifies a maximum datapath delay exception for a given path.

The maximum delay analysis includes Tco of the launching register,
and Tsu of the latching register.  By default, it does not include
clock arrival times at the launching or latching register. To include
launch clock arrival times, use the -allow_launch_clock option. To
include latch clock arrival times, use the -allow_latch_clock option.
If the path starts or ends at a port, the analysis does not
include delays due to set_input_delay or set_output_delay.
```

continued...

```
Use -get_value_from_clock_period to set the delay requirement
for each path to be equal to the launching or latching clock period,
or whichever of the two has a smaller or larger period.
If -value_multiplier is used, the requirement will be multiplied by
that value.  If there are no clocks clocking the endpoints
of the path (such as if the path begins or ends at an unconstrained I/O),
the constraint will be ignored.

The datapath delay constraint is applied in addition to other constraints on
the given path, including the default constraint. Furthermore, the datapath delay
constraint is analyzed independently from other SDC constraints, including
set_false_path and set_clock_groups, and cannot be overridden by other SDC constraints.
For example, you can use set_data_delay to specify an upper limit on logic and routing
delay for paths cut by set_false_path.

To both cut a path for (clock-aware) timing and constrain its datapath delay, the path
must be constrained with both set_false_path and set_data_delay.

The -from and -to values are collections of clocks, registers, ports,
pins, or cells in the design.

Applying exceptions between clocks applies the exception from all
register or ports driven by the -from clock to all registers or ports
driven by the -to clock.

If pin names or collections are used, the -from value must be a clock
pin and the -to value must be any non-clock input pin to a register.
Assignments from clock pins or to and from cells applies to all
registers in the cell or driven by the clock pin.

The -through values are collections of pins or nets in the design.  An
exception applied through a node in the design applies only to paths
through the specified node.
the Timing Analyzer allows you to specify the -through argument multiple times to
describe paths that go through multiple points.
For instance, users can select all paths that go through node X, and then go through node Y.
This helps you narrow down and select the specific paths that you are interested in.

The -rise_from and -fall_from options can be used in place of the
-from destination nodes.  The rise or fall value of the option
indicates that the "from" nodes are driven by the rising or falling
edge of the clock that feeds this node taking into consideration any
logical inversions along the clock path.  The "-from" option is the
combination of both rising and falling "from" nodes.  If the "from"
collection is a clock collection, the assignment applies to those
nodes that are driven by the respective rising or falling clock edge.

The -rise_to and -fall_to options behave similarly to the "from"
options described previously.  These assignments restrict the given
assignment to only those nodes or clocks that correspond to the
specified rise or fall value taking into consideration any logical
inversions that are along the clock path.

The values of the -from, -to, -through, and other similar options are
either collections or a Tcl list of wildcards used to create
collections of appropriate types. The values used must follow standard
Tcl or Timing Analyzer-extension substitution rules. See help for the
use_timing_analyzer_style_escaping command for details.

If the source of a path with a set_data_delay constraint has any time
borrowed, the delay budget will be reduced by the time borrowed.

By default, the delay budget will not be increased by time borrowed at the
destination of a path constrained by a set_data_delay constraint, and
negative slack on a set_data_delay constraint will not cause time borrowing
to happen. To change this behavior, use the -allow_destination_borrowing
option.
```

Example Usage

```
# Apply a 10ns max data delay on paths between two unrelated clocks
set_data_delay -from [get_clocks clkA] -to [get_clocks clkB] 10.000

# Apply a 2ns max data delay from an input port to any register
set_data_delay -from [get_ports in[*]] -to [get_registers *] 2.000

# Require net delay to be at most 90% of the period of the clock driving the inst9 register
set_data_delay -get_value_from_clock_period dst_clock_period -value_multiplier 0.9 -from
[get_clocks clk] -to [get_keepers inst9]

# Apply a 2ns max data delay for an input port only to nodes driven by
# the rising edge of clock CLK
set_data_delay -from [get_ports in[*]] -rise_to [get_clocks CLK] 2.000
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.36.17. set_max_skew (::quartus::sdc_ext)

The following table displays information for the `set_max_skew` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`set_max_skew [-h | -help] [-long_help] [-fall_from_clock <names>] [-fall_to_clock <names>] [-from <names>] [-from_clock <names>] [-get_skew_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period>] [-rise_from_clock <names>] [-rise_to_clock <names>] [-skew_value_multiplier <multiplier>] [-to <names>] [-to_clock <names>] [<skew>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-get_skew_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period>`	Compute skew constraint as a multiple of the clock period
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-skew_value_multiplier <multiplier>`	Value by which the clock period should be multiplied to compute skew requirement
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	<skew>	Required skew
Description		

```
Use the set_max_skew constraint to perform maximum
allowable skew analysis between sets of registers or
ports. In order to constrain skew across multiple
paths, all such paths must be defined within a single
set_max_skew constraint. The set_max_skew timing
constraint is not affected by the set_max_delay,
set_min_delay, and set_multicycle_path constraints,
but is affected by the set_clock_groups -exclusive
constraint. Paths between exclusive clocks are not
analyzed for skew, and no two paths are compared for
skew if their clocks are exclusive to each other.
However, paths whose clocks are asynchronous are
```

continued...

```
still analyzed for skew.

Legal values for the -from and -to options are
collections of clocks, registers, ports, pins,
cells or partitions in a design.

Applying maximum skew constraints between clocks
applies the constraint from all register or ports
driven by the clock specified with the -from option to
all registers or ports driven by the clock specified
with the -to option.

If pin names or collections are used, the -from value
must be a clock pin and the -to value must be any
non-clock input pin to a register. Assignments from
clock pins or to and from cells apply to all registers
contained in the cell or driven by the clock pin.
Similarly, -to and -from partition specifications
apply to all registers in the specified partition.

Max skew analysis includes data arrival times, clock arrival times,
register micro parameters, clock uncertainty, on-die variation and
ccpp removal.

Use -get_skew_value_from_clock_period to set the skew
requirement to be equal to the launching or latching clock
period, or whichever of the two has a smaller period.
If -skew_value_multiplier is used, the requirement is multiplied
by that value.  If this option is used, then the positional
skew option may not be set.  If the set of skew paths is
clocked by more than one clock, the Timing Analyzer will use the
one with smallest period to compute the skew constraint.

When this constraint is used, results of max skew
analysis are displayed in the Report Max Skew
(report_max_skew) report from the Timing Analyzer.
Since skew is defined between two or more paths, no
results are displayed if the -from/-from_clock and
-to/-to_clock filters satisfy less than two paths.
```

Example Usage

```
# Constrain the skew on an input port to all registers it feeds
set_max_skew -from [get_ports din] 0.200

# Constrain the skew on output bus dout[*]
set_max_skew -to [get_ports dout\[*\]] 0.200

# Constrain skew to be less than 90% of the period of any clock  in the source
# register set
set_max_skew -to [get_keepers inst1|*] -get_skew_value_from_clock_period src_clock_period -
skew_value_multiplier 0.900

# Report the results of max skew assignments
report_max_skew -panel_name "Report Max Skew" -npaths 10 -detail path_only
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: *<string>* is not a valid argument for option -exclude. Available argument is to_clock
TCL_ERROR	1	ERROR: Following options are missing required arguments: *<string>*
TCL_ERROR	1	ERROR: The option -skew_value_multiplier must be a non-zero floating point number
TCL_ERROR	1	ERROR: The option -skew_value_multiplier may only be used if -get_skew_value_from_clock_period is used

3.1.36.18. set_net_delay (::quartus::sdc_ext)

The following table displays information for the `set_net_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`set_net_delay [-h | -help] [-long_help] -from <names> [-get_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period|max_clock_period> ] [-max] [-min] [-to <names> ] [-value_multiplier <multiplier> ] [ <delay> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-from <names>`	Valid source pins, ports, registers or nets (string patterns are matched using Tcl string matching)
	`-get_value_from_clock_period <src_clock_period|dst_clock_period|min_clock_period|max_clock_period>`	Compute net delay constraint as a multiple of the clock period
	`-max`	Specifies maximum delay
	`-min`	Specifies minimum delay
	`-to <names>`	Valid destination pins, ports, registers or nets (string patterns are matched using Tcl string matching)
	`-value_multiplier <multiplier>`	Value by which the clock period should be multiplied to compute net delay requirement
	<delay>	Required delay
Description		

```
Use the set_net_delay command to query the net delays
and perform minimum or maximum timing analysis across
nets. The -from and -to options can be string patterns
or pin, port, register, or net collections. When pin
or net collection is used, the collection should
include output pins or nets.

If the -to option is unused or if the -to filter is an
"*" character, all the output pins and registers on
timing netlist become valid destination points.

When you use the -min option, slack is calculated by
looking at the minimum delay on the edge. If you use
-max option, slack is calculated with the maximum edge
delay.

Use -get_value_from_clock_period to set the net delay requirement
to be equal to the launching or latching clock period,
or whichever of the two has a smaller or larger period.
If -value_multiplier is used, the requirement will be multiplied by
that value.  If the set of nets is clocked by more than one clock,
the Timing Analyzer will use the one with smallest period to compute
the constraint for a -max constraint, and the largest period for
a -min constraint.  If there are no clocks clocking the endpoints
of the net (e.g. if the endpoints of the nets are not registers or
constrained ports), then the net delay constraint will be ignored.
```

Example Usage

```
project_open my_project
create_timing_netlist
read_sdc
update_timing_netlist

# add min delay constraint
set_net_delay -min 0.160 -from [get_pins inst9|combout] -to [get_pins *|dataf]

# add max delay constraint
set_net_delay -max 0.500 -from inst8|combout

# this is same as the previous call
set_net_delay -max 0.500 -from inst8|combout -to *
```

continued...

	`` # Require net delay to be at most 90% of the period of the clock driving the inst9 register set_net_delay -max -get_value_from_clock_period dst_clock_period -value_multiplier 0.9 -from inst8|combout -to [get_keepers inst9] update_timing_netlist report_net_delay -panel "Net Delay" ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: <*string*> is not a valid delay value
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: The option -value_multiplier must be a non-zero floating point number
	TCL_ERROR	1	ERROR: The option -value_multiplier may only be used if -get_value_from_clock_period is used

3.1.36.19. set_scc_mode (::quartus::sdc_ext)

The following table displays information for the `set_scc_mode` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595		
Syntax	`set_scc_mode [-h | -help] [-long_help] [-size <size> ] [-use_heuristic]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-size <size>`	Maximum SCC loop size	
	`-use_heuristic`	Always use heuristic for SCC processing	
Description	``` Allows you to set maximum Strongly Connected Components (SCC) loop size or force the Timing Analyzer to always estimate delays through SCCs. When the Timing Analyzer encounters a loop of size greater than the specified maximum SCC loop size, it uses a heuristic which only estimates delays through the loop. If the loop is smaller than the maximum SCC loop size, a full processing of loops is performed unless the -use_heuristic option is used. ```		
Example Usage	``` # Make the Timing Analyzer use normal processing for all loops # the size of which is less than or equal to 100. For loops of size # greater than 100, a runtime-saving heuristic will be used set_scc_mode -size 100 # Force the Timing Analyzer to use heuristic for all SCCs # disregarding their size set_scc_mode -use_heuristic ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.36.20. set_time_format (::quartus::sdc_ext)

The following table displays information for the `set_time_format` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595		
Syntax	`set_time_format [-h | -help] [-long_help] [-decimal_places` *<decimal_places>* `] [-unit` *<unit>* `]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-decimal_places` *<decimal_places>*	Number of decimal places to use	
	`-unit` *<unit>*	Default time unit to use	
Description	`Sets time format, including time unit and decimal places.` `Time units are assumed to be nanoseconds (ns) by default. The "-unit" option overrides the default time units. Legal time unit values are: ps, ns, us, ms.` `Time units are displayed with three decimal places by default. The "-decimal_places" option overrides the default number of decimal places to show.` `The smallest resolution of all times units is one picosecond (ps). Any additional specified precision will be truncated.`		
Example Usage	`# Create two clocks with a clock period of 8 nanoseconds.` `create_clock -period 8.000 clk1` `set_time_format -unit ps -decimal_places 0` `create_clock -period 8000 clk2`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The default time unit can be set to ms, us, ns, or ps. Please specify one of these units instead.

3.1.36.21. set_timing_derate (::quartus::sdc_ext)

The following table displays information for the `set_timing_derate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sdc_ext on page 595	
Syntax	`set_timing_derate [-h | -help] [-long_help] [-cell_delay] [-early] [-late] [-net_delay] [-operating_conditions` *<operating_conditions>* `] <derate_value> [ <cells> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cell_delay`	Specifies that derating factors are only to apply to cell delays
	`-early`	Specifies the minimum derating factor. This factor specifies how early the signal can arrive
	`-late`	Specifies the maximum derating factor. This factor specifies how late the signal can arrive
	`-net_delay`	Specifies that derating factors are only to apply to net delays

continued...

	-operating_conditions <*operating_conditions*>	Operating conditions Tcl object
	<*derate_value*>	Timing derate value
	<*cells*>	List of cell type objects
Description		

```
Sets the global derate factors for the current
design. The maxmimum and minimum delays of all timing
arcs in the design are multiplied by the factors
specified with the -late and -early options
respectively.  Only positive derate factors are
allowed. If neither the -cell_delay nor -net_delay
option is used, the derating factors apply to both
cell and net delays. For net delay derates, the derate
factor is applied to nets driven by matching cells.

Specifying a derate value of less than 1.0 for the
-late option or a derate value of greater than 1.0 for
the -early option reduces delay pessimisim, which
might lead to optimistic results from timing analysis.

The effect of set_timing_derate command is deferred
until the next time update_timing_netlist is
called. To reset derate factors to original values,
use the reset_timing_derate command.

This assignment is for timing analysis only, and is
not considered during timing-driven compilation.
```

Example Usage

```
set_timing_derate -early 0.9 [get_cells *]
set_timing_derate -late 1.1 [get_cells *]
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Option -<*string*> has illegal value: <*string*>. Specify a legal option value.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37. ::quartus::sta

The following table displays information for the **::quartus::sta** Tcl package:

Tcl Package and Version	::quartus::sta 1.0
Description	

```
This package contains the set of Tcl functions for obtaining
information from the Timing Analyzer.
```

Availability

```
This package is loaded by default in the following executables:

    quartus_fit
    quartus_sta
```

Tcl Commands

add_to_collection (::quartus::sta) on page 616
check_timing (::quartus::sta) on page 617
create_report_histogram (::quartus::sta) on page 619
create_slack_histogram (::quartus::sta) on page 621
create_timing_netlist (::quartus::sta) on page 623
create_timing_summary (::quartus::sta) on page 626
delete_sta_collection (::quartus::sta) on page 628
delete_timing_netlist (::quartus::sta) on page 628
enable_ccpp_removal (::quartus::sta) on page 629
enable_sdc_extension_collections (::quartus::sta) on page 630
get_available_operating_conditions (::quartus::sta) on page 630
get_cell_info (::quartus::sta) on page 631
get_clock_domain_info (::quartus::sta) on page 632
get_clock_fmax_info (::quartus::sta) on page 633
get_clock_info (::quartus::sta) on page 634

continued...

3.1.37.1. add_to_collection (::quartus::sta)

The following table displays information for the `add_to_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`add_to_collection [-h | -help] [-long_help] <collection_obj_1> <collection_obj_2>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	<collection_obj_1>	First object collection

continued...

	<collection_obj_2>	Second object collection
Description	This command takes two collections and returns a new collection that is a union of the two. The second collection is allowed to be a string, whereas the first has to be previously-created collection, either by passing any of the "get_" functions directly, or by passing a variable that contains a collection (see code examples for this command). If a collection is used for the second argument, the types in the second collection must be the same as or a subset of the types in the first collection. If the first collection consists of keepers, the second collection can only consist of keepers, registers or ports. If the first collection consists of partitions, the second collection can only consist of partitions or cells. If the first collection consists of nodes, the second collection can only consist of nodes, keepers, registers, ports, pins, nets or combinational nodes.	
Example Usage	`set kprs1 [get_keepers b*]` `set regs1 [get_registers a*]` `set regs_union [add_to_collection $kprs1 $regs1]` `#or:` `set regs_union [add_to_collection [get_keepers b*] $regs1]` `#or even:` `set regs_union [add_to_collection $kprs1 a*]` `# - note that the last statement will actually add all keepers with name a*` `#   not only registers! (will add IOs with name a*, if any)` `# Get the first 100 nodes in the collection.` `query_collection $regs_union -limit 100`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Cannot find specified collection. Specify an existing collection.

3.1.37.2. check_timing (::quartus::sta)

The following table displays information for the `check_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`check_timing [-h | -help] [-long_help] [-append] [-file` *<name>* `] [-include` *<check_list>* `] [-panel_name` *<name>* `] [-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file` *<name>*	Sends the results to an ASCII or HTML file. Depending on the extension
	`-include` *<check_list>*	Checks to perform
	`-panel_name` *<name>*	Sends the results to the panel and specifies the name of the new panel

continued...

	-stdout	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.

Description

```
Checks for problems in the design or problems with
design constraints. The check_timing command performs
a series of different checks based on user-specified
variables and options. There is no default list of
checks. Use the -include option to specify which
checks to perform. You must preceed check_timing with
update_timing_netlist.

The no_clock check reports whether registers have at
least one clock at their clock pin, and that ports
determined to be clocks have a clock assigned to them,
and also checks that PLLs have a clock assignment.

The multiple_clock check verifies that registers have
at most one clock at their clock pin. (When multiple
clocks reach a register clock pin, it is undefined
which clock is used for analysis.

The generated_clock check verifies that generated
clocks are valid. Generated clocks must have a source
that is triggered by a valid clock.

The no_input_delay check verifies that every input
port that is not determined to be a clock has an input
delay assignment.

The no_output_delay check verifies that every output
port has an output delay constraint.

The partial_input_delay check verifies that input
delays are complete, and ensures that input delays
have a rise-min, fall-min, rise-max, and fall-max
portion set.

The partial_output_delay check verifies that output
delays are complete, and makes sure that output delays
have a rise-min, fall-min, rise-max, and fall-max
portion set.

The io_min_max_delay_consistency check verifies that
min delay values specified by set_input_delay or
set_output_delay assignments are less than max delay
values.

The reference_pin check verifies that reference pins
specified in set_input_delay and set_output_delay
using the -reference_pin option are valid. A
reference_pin is valid if the -clock option specified
in the same set_input_delay/set_output_delay command
matches the clock that is in the direct fanin of the
reference_pin. Being in the direct fanin of the
reference_pin means that there must be no keepers
between the clock and the reference_pin.

The latency_override check reports whether the clock
latency set on a port or pin overrides the more
generic clock latency set on a clock. Clock latency
can be set on a clock, where the latency applies to
all keepers clocked by the clock, whereas clock
latency can also be set on a port or pin, where the
latency applies to registers in the fanout of the port
or pin.

The loops check verifies that there are no strongly
connected components in the netlist. These loops
prevent a design from being properly analyzed. The
loops check also reports if loops exist but were
marked so that they would not be traversed.

The latches check reports latches in the design and
warns that latches may not be analyzed properly. For
best results, change your design to remove latches
whenever possible.

The pos_neg_clock_domain check determines if any
register is clocked by both the rising and falling
edges of the same clock. If this scenario is necessary
such as in a clock multiplexer, create two separate
clocks that have similar settings and are assigned to
the same node.

The pll_cross_check checks the clocks that are
```

continued...

```
assigned to a PLL against the PLL settings defined in
design files. Inconsistent settings or an unmatched
number of clocks associated with the PLL are reported
to the user.

The uncertainty check reports each clock-to-clock
transfer that does not have a clock uncertainty
assignment set between the two clocks.  When a device
family has derive_clock_uncertainty support, this
report also checks if a user-defined
set_clock_uncertainty assignment has a less than
recommended clock uncertainty value.

The virtual_clock check reports all unreferenced
virtual clocks. It also reports if design does not
have any virtual clock assignment.

The partial_multicycle check ensures that each setup
multicycle assignment has a corresponding hold
multicycle assignment, and each hold muticycle
assignment has a corresponding setup multicycle
assignment.

The multicycle_consistency check reports all the
multicycle cases where a setup multicycle does not
equal one greater than the hold multicycle. Hold
multicycle assignments are usually one cycle less than
setup multicycle assignments.

The partial_min_max_delay check verifies that each
minimum delay assignment has a corresponding maximum
delay assignment, and vice versa.

The clock_assignments_on_output_ports check reports
all the clock assignments that have been applied to
output ports.

The input_delay_assigned_to_clock check verifies that
no input delay value is set for a clock. Input delays
set on clock ports are ignored because clock-as-data
analysis takes precedence.

The internal_io_delay check reports all the IO delays
that have no specifications for -reference_pin and
-source_latency_included, and -clock is a clock that
is not assigned to a top level input or output port.
```

Example Usage

```
# Constrain design
create_clock -name clk -period 4.000 -waveform { 0.000 2.000 } [get_ports clk]
set_input_delay -clock clk2 1.5 [get_ports in*]
set_output_delay -clock clk 1.6 [get_ports out*]
set_false_path -from [get_keepers in] -through [get_nets r1] -to [get_keepers out]

# Check if there were any problems
check_timing -include {loops latches no_input_delay partial_input_delay}
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.37.3. create_report_histogram (::quartus::sta)

The following table displays information for the `create_report_histogram` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`create_report_histogram [-h | -help] [-long_help] [-append] [-color_div` *<color_div>* `] [-color_list` *<color_list>* `] [-file` *<name>* `] [-max_data` *<max_data>* `] [-min_data` *<min_data>* `] [-num_bins` *<num_bins>* `] [-panel_name` *<name>* `] [-stdout] [-x_label` *<x_label>* `] [-x_unit` *<x_unit>* `] [-y_label` *<y_label>* `] <data>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-color_div` *<color_div>*	Color divisions for the created histogram
	`-color_list` *<color_list>*	List of colors for painting the created histogram
	`-file` *<name>*	Sends the results to an ASCII or HTML file. Depending on the extension
	`-max_data` *<max_data>*	Maximum data value of the created histogram
	`-min_data` *<min_data>*	Minimum data value of the created histogram
	`-num_bins` *<num_bins>*	Number of bins
	`-panel_name` *<name>*	Sends the results to the panel and specifies the name of the new panel
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-x_label` *<x_label>*	Text label on x-axis
	`-x_unit` *<x_unit>*	Unit to be displayed on x-axis
	`-y_label` *<y_label>*	Text label on y-axis
	<data>	List of data to be analyzed
Description		

```
Create a user defined histogram.

Use <data> to specify the data entries to be displayed
on the histogram. It can be a tcl list of either one
of the following two formats or a mix of the two:
{time_integer} or {time_integer number_count}, where
time_integer is an integer possibly with a unit
representing time (default unit is second), and
number_count is a positive integer specifying number
of entries (y value) of the corresponding
time_integer.

Use -num_bins to specify the number of bins, or the
number of bars to be displayed on the histogram.

Use -color_div and -color_list to specify the color of
each bin.  -color_div takes a tcl list of
time_integers (see <data> above).  Each entry in the
list specifies the upper bound of each color division
and therefore is forced to be a boundary of bins.
-color_list takes a tcl list of colors. Each color in
the list is used in the order specified and if less
color is given than color divisions, the list is
re-used.  For example, if specified "-color_div {-1 0
1} -color_list {red green}", then bins below -1 are
red, bins between -1 and 0 are green, bins
between 0 and 1 are red again, and bins larger
than 1 are blue again.  Posssible choices of
colors: black, blue, brown, green, grey, light_grey,
orange, purple, red, white.  Default -color_div is {0}
and default -color_list is {red blue}.

Use -max_data to specify the upper bound, i.e. largest
number to be included in the histogram.

Use -min_data to specify the lower bound,
i.e. smallest number to be included in the histogram.

Use -panel_name to specify the path and panel name of
the created histogram. e.g. "-panel_name {Folder
1||Histogram 1} creates a histogram named
"Histogram 1" and put it in a folder with the name
"Folder 1".

Use -x_label to specify the text label on x_axis.

Use -y_label to specify the text label on y_axis.
```

continued...

	Use -x_unit to specify a text unit to be attached to x_axis.
Example Usage	`# create a path-based slack histogram` `project_open my_project` `create_timing_netlist` `read_sdc` `update_timing_netlist` `# get path-based slack data in the format of a tcl list` `set data [list]` `set paths [get_timing_paths -setup -npaths 1000]` `foreach_in_collection path $paths {` `  append data [get_path_info $path -slack]` `}` `# output data to histogram` `create_report_histogram $data -panel_name {Path-based Slack Histogram} -num_bins 20` `delete_timing_netlist` `project_close`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.

3.1.37.4. create_slack_histogram (::quartus::sta)

The following table displays information for the `create_slack_histogram` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`create_slack_histogram [-h | -help] [-long_help] [-all_edges] [-append] [-clock_name <name> ] [-data_delay] [-file <name> ] [-hold] [-max_slack <max_slack> ] [-min_slack <min_slack> ] [-num_bins <num_bins> ] [-panel_name <name> ] [-partition] [-recovery] [-removal] [-setup] [-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all_edges`	Consider slacks at all edges, not just at the endpoint nodes (results in increased memory consumption)
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-clock_name <name>`	Name of the Clock Domain
	`-data_delay`	Data Delay Analysis (only applicable for setup and recovery analysis)
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-hold`	Hold Analysis
	`-max_slack <max_slack>`	Maximum slack value of the created histogram
	`-min_slack <min_slack>`	Minimum slack value of the created histogram
	`-num_bins <num_bins>`	Number of bins

continued...

	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-partition`	Show slack count for each of the partition
	`-recovery`	Recovery Analysis
	`-removal`	Removal Analysis
	`-setup`	Setup Analysis
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.

Description

```
Creates a slack histogram in the timing report for the specified clock
domain "-clock_name," showing the number of timing edges within
various ranges of slacks for a clock setup analysis.  The histogram
can be named using the "-panel_name" option.

By default, only slack at the endpoint nodes of the timing netlist are
considered. To include slack at all edges in the histogram, use the
"-all_edges" option. This option results in increased memory consumption.

Use the "-setup", "-hold", "-recovery", or "-removal" options to
specify which kind of analysis should be performed. If none is
specified, setup analysis is used by default.

Reports can be directed to the Tcl console ("-stdout", default), a
file ("-file"), the Timing Analyzer graphical interface ("-panel_name"), or
any combination of the three.

The range of reported slack values can be controlled by specifying the
"-min_slack" and "-max_slack" options.  The number of bins (histogram
bars) can also be specified using the "-num_bins" option.

Use the "-partition" to show more information about each partition.
A path is in a partition if it's starting point is in the partition.
This option only works for "-panel".
```

Example Usage

```
project_open top
create_timing_netlist
read_sdc
update_timing_netlist

# Create a slack histogram for clk1, defaulting to
# the name "Slack Histogram (clk1)"
create_slack_histogram -clock_name clk1

# Create a slack histogram for clk2 named "MyHistogram"
create_slack_histogram -clock_name clk2 -panel_name MyHistogram

delete_timing_netlist
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Clock node not found or specified. Check valid clocks
TCL_ERROR	1	ERROR: Value <*string*> of option <*string*> is an invalid slack. Specify a valid slack value.
TCL_ERROR	1	ERROR: The max_slack value is less than or equal to the min_slack value. Specify a max_slack value that is greater than the min_slack value.
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Number of bins is 0. Specify a number greater than 0.

3.1.37.5. create_timing_netlist (::quartus::sta)

The following table displays information for the `create_timing_netlist` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`create_timing_netlist [-h | -help] [-long_help] [-force_dat] [-grade <c|i|m|e|a> ] [-model <fast|slow> ] [-no_latch] [-post_map] [-post_syn] [-snapshot <snapshot> ] [-speed <speed> ] [-temperature <value_in_C> ] [-voltage <value_in_mV> ] [-zero_ic_delays] [ <operating_conditions> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-force_dat`	Option to force delay annotation
	`-grade <c|i|m|e|a>`	Option to specify temperature grade
	`-model <fast|slow>`	Option to specify timing model
	`-no_latch`	Option to disable the analysis of latches as synchronous elements
	`-post_map`	Option to perform timing analysis on the post-synthesis netlist
	`-post_syn`	Option to perform timing analysis on the post-synthesis netlist
	`-snapshot <snapshot>`	Snapshot of the design to load
	`-speed <speed>`	Speed grade
	`-temperature <value_in_C>`	Operating temperature
	`-voltage <value_in_mV>`	Operating voltage
	`-zero_ic_delays`	Option to set all IC delays to zero
	<operating_conditions>	Operating conditions Tcl object name string
Description		

```
Creates the timing netlist by annotating the atom
netlist with delay information using post-fitting
results.

Use the -post_syn option to obtain post-synthesis
results.

In Quartus Prime Pro edition, you can use the -snapshot option
to specify which netlist you want to perform timing analysis on.

The create_timing_netlist command skips delay
annotation by default.  Use -force_dat to rerun delay
annotation. This is required if any delay annotation
setting is changed in the Quartus Prime project revision
(e.g. OUTPUT_PIN_LOAD).

Use "-model fast" to run the analysis using the fast
corner delay models first.  The -temperature,
-voltage, and -speed, options are also available. See
help for set_operating_conditions for details on these
options.

You can use model, temperature and voltage options to
specify operating conditions while creating timing
netlist (temperature and voltage options are not
supported by all families). You can also set operating
conditions by passing an operating conditions object
name as a positional argument to create_timing_netlist
command. After the timing netlist has been created,
you can use set_operating_conditions command to change
timing models without deleting and re-creating the
```

continued...

```
timing netlist.

Use the -grade option to analyze the design at a
different temperature grade. This option is provided
to support what-if analysis and is not recommended for
final sign-off analysis.

Use the -no_latch option to analyze latches as
combinational loops instead of synchronous elements.

Use the -zero_ic_delays option to set all IC delays in
the netlist to zero.
```

Example Usage

```
project_open my_top

# Create timing netlist before calling
# any report functions
create_timing_netlist

# Read SDC and update timing
read_sdc
update_timing_netlist

# Ready to call report functions
report_timing -npaths 1 -clock_setup

# The following command is optional
delete_timing_netlist

project_close

project_open my_top

# Report worst case period for -9 speed grade
create_timing_netlist -speed 9

# Read SDC and update timing
read_sdc
update_timing_netlist

report_timing -clock_setup -clock_filter clk
delete_timing_netlist

# Report hold violation for fastest corner
# Use set_operating_conditions instead
create_timing_netlist -model fast

# Read SDC and update timing
read_sdc
update_timing_netlist

report_timing -clock_hold -clock_filter clk
delete_timing_netlist

# If Delay Annotation has been run for the fast corner
# Force Delay Annotation
create_timing_netlist -model fast -force_dat

# Read SDC and update timing
read_sdc
update_timing_netlist

report_timing -clock_hold -clock_filter clk
delete_timing_netlist

# Report worst case period for post-technology mapping netlist
create_timing_netlist -post_map

# Read SDC and update timing
read_sdc
update_timing_netlist

report_timing -clock_setup -clock_filter clk
delete_timing_netlist

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

continued...

TCL_ERROR	1	ERROR: The TCL command <*string*> is not supported in quartus_fit. It is only supported in quartus_sta. Please use quartus_sta instead of quartus_fit if you want to use this TCL command.
TCL_ERROR	1	ERROR: Can't create timing netlist for device family <*string*>. Run Analysis and Synthesis (quartus_syn) using this device family as a value for the --family option before running the Timing Analyzer (create_timing_netlist).
TCL_ERROR	1	ERROR: The Fast Forward snapshot cannot be used for sign-off timing analysis. Please see the Fast Forward Timing Closure Recommendations report for performance estimates.
TCL_ERROR	1	ERROR: Can't run the Timing Analyzer (quartus_sta) -- Periphery placement (quartus_fit --plan) failed or was not run. Run I/O Assignment Analysis (quartus_fit --plan) successfully before running the Timing Analyzer (create_timing_netlist -post_map).
TCL_ERROR	1	ERROR: Invalid snapshot <*string*>. Available snapshot(s): <*string*>
TCL_ERROR	1	ERROR: Both the -temperature and -voltage options and their values are required.
TCL_ERROR	1	ERROR: Can't find active revision. Make sure there is an open, active revision name.
TCL_ERROR	1	ERROR: Could not find the <*string*> timing netlist on disk. Please run this fitter stage first, and make sure you have enabled Optimize Timing in Advanced Fitter Settings.
TCL_ERROR	1	ERROR: Option -no_latch is not supported for the current family.
TCL_ERROR	1	ERROR: Values entered did not match any valid operating conditions. Available operating conditions are: <*string*>
TCL_ERROR	1	ERROR: The <*string*> device family cannot perform automatic multi-corner timing analysis because the family does not support the set_operating_conditions command.
TCL_ERROR	1	ERROR: <*string*> Device family is not supported by the Timing Analyzer.
TCL_ERROR	1	ERROR: Illegal value: <*string*>. Specify an integer ranging from -999999999 to 999999999 for the option -voltage
TCL_ERROR	1	ERROR: The design has not been fully routed. If you want to perform Timing Analysis on an earlier netlist please choose which snapshot to load for analysis. Available snapshot(s): <*string*>
TCL_ERROR	1	ERROR: The design has not been fully routed and retimed to optimize timing. If you want to perform Timing Analysis on an earlier netlist please choose which snapshot to load for analysis. Available snapshot(s): <*string*>
TCL_ERROR	1	ERROR: Can't create timing netlist with -post_map option for device family <*string*>. Create timing netlist without the -post_map OPTION
TCL_ERROR	1	ERROR: Can't create timing netlist with -post_map option for device family <*string*>. Run I/O Assignment Analysis (quartus_fit --plan) successfully before running the Timing Analyzer (create_timing_netlist -post_map).

continued...

	TCL_ERROR	1	ERROR: Use of --post_map and --post_fpp options are not allowed in hierarchical mode. You can use --snapshot to specify the database to be used for analysis
	TCL_ERROR	1	ERROR: Can't run the Timing Analyzer (quartus_sta) -- Fitter (quartus_fit) failed or was not run. Run the Fitter (quartus_fit) successfully before running the Timing Analyzer (create_timing_netlist).
	TCL_ERROR	1	ERROR: Can't run the Timing Analyzer (quartus_sta) -- Partition Merge (quartus_cdb --merge) failed or was not run. Run the Partition Merge (quartus_cdb --merge) successfully before running the Timing Analyzer (create_timing_netlist -post_map).
	TCL_ERROR	1	ERROR: Can't run the Timing Analyzer (quartus_sta) -- Analysis and Synthesis (quartus_syn) failed or was not run. Run Analysis and Synthesis (quartus_syn) successfully before running the Timing Analyzer (create_timing_netlist -post_map).
	TCL_ERROR	1	ERROR: Delay annotation not run. Run delay annotation before running the Timing Analyzer (quartus_sta).
	TCL_ERROR	1	ERROR: You can only specify the -snapshot option when Hierarchical Design is enabled.
	TCL_ERROR	1	ERROR: Pre-fit timing analysis is not supported for the specified operating conditions. Please use the <*string*>.
	TCL_ERROR	1	ERROR: The Synthesis snapshot cannot currently be used for timing analysis.
	TCL_ERROR	1	ERROR: Timing netlist already exists. Delete the timing netlist before running this command.
	TCL_ERROR	1	ERROR: Unsupported option: <*string*>.

3.1.37.6. create_timing_summary (::quartus::sta)

The following table displays information for the `create_timing_summary` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`create_timing_summary [-h | -help] [-long_help] [-append] [-data_delay] [-file <name> ] [-hold] [-mpw] [-panel_name <name> ] [-recovery] [-removal] [-setup] [-split_by_corner] [-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-data_delay`	Data Delay Analysis (only applicable for setup and recovery analysis)
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-hold`	Hold Analysis
	`-mpw`	Minimum Pulse Width Analysis

continued...

	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-recovery`	Recovery Analysis
	`-removal`	Removal Analysis
	`-setup`	Setup Analysis (Default)
	`-split_by_corner`	When set, running this command with the -panel option creates a folder containing versions of this report for selected multiple operating conditions. This option has no effect when used with the -stdout or -file options.
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.

Description

```
Reports the worst-case Clock Setup and Clock Hold
slacks and endpoint TNS (total negative slack) per
clock domain. Total negative slack is the sum of all
slacks less than zero for either destination registers
or ports in the clock domain. The number of endpoints
in the domain with negative slack is also shown.

This command shows the worst-case slack for each clock
domain. You right click in these reports to run more
detailed reports like Histograms and Report Timing.

By default, this command creates a Setup Summary. This
command can also generate a Hold Summary (-hold),
Recovery Summary (-recovery), Removal Summary
(-removal), or Minimum Pulse Width Summary (-mpw).

The report can be directed to the Tcl console (-stdout,
default), a file (-file), the Timing Analyzer graphical
interface (-panel_name), or any combination of the
three.
```

Example Usage

```
project_open my_project

# Always create the netlist first and process constraints
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# Create Clock Domain Summary
create_timing_summary -panel_name "Setup Summary"
create_timing_summary -hold -panel_name "Hold Summary"

# The following command is optional
delete_timing_netlist

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Panel name cannot contain repeated pipe characters. (||).
TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Report database is not open

3.1.37.7. delete_sta_collection (::quartus::sta)

The following table displays information for the `delete_sta_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`delete_sta_collection [-h | -help] [-long_help] <collection>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<collection>	The collection to delete	
Description	This function is deprecated since sta cleans up the memory once the collection is out of scope. If you want to force a collection to go out of scope, use built-in tcl command 'unset'. Otherwise this function can remove the collection in cache.		
Example Usage	`project_open chiptrip` `create_timing_netlist` `set nodes [get_nodes Reg*]` `# ...` `# do some work with the $nodes collection` `# ...` `# Delete the collection.` `delete_sta_collection $nodes` `# ...` `# do more work` `# ...` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Cannot find specified collection. Specify an existing collection.

3.1.37.8. delete_timing_netlist (::quartus::sta)

The following table displays information for the `delete_timing_netlist` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`delete_timing_netlist [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Use this command to delete a timing netlist previously created using create_timing_netlist. This should be done at the end of a script or before calling create_timing_netlist again using different options or after recompiling the design. Use the set_operating_conditions command instead of	

continued...

	delete_timing_netlist and create_timing_netlist to change timing models. This avoids the cost of deleting and re-creating the timing netlist, and also preserves current timing assignments.		
Example Usage	This command currently contains no example usage.		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.9. enable_ccpp_removal (::quartus::sta)

The following table displays information for the `enable_ccpp_removal` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`enable_ccpp_removal [-h | -help] [-long_help] [-depth <depth> ] [-off] [-on]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-depth <depth>`	maximum clock tree depth for cppp removal	
	`-off`	Disable this setting.	
	`-on`	Enable this setting.	
Description	Enables (or disables) common clock path pessimism (CCPP) removal during slack computation. CCPP removal can improve timing results by removing min/max delay differences from common portions of clock paths. Enabling CCPP removal increases the time required to perform timing analysis. When specified, the optional depth parameter limits the clock tree depth used for CCPP removal. This is generally not applicable to FPGA compilations where the clock tree is fixed, but for large designs with potentially deep synthesized clock trees this can reduce outlier run time. When not specified, or when specified with a value of 0, the complete clock tree is used for CCPP removal (i.e. full clock-tree depth).		
Example Usage	`project_open top` `create_timing_netlist` `read_sdc` `# Report timing without CCPP removal` `report_timing` `# Enable CCPP removal and re-report timing.` `enable_ccpp_removal` `report_timing` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.10. enable_sdc_extension_collections (::quartus::sta)

The following table displays information for the `enable_sdc_extension_collections` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`enable_sdc_extension_collections [-h | -help] [-long_help] [-off] [-on]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-off`	Disable this setting.	
	`-on`	Enable this setting.	
Description	`Enable the support of SDC extension collections, such as keeper, register and node collections. When enable_sdc_extension_collections is not used, using these collections causes an error. Default to -on option.`		
Example Usage	`project_open top` `enable_sdc_extension_collections -on` `create_timing_netlist` `read_sdc` `report_timing -to_clock clk1` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Found an SDC extension Tcl collection creation command: <*string*>. Either switch to an SDC Tcl collection creation command or turn on enable_sdc_extension_collections.
	TCL_ERROR	1	ERROR: Found an SDC extension collection type: <*string*>. Either switch to an SDC collection type or turn on enable_sdc_extension_collections.
	TCL_ERROR	1	ERROR: Found timing netlist in memory. Delete timing netlist before use this command.

3.1.37.11. get_available_operating_conditions (::quartus::sta)

The following table displays information for the `get_available_operating_conditions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_available_operating_conditions [-h | -help] [-long_help] [-all] [-compile]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all`	Returns all available operating conditions
	`-compile`	Returns only the operating conditions that are critical to analyze in the inner loop compilation

continued...

Description	Returns a Tcl collection of available operating conditions for the current device. The Tcl collection contains the most extreme operating conditions within a user-specified junction temperature range. Use the -all option to return all available operating conditions.		
Example Usage	`#do report timing for different operating conditions` `foreach_in_collection op [get_available_operating_conditions] {` `  set_operating_conditions $op` `  update_timing_netlist` `  report_timing` `}` `#see detailed information about operating conditions` `foreach_in_collection op [get_available_operating_conditions] {` `  puts "Delay Model: [get_operating_conditions_info $op -model]"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: VID mode is not supported by part <*string*>.

3.1.37.12. get_cell_info (::quartus::sta)

The following table displays information for the `get_cell_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_cell_info [-h | -help] [-long_help] [-buried_nodes] [-buried_regs] [-in_pin_names] [-in_pins] [-location] [-name] [-out_pin_names] [-out_pins] [-pin_names] [-pins] [-type] [-wysiwyg_type]` <*cell_object*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-buried_nodes`	Return a collection of buried node IDs
	`-buried_regs`	Return a collection of buried register IDs
	`-in_pin_names`	Return a list of input pin names
	`-in_pins`	Return a collection of input pin IDs
	`-location`	Return the atom location in device
	`-name`	Return the cell name
	`-out_pin_names`	Return a list of output pin names
	`-out_pins`	Return a collection of output pin IDs
	`-pin_names`	Return a list of input and output pin names
	`-pins`	Return a collection of input and output pin IDs
	`-type`	Return the cell type
	`-wysiwyg_type`	Return the WYSIWYG type of the cell
	<*cell_object*>	Cell object

continued...

Description	Gets information about the specified cell (referenced by cell ID). You can obtain cell using the get_cells Tcl command. The "-type" option returns "cell". Options "-name", "-type", "-pin_name", "-in_pin_names", "-out_pin_names", "-pins", "-clock_pins", "-in_pins", "-out_pins", "-buried_nodes", "-buried_regs", "-location", and "-wysiwyg_type" are mutually exclusive.
Example Usage	`project_open chiptrip` `create_timing_netlist` `set cells [get_cells]` `foreach_in_collection cell $cells {` `  puts "[get_cell_info $cell -name]: [get_cell_info $cell -type]"` `}` `delete_timing_netlist` `project_close`

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Options are mutually exclusive: <*string*>. Specify only one of the these options.
TCL_ERROR	1	ERROR: Object with ID <*string*> is not an object of type <*string*>. Specify the ID of an object with the correct type.
TCL_ERROR	1	ERROR: Cannot find object of ID <*string*>. Specify an existing object ID.
TCL_ERROR	1	ERROR: Unsupported object type: <*string*>. Specify a supported object type.

3.1.37.13. get_clock_domain_info (::quartus::sta)

The following table displays information for the `get_clock_domain_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_clock_domain_info [-h | -help] [-long_help] [-data_delay] [-edge_slack] [-hold] [-mpw] [-recovery] [-removal] [-setup]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-data_delay`	Data Delay Analysis (only applicable for setup and recovery analysis)
	`-edge_slack`	Compute edge TNS (this option may significantly increase memory consumption)
	`-hold`	Hold Analysis
	`-mpw`	Minimum Pulse Width Analysis
	`-recovery`	Recovery Analysis
	`-removal`	Removal Analysis
	`-setup`	Setup Analysis (Default)
Description	Similar to create_timing_summary, the get_clock_domain_info command returns a Tcl list of	

continued...

	information about each active clock domain: the clock name, worst-case slack, endpoint TNS, number of endpoints in the domain with negative slack, and the timing model for which the worst-case slack is most critical. If the edge_slack option is specified, an extra entry for the edge TNS will be present, placed after the endpoint TNS. Computing the edge TNS may result in a significant increase in memory consumption. TNS is total negative slack, and it is the sum of all slacks less than zero for either destination registers or ports in the clock domain (endpoint TNS) or for all edges affecting the clock domain (edge TNS). By default, this command creates a Setup Summary. This command can also generate a Hold Summary (-hold), Recovery Summary (-recovery), Removal Summary (-removal), or Minimum Pulse Width Summary (-mpw).
Example Usage	`project_open my_project` `# Always create the netlist first` `create_timing_netlist` `read_sdc my_project.sdc` `update_timing_netlist` `# Get domain summary object` `set domain_list [get_clock_domain_info -setup]` `foreach domain $domain_list {` `  set name [lindex $domain 0]` `  set slack [lindex $domain 1]` `  set keeper_tns [lindex $domain 2]` `  puts "Clock $name : Slack = $slack , TNS = $keeper_tns"` `}` `# The following command is optional` `delete_timing_netlist` `project_close`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.14. get_clock_fmax_info (::quartus::sta)

The following table displays information for the `get_clock_fmax_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_clock_fmax_info [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Reports potential Fmax for every clock in the design, regardless of the user-specified clock periods. Fmax is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, Fmax is computed as if the rising and falling edges of the clock are scaled along with fmax, such that the duty cycle (in terms of a percentage) is maintained. Restricted Fmax considers hold timing in addition to setup timing, as well as minimum pulse and minimum period restrictions. Similar to unrestricted Fmax, the restricted Fmax is computed as if the rising	

continued...

	and falling edges of the clock are scaled along with Fmax, such that the duty cycle (in terms of a percentage) is maintained. Refer to hold timing reports (e.g., report_timing with the -hold option) or minimum pulse width reports (report_min_pulse_width) for details about specific paths, registers, or ports. This command is similar to report_clock_fmax_summary, except that it returns the results as a Tcl list for use in Tcl scripts. Each entry in the list represents one clock domain. Each entry is a Tcl list of the clock name, fmax (MHz), and restricted Fmax (MHz).

Example Usage

```
project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# Get domain summary object
set domain_list [get_clock_fmax_info]
foreach domain $domain_list {
    set name [lindex $domain 0]
    set fmax [lindex $domain 1]
    set restricted_fmax [lindex $domain 2]

    puts "Clock $name : Fmax = $fmax (Restricted Fmax = $restricted_fmax)"
}

# The following command is optional
delete_timing_netlist

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.15. get_clock_info (::quartus::sta)

The following table displays information for the `get_clock_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_clock_info [-h | -help] [-long_help] [-child_clocks] [-divide_by] [-duty_cycle] [-edge_shifts] [-edges] [-fall] [-is_inverted] [-latency] [-master_clock] [-master_clock_pin] [-max] [-min] [-multiply_by] [-name] [-nreg_neg] [-nreg_pos] [-offset] [-period] [-phase] [-rise] [-targets] [-type] [-waveform] <clk_object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-child_clocks`	Returns a list of child clock names
	`-divide_by`	Return the frequency divider (to the base clock)
	`-duty_cycle`	Return the duty cycle
	`-edge_shifts`	Return a list of edge shifts that the specified edges are to undergo to yield the final generated clock waveform
	`-edges`	Return a list of integer representing edges from the source clock that are to form edges of the generated clock
	`-fall`	Return clock fall latency
	`-is_inverted`	Return a boolean value to indicate if the generated clock is inverted

continued...

	-latency	Return clock latency
	-master_clock	Return the master clock name
	-master_clock_pin	Return the master clock source pin
	-max	Return max clock latency
	-min	Return min clock latency
	-multiply_by	Return the frequency multiplier (to the base clock)
	-name	Return the clock name
	-nreg_neg	Return number of registers negatively clocked by clock
	-nreg_pos	Return number of registers positively clocked by clock
	-offset	Return the clock offset
	-period	Return the clock period
	-phase	Return the clock phase
	-rise	Return clock rise latency
	-targets	Return the clock targets collection
	-type	Return the clock type
	-waveform	Return the waveform (rise time and fall time)
	<clk_object>	Clock object

Description

```
Returns information about the specified clock (referenced by clock
ID). Clock IDs can be obtained by Tcl commands such as get_clocks.

The "-type" option returns one of "base", "virtual_base", "generated",
"virtual_generated".

Options "-name", "-type", "-period", "-duty_cycle", "-waveform",
"-edges", "-edge_shifts", "-multiply_by", "-divide_by",
"-is_inverted", "-latency", "-master_clock", and "-targets" are
mutually exclusive. The "-latency" option requires a specified "-max"
or "-min" option as well as a "-rise" or "-fall" option.
```

Example Usage

```
project_open chiptrip
create_timing_netlist
set clocks [get_clocks]
foreach_in_collection clk $clocks {
    puts "[get_clock_info $clk -name]: [get_clock_info $clk -period]"
}
delete_timing_netlist
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Options are mutually exclusive: *<string>*. Specify only one of the these options.
TCL_ERROR	1	ERROR: Object with ID *<string>* is not an object of type *<string>*. Specify the ID of an object with the correct type.
TCL_ERROR	1	ERROR: Cannot find object of ID *<string>*. Specify an existing object ID.
TCL_ERROR	1	ERROR: Unsupported object type: *<string>*. Specify a supported object type.

3.1.37.16. get_clock_pair_info (::quartus::sta)

The following table displays information for the `get_clock_pair_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_clock_pair_info [-h | -help] [-long_help] [-fall_from <clk_object> ] [-fall_to <clk_object> ] [-false_path] [-from <clk_object> ] [-hierarchy] [-hold] [-rise_from <clk_object> ] [-rise_to <clk_object> ] [-setup] [-to <clk_object> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-fall_from <clk_object>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to <clk_object>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-false_path`	Return a description of the satisfied false-path-type assignment applied between the "from" and "to" clocks, if any
	`-from <clk_object>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-hierarchy`	Return a description the hierarchical relationship between the "from" and "to" clocks
	`-hold`	Return the hold analysis information if you use the "-false_path" option
	`-rise_from <clk_object>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to <clk_object>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Return the setup analysis information if you use the "-false_path" option. The setup analysis relationship is returned by default
	`-to <clk_object>`	Valid destination clocks (string patterns are matched using Tcl string matching)
Description		

```
The get_clock_pair_info command returns various clock information between two given clocks.

If you specify the "-false_path" option, the command returns a description of the satisfied
false path assignment between the "from" and "to" clocks, which includes clock groups.

If you specify the "-hierarchy" option, the command returns a description of the clock hierarchy
relationship between the two clocks, such as whether the "from" clock is a parent of the "to"
clock.

Use the "-from" option to specify the source clock that you want to query, and use the "-to"
option
to specify the destination clock that you want to query.

When using the "-false_path" option, you can use the "-rise_from" option to specify a source
clock's
rising edge to report on, or use the "-fall_from" option to specify a source clock's falling
edge to
report on. Likewise, you can use the "-rise_to" option to specify a destination clock's rising
edge,
or use the "-fall_to" option to specify a destination clock's falling edge. You can also specify
to
retrieve either the setup or hold false path relationship using the "-setup" or "-hold" option
respectively. By default, the setup relationship is reported.
```

continued...

Example Usage	<pre>create_clock clkA -period 10 create_generated_clock clkB -source clkA -divide_by 2 create_generated_clock clkC -source clkB -divide_by 2 set_false_path -from clkA -to clkB -latency_insensitive set_clock_groups -group clkA -group clkC -asynchronous get_clock_pair_info -from clkA -to clkB -false_path -> "false_path_latency_insensitive" get_clock_pair_info -from clkA -to clkC -false_path -> "clock_group_asynchronous" get_clock_pair_info -from clkA -to clkB -hierarchy -> "parent_child" get_clock_pair_info -from clkC -to clkA -hierarchy -> "descendant_ancestor"</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.17. get_datasheet (::quartus::sta)

The following table displays information for the `get_datasheet` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_datasheet [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	<pre>This function returns a tcl collection which contains the datasheet report. Its format is as follows: { { tsu { <tsu rise time> <tsu fall time> <input port> <clock port> <clock edge> <clock reference> } } { th { <th rise time> <th fall time> <input port> <clock port> <clock edge> <clock reference> } } { tco { <tco rise time> <tco fall time> <output port> <clock port> <clock edge> <clock reference> } } { mintco { <mintco rise time> <mintco fall time> <output port> <clock port> <clock edge> <clock reference> } } { tpd { <tpd rise-rise time> <tpd rise-fall time> <tpd fall-rise time> <tpd fall-fall time></pre>	

continued...

```
      <input port>
      <output port>
    }
  }

  { mintpd
    { <mintpd rise-rise time>
      <mintpd rise-fall time>
      <mintpd fall-rise time>
      <mintpd fall-fall time>
      <input port>
      <output port>
    }
  }

  { tzx
    { <tzx rise time>
      <tzx fall time>
      <output port>
      <clock port>
      <clock edge>
      <clock reference>
    }
  }

  { mintzx
    { <mintzx rise time>
      <mintzx fall time>
      <output port>
      <clock port>
      <clock edge>
      <clock reference>
    }
  }

  { txz
    { <tlz time>
      <thz time>
      <output port>
      <clock port>
      <clock edge>
      <clock reference>
    }
  }

  { mintxz
    { <mintlz time>
      <minthz time>
      <output port>
      <clock port>
      <clock edge>
      <clock reference>
    }
  }
}

There are no options for this command, and the data returned
is the same as from the report_datasheet command.
```

Example Usage

```
project_open proj1
create_timing_netlist
read_sdc
update_timing_netlist

# get the datasheet collection
set datasheet [get_datasheet]

# loop through contents of datasheet collection
foreach i $datasheet {
    foreach j $i {
        foreach k $j {
            #
            # extract individual items or
            # manipulate as necessary
            #
        }
    }
}
```

Return Value	Code Name	Code	String Return

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.18. get_default_sdc_file_names (::quartus::sta)

The following table displays information for the `get_default_sdc_file_names` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`get_default_sdc_file_names [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Returns the default SDC file name(s) used by the Quartus Prime Compiler when doing timing-driven optimizations.` `Returns the value for the QSF variable SDC_FILE. If multiple assignments are found, return them as a list If not specified, return <revision_name>.sdc.`		
Example Usage	`project_new test` `create_timing_netlist` `foreach file [get_default_sdc_file_names] {` `  read_sdc $file` `}` `update_timing_netlist` `report_timing` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.19. get_edge_info (::quartus::sta)

The following table displays information for the `get_edge_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`get_edge_info [-h | -help] [-long_help] [-delay] [-delay_type] [-dst] [-ff] [-fr] [-hslp] [-is_disabled] [-max] [-min] [-name] [-rf] [-rr] [-src] [-type] [-unateness] <edge_object>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-delay`	Return the delay.	
	`-delay_type`	Return the type of the delay (ic/cell/loop/user).	
	`-dst`	Return the destination node ID.	
	`-ff`	Return the fall-to-fall delay	

continued...

	-fr	Return the fall-to-rise delay
	-hslp	Return the HS/LP setting
	-is_disabled	Return whether the edge has been disabled, so should not be traversed through
	-max	Max delay
	-min	Min delay
	-name	Return the edge name
	-rf	Return the rise-to-fall delay
	-rr	Return the rise-to-rise delay
	-src	Return the source node ID
	-type	Return the edge type.
	-unateness	Return the unateness.
	<edge_object>	Edge object

Description

```
Returns information about the specified edge (referenced by edge ID).
Edge IDs can be obtained by Tcl commands such as get_node_info <node_id>
-synch_edges.

The "-type" and "-name" options exist only to keep the interface compliant
with the get_object_info command. The "-type" option returns specific edge
type as "synchronous", "asynchronous", "clock", or "combinational",
while the "-name" option always returns an empty string.

The "-delay" option returns the delay associated to the edge. Use
-max, -min and -rr, -rf, -fr, -ff options to specify the type of returned
delay. One of the -max, -min options must be specified. One of the
-rr, -rf, -fr, -ff options must be specified.

The -hslp option returns the HS/LP setting associated to the edge.

The -unateness option returns the unateness associated to the edge.

The -is_disabled option returns 1 if the edge should not be traversed
through, and 0 if the edge can be traversed through. Disabled edges include
edges in SCC loops and edges that the user has manually cut with the
set_disable_timing command.
```

Example Usage

```
project_open chiptrip
create_timing_netlist
set nodes [get_pins]
foreach_in_collection node $nodes {
    set node_name [get_node_info -name $node]
    set edges [get_node_info $node -fanout_edges]
    foreach edge $edges {
        # Traverse to the fanout node
        set dst_node [get_edge_info -dst $edge]
        set dst_name [get_node_info -name $dst_node]
        set delay_type [get_edge_info -delay_type $edge]

        set rr_delay [get_edge_info $edge -max -delay -rr]
        set rf_delay [get_edge_info $edge -max -delay -rf]
        set fr_delay [get_edge_info $edge -max -delay -fr]
        set ff_delay [get_edge_info $edge -max -delay -ff]
        puts "Max $delay_type delay of edge $edge, from $node_name to $dst_name: (RR:$rr_delay
RF:$rf_delay FR:$fr_delay FF:$ff_delay)"
    }
}
delete_timing_netlist
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Option *<string>* is not allowed to be specified with option -*<string>*. Remove the disallowed option.

continued...

	TCL_ERROR	1	ERROR: Options <*string*> are exclusively allowed to be specified with option -<*string*>. Specify one of the allowed options.
	TCL_ERROR	1	ERROR: Options are mutually exclusive: <*string*>. Specify only one of the these options.
	TCL_ERROR	1	ERROR: Object with ID <*string*> is not an object of type <*string*>. Specify the ID of an object with the correct type.
	TCL_ERROR	1	ERROR: Cannot find object of ID <*string*>. Specify an existing object ID.
	TCL_ERROR	1	ERROR: Option <*string*> is required to be specified with option -<*string*>. Specify a required option.
	TCL_ERROR	1	ERROR: Unsupported object type: <*string*>. Specify a supported object type.

3.1.37.20. get_entity_instances (::quartus::sta)

The following table displays information for the `get_entity_instances` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`get_entity_instances [-h | -help] [-long_help] [-nowarn] <entity_name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-nowarn`		Do not issue warning messages about missing entities
	<*entity_name*>		entity name
Description	`Returns a tcl list of all hierarchical instance paths to a named given entity/module.` `This can be useful in SDC files which need to be applied to all instances of a module automatically.`		
Example Usage	`get_entity_instances entity_name`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.21. get_min_pulse_width (::quartus::sta)

The following table displays information for the `get_min_pulse_width` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`get_min_pulse_width [-h | -help] [-long_help] [-nworst <number> ] [-type <all|min_period|clock_pulse> ] [ <targets> ]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-nworst <number>`		Specifies the number of pulse width checks to report (default=1)

continued...

	`-type` *<all|min_period|clock_pulse>*	Option to determine the minimum pulse width analysis type
	<targets>	Registers or ports
Description		

```
This command returns a Tcl list which contains the minimum
pulse width report. Its format is as follows:

{
  { <slack>,
    <actual width>,
    <required width>,
    <pulse>,
    <clock>,
    <clock edge>,
    <target>
  }
}

Refer to help for the report_min_pulse_width command for help on the
-nworst and -targets options.
```

Example Usage

```
get_min_pulse_width
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.22. get_net_info (::quartus::sta)

The following table displays information for the `get_net_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_net_info [-h | -help] [-long_help] [-name] [-pin] [-type]` *<net_object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name`	Return the net name
	`-pin`	Return the pin ID of this net
	`-type`	Return the net type.
	<net_object>	Net object
Description		

```
Returns information about the specified net (referenced by net ID).
Net ID's can be obtained by Tcl commands such as get_nets.

The "-type" option returns "net".

The options "-name", "-type", and "-pin" are mutually exclusive.
```

Example Usage

```
project_open chiptrip
create_timing_netlist

set nets [get_nets]
foreach_in_collection net $nets {
    puts [get_net_info $net -name]
}

delete_timing_netlist
project_close
```

Return Value	**Code Name**	**Code**	**String Return**

continued...

TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Option <*string*> is not allowed to be specified with option -<*string*>. Remove the disallowed option.
TCL_ERROR	1	ERROR: Options <*string*> are exclusively allowed to be specified with option -<*string*>. Specify one of the allowed options.
TCL_ERROR	1	ERROR: Options are mutually exclusive: <*string*>. Specify only one of the these options.
TCL_ERROR	1	ERROR: Object with ID <*string*> is not an object of type <*string*>. Specify the ID of an object with the correct type.
TCL_ERROR	1	ERROR: Cannot find object of ID <*string*>. Specify an existing object ID.
TCL_ERROR	1	ERROR: Option <*string*> is required to be specified with option -<*string*>. Specify a required option.
TCL_ERROR	1	ERROR: Unsupported object type: <*string*>. Specify a supported object type.

3.1.37.23. get_node_info (::quartus::sta)

The following table displays information for the `get_node_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_node_info [-h | -help] [-long_help] [-asynch_edges] [-cell] [-clock_edges] [-fanout_asynch_edges] [-fanout_clock_edges] [-fanout_edges] [-fanout_synch_edges] [-location] [-name] [-synch_edges] [-type]` <*node_object*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-asynch_edges`	Return a list of asynchronous edge IDs
	`-cell`	Return the host cell
	`-clock_edges`	Return a list of clock edge IDs
	`-fanout_asynch_edges`	Return a list of asynchronous fanout edge IDs
	`-fanout_clock_edges`	Return a list of clock fanout edge IDs
	`-fanout_edges`	Return a list of fanout edge IDs
	`-fanout_synch_edges`	Return a list of synchronous fanout edge IDs
	`-location`	Return the atom location in device
	`-name`	Return the node name
	`-synch_edges`	Return a list of synchronous edge IDs
	`-type`	Return the node type
	<*node_object*>	Node object
Description		

```
Gets information about the specified node (referenced by node ID).
Use Tcl commands such as get_nodes to obtain node IDs. The -type
option returns "reg", "port", "pin", "net", or "comb". The -name, -type,
```

continued...

	`-clock_edges, -synch_edges, -asynch_edges, -fanout_edges, -fanout_clock_edges, -fanout_synch_edges, -fanout_asynch_edges, -cell and -location options are mutually exclusive.`
Example Usage	`project_open chiptrip` `create_timing_netlist` `set registers [get_registers]` `foreach_in_collection reg $registers {` `  puts "[get_node_info $reg -name]: [get_node_info $reg -type]"` `}` `delete_timing_netlist` `project_close`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Object with ID <*string*> is not an object of type <*string*>. Specify the ID of an object with the correct type.
	TCL_ERROR	1	ERROR: Cannot find object of ID <*string*>. Specify an existing object ID.
	TCL_ERROR	1	ERROR: Unsupported object type: <*string*>. Specify a supported object type.

3.1.37.24. get_object_info (::quartus::sta)

The following table displays information for the `get_object_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_object_info [-h | -help] [-long_help] [-name] [-type] <object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-name`	Return the object name
	`-type`	Return the object type
	<*object*>	Object
Description	`Gets information about the specified object (referenced by object ID). Object IDs can be obtained by Tcl commands such as get_clocks, get_ports, get_cells, and others. The -type option returns "clk", "reg", "port", "cell", "pin", "comb", "net", or "edge". The -name and -type options are mutually exclusive.`	
Example Usage	`project_open chiptrip` `create_timing_netlist` `set ports [get_ports]` `foreach_in_collection port $ports {` `  puts [get_object_info $port -name]` `}` `delete_timing_netlist` `project_close`	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Argument <*string*> is a collection ID that does not link to any collection. Specify a legal collection ID.
	TCL_ERROR	1	ERROR: Argument <*string*> is an object ID that does not link to any object. Specify a valid object ID.

continued...

TCL_ERROR	1	ERROR: Argument <*string*> is an empty collection. Specify one that is a non-empty collection.
TCL_ERROR	1	ERROR: Argument <*string*> is not a valid object. Specify a valid object.
TCL_ERROR	1	ERROR: Argument <*string*> gives an empty collection. Specify one that gives a non-empty collection.
TCL_ERROR	1	ERROR: Argument <*string*> gives a collection with more than one object. Specify one that gives a collection with one object.
TCL_ERROR	1	ERROR: Argument <*string*> gives a collection that is not of <*string*> type. Specify one that gives a collection of required type.
TCL_ERROR	1	ERROR: Argument <*string*> gives an object that is not of <*string*> type. Specify one that gives an object of required type.
TCL_ERROR	1	ERROR: Argument <*string*> is a collection with more than one object. Specify a collection with one object.
TCL_ERROR	1	ERROR: Argument <*string*> is not a collection ID. Specify a legal collection ID.
TCL_ERROR	1	ERROR: Argument <*string*> is not an object ID. Specify a valid object ID.
TCL_ERROR	1	ERROR: Argument <*string*> is an object filter that matches more than one object. Specify a filter that matches only one object.
TCL_ERROR	1	ERROR: Argument <*string*> is an object filter that matches no objects. Specify one matches only one object.
TCL_ERROR	1	ERROR: Object with ID <*string*> is not an object of type <*string*>. Specify the ID of an object with the correct type.
TCL_ERROR	1	ERROR: Cannot find object of ID <*string*>. Specify an existing object ID.
TCL_ERROR	1	ERROR: Unsupported object type: <*string*>. Specify a supported object type.
TCL_ERROR	1	ERROR: Argument <*string*> is a collection that is not of <*string*> type. Specify a collection of required type.
TCL_ERROR	1	ERROR: Argument <*string*> is an object that is not of <*string*> type. Specify an object of required type.
TCL_ERROR	1	ERROR: Argument <*string*> is not <*string*> <*string*>. Specify an argument of the correct type.

3.1.37.25. get_operating_conditions (::quartus::sta)

The following table displays information for the `get_operating_conditions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_operating_conditions [-h | -help] [-long_help] [-for_analysis]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	-for_analysis		Get the set of analysis corners instead of the set of reporting corners
Description	Returns a list of the current operating conditions Tcl objects.		
Example Usage	puts "Delay Model : [get_operating_conditions_info [get_operating_conditions] -model]"		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.26. get_operating_conditions_info (::quartus::sta)

The following table displays information for the `get_operating_conditions_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`get_operating_conditions_info [-h | -help] [-long_help] [-display_name] [-grade] [-is_hold_only] [-model] [-name] [-speed] [-temperature] [-voltage]` *<operating_condition>*		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-display_name`		Returns the operating conditions display name
	`-grade`		Returns the temperature grade of the current device
	`-is_hold_only`		Returns whether the operating conditions only support short-path analysis
	`-model`		Returns the operating corner
	`-name`		Returns the operating conditions Tcl_Obj name
	`-speed`		Returns the speed grade of the current device
	`-temperature`		Returns the operating temperature
	`-voltage`		Returns the operating voltage
	<operating_condition>		Operating condition object
Description	Returns information about the operating_conditions Tcl object.		
Example Usage	`#see detailed information about operating conditions` `foreach_in_collection op [get_available_operating_conditions] {` `  puts "Delay Model: [get_operating_conditions_info $op -model]"` `}`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.27. get_partition_info (::quartus::sta)

The following table displays information for the `get_partition_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`get_partition_info [-h | -help] [-long_help] [-child] [-name] [-parent] [-type] <partition_object>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-child`	Return child partition name(s)	
	`-name`	Return the partition name	
	`-parent`	Return parent partition name	
	`-type`	Return the partition type	
	<partition_object>	Partition object	
Description	`Gets information about the specified partition (referenced by partition ID). Partition ID's can be obtained by Tcl commands such as get_partitions.` `The -name, -type, -parent, and -child options are mutually exclusive.`		
Example Usage	`project_open chiptrip` `create_timing_netlist` `set partitions [get_partitions *]` `foreach_in_collection partition $partitions {` `  puts "[get_partition_info $partition -name]"` `}` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Object with ID *<string>* is not an object of type *<string>*. Specify the ID of an object with the correct type.
	TCL_ERROR	1	ERROR: Cannot find object of ID *<string>*. Specify an existing object ID.
	TCL_ERROR	1	ERROR: Unsupported object type: *<string>*. Specify a supported object type.

3.1.37.28. get_path (::quartus::sta)

The following table displays information for the `get_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_path [-h | -help] [-long_help] [-fall_from <names>] [-fall_through <names>] [-fall_to <names>] [-from <names>] [-logic_depth] [-min_path] [-npaths <number>] [-nworst <number>] [-pairs_only] [-rise_from <names>] [-rise_through <names>] [-rise_to <names>] [-show_routing] [-through <names>] [-to <names>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

`-fall_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
`-fall_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
`-logic_depth`	Option to display the logic depth instead of path delay
`-min_path`	Find the minimum delay path(s)
`-npaths <number>`	Specifies the number of paths to report. The default value is 1 or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported (be wary that this may be slow)
`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same value as nworst
`-pairs_only`	When set, paths with the same start and end points are considered equivalent. Only the longest delay path for each unique combination is displayed.
`-rise_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
`-rise_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
`-show_routing`	Option to display detailed routing in the path
`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)

Description

```
Returns a collection of path objects for the longest delay paths
between arbitrary points in the netlist.

This command behaves the same as the report_path command.
However, instead of reporting the paths, it returns a Tcl
collection of path objects. You can retrieve path object data
using the get_path_info and get_point_info commands.

Note that get_path_info does not provide any clock-related
information, required points, or meaningful slack values, for
paths represented by the path objects returned by this function.

For help on the options shared with report_path, see help for the
report_path command.
```

Example Usage

```
# Define a few helper procedures to print out points
# on a path, and the path itself

proc print_point { point } {
    set total     [ get_point_info $point -total    ]
    set incr      [ get_point_info $point -incr     ]
    set node_id   [ get_point_info $point -node     ]
    set type      [ get_point_info $point -type     ]
    set rf        [ get_point_info $point -rise_fall]
    set node_name ""

    if { $node_id ne "" } {
```

continued...

```
        set node_name [ get_node_info $node_id -name ]
    }

    puts [format "%10s %8s %2s %-6s %s" $total $incr $rf $type $node_name ]
}

proc print_path { path } {
    puts "Delay      : [ get_path_info $path -arrival_time]"
    puts ""
    puts [format "%10s %8s %-2s %-6s %s" "Total" "Incr" "RF" "Type" "Name"]
    puts "================================================================="

    foreach_in_collection pt [ get_path_info $path -arrival_points ] {
        print_point $pt
    }
}

project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# And now simply iterate over the 10 longest delay paths,
# printing each as we go.
foreach_in_collection path [get_path -nworst 10] {
    print_path $path
    puts ""
}

delete_timing_netlist
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.
	TCL_ERROR	1	ERROR: Collection type '<*string*>' is not a valid type for a through collection. Valid collection types are 'pin' and 'net'
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.29. get_path_info (::quartus::sta)

The following table displays information for the `get_path_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_path_info [-h | -help] [-long_help] [-advanced] [-arrival_points] [-arrival_time] [-borrow_dst] [-borrow_src] [-clock_relationship] [-clock_skew] [-corner] [-data_delay] [-from] [-from_clock] [-from_clock_is_inverted] [-hold_end_multicycle] [-hold_start_multicycle] [-latch_time] [-launch_time] [-num_logic_levels] [-operating_conditions] [-required_points] [-required_time] [-setup_end_multicycle] [-setup_start_multicycle] [-slack] [-to] [-to_clock] [-to_clock_is_inverted] [-type] <*path_ref*>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-advanced`	Return clock pessimism due to advanced effects
	`-arrival_points`	Return a collection of point objects for the arrival path
	`-arrival_time`	Return the data arrival time for the path

continued...

	-borrow_dst	Return the time borrowed at destination (when the path ends at a level-sensitive latch)
	-borrow_src	Return the time borrowed at source (when the path starts at a level-sensitive latch)
	-clock_relationship	Return the clock relationship for the path
	-clock_skew	Return the clock skew for the path
	-corner	Returns a string indicating the operating conditions at which the path's delay values are found
	-data_delay	Return the data delay for the path
	-from	Return the source node ID
	-from_clock	Return the source clock ID
	-from_clock_is_inverted	Return 1 if the source clock is inverted, 0 otherwise
	-hold_end_multicycle	Return the hold end multicycle for the path
	-hold_start_multicycle	Return the hold start multicycle for the path
	-latch_time	Return the latch time for the path
	-launch_time	Return the launch time for the path
	-num_logic_levels	Return the number of logic levels on the path between the to node and from node
	-operating_conditions	Returns a string indicating the operating conditions at which the path's delay values are found
	-required_points	Return a collection of point objects for the required path
	-required_time	Return the data required time for the path
	-setup_end_multicycle	Return the setup end multicycle for the path
	-setup_start_multicycle	Return the setup start multicycle for the path
	-slack	Return the slack for the path
	-to	Return the destination node ID
	-to_clock	Return the destination clock ID
	-to_clock_is_inverted	Return 1 if the destination clock is inverted, 0 otherwise
	-type	Return the type of this path. Possible return values are: "setup", "hold", "recovery", "removal", "max_path", "min_path"
	<path_ref>	Path object
Description		

```
Returns information about the referenced timing path
object.

You can generate references to path objects with the
get_timing_paths and get_path functions.

The -type option returns one of the following types:
"setup", "hold", "recovery", "removal", "max_path", or
"min_path".

The -from and -to options return the ID of the nodes
at the start and end, respecitvely, of the arrival
path. If there is no node, an empty string is
returned. The -from node remains the same, regardless of
the level of clock detail provided. It is always the
first node clocked by the -from clock in the data
```

continued...

```
arrival path. You can use the node ID with the
get_node_info function to obtain additional
information about the node.

The -from_clock and -to_clock options return the ID of
the launching and latching clocks, respectively. If
there is no clock, an empty string is returned. You
can obtain additional information about the clocks
using the get_clock_info function.

Path objects generated by get_path do not have clock
information, required points, or meaningful slack
values.

The -arrival_points and -required_points options
return a collection of point objects for the arrival
and required paths, respectively. By iterating over
the collection, and using the get_point_info function,
you can obtain specific details about each portion of
the path.

If a path was created with additional clock detail,
the elements of the clock path are included in
each collection of points.

The values for the -from, -to, and other options are
either collections or a Tcl list of wildcards used to
create collections of appropriate types. The values
used must follow standard Tcl or Timing Analyzer-extension
substitution rules. See the help for
use_timing_analyzer_style_escaping for details.

When the path starts at a level-sensitive latch, the
-borrow_src option may be used to get the time borrowed
at the source. Similarly, when the path ends at a level-sensitive
latch, -borrow_dst may be used to get the time borrowed at
the destination. When these options are used with anything
other than level-sensitive latches, zero is returned.

For level-sensitive latches, when you use the -launch_time
or -latch_time options, the times reported do not include time borrowed.

The operating condition corresponding to all of a path's delay and time
values can be found using the -corner option.
```

Example Usage

```
# Define a few helper procedures to print out points
# on a path, and the path itself
proc get_clock_string { path clk } {
    set clk_str ""
    set clk_id [ get_path_info $path -${clk}_clock ]

    if { $clk_id ne "" } {
        set clk_str [ get_clock_info $clk_id -name ]

        if { [ get_path_info $path -${clk}_clock_is_inverted ] } {
            append clk_str " (INVERTED)"
        }
    }

    return $clk_str
}

proc print_point { point } {
    set total     [ get_point_info $point -total    ]
    set incr      [ get_point_info $point -incr     ]
    set node_id   [ get_point_info $point -node     ]
    set type      [ get_point_info $point -type     ]
    set rf        [ get_point_info $point -rise_fall]
    set node_name ""

    if { $node_id ne "" } {
        set node_name [ get_node_info $node_id -name ]
    }

    puts [format "%10s %8s %2s %-6s %s" $total $incr $rf $type $node_name ]
}

proc print_path { path } {
    puts "Slack      : [ get_path_info $path -slack]"
    puts "To Clock   : [ get_clock_string $path to ]"
    puts "From Clock : [ get_clock_string $path from]"
    puts ""
    puts [format "%10s %8s %-2s %-6s %s" "Total" "Incr" "RF" "Type" "Name"]
    puts "======================================================================="

    foreach_in_collection pt [ get_path_info $path -arrival_points ] {
```

continued...

```
        print_point $pt
    }
}

project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# And now simply iterate over the 10 worst setup paths, printing each path
foreach_in_collection path [ get_timing_paths -npaths 10 -setup ] {
    print_path $path
    puts ""
}

delete_timing_netlist
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Object with ID <*string*> is not an object of type <*string*>. Specify the ID of an object with the correct type.

3.1.37.30. get_pin_info (::quartus::sta)

The following table displays information for the `get_pin_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_pin_info [-h | -help] [-long_help] [-is_clock_pin] [-is_in_pin] [-is_out_pin] [-name] [-net] [-parent_cell] [-suffix] [-type] <pin_object>`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-is_clock_pin`	Return true if it is a clock pin, or false otherwise
	`-is_in_pin`	Return true if it is an input pin, or false otherwise
	`-is_out_pin`	Return true if it is an output pin, or false otherwise
	`-name`	Return the pin name
	`-net`	Return the net ID if this is an output pin
	`-parent_cell`	Return the parent cell ID
	`-suffix`	Return the suffix of the pin
	`-type`	Return the pin type
	<*pin_object*>	Pin object

Description

```
Gets information about the specified pin (referenced by pin ID).  Pin
ID's can be obtained by Tcl commands such as get_pins.

The -type option returns "pin".

Options -name, -type, -parent_cell, -net, -suffix, -is_clock_pin,
-is_in_pin and -is_out_pin are mutually exclusive.
```

Example Usage

```
project_open chiptrip
create_timing_netlist
set pins [get_pins]
foreach_in_collection pin $pins {
    set pin_name [get_pin_info $pin -name]
```

continued...

```
    set parent_cell [get_pin_info $pin -parent_cell]
    puts "Pin $pin_name belongs to cell [get_cell_info -name $parent_cell]"
}
delete_timing_netlist
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Options are mutually exclusive: <*string*>. Specify only one of the these options.
	TCL_ERROR	1	ERROR: Object with ID <*string*> is not an object of type <*string*>. Specify the ID of an object with the correct type.
	TCL_ERROR	1	ERROR: Cannot find object of ID <*string*>. Specify an existing object ID.
	TCL_ERROR	1	ERROR: Unsupported object type: <*string*>. Specify a supported object type.

3.1.37.31. get_point_info (::quartus::sta)

The following table displays information for the `get_point_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_point_info [-h | -help] [-long_help] [-edge] [-incremental_delay] [-location] [-node] [-number_of_fanout] [-rise_fall] [-total_delay] [-type]` <*point_ref*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-edge`	Return the edge ID for the edge associated with this point. If the point has no edge, this returns an empty string
	`-incremental_delay`	Return the incremental delay through this point
	`-location`	Return a string indicating the location of the point's node, if there is one, else an empty string
	`-node`	Return the node ID for the node associated with this point. If the point has no node, this returns an empty string
	`-number_of_fanout`	Return the number of fanout that this point has in the netlist
	`-rise_fall`	Return a string indicating the rise_fall type of this point. Return values are r, f, rr, rf, fr, ff, or an empty string for undefined
	`-total_delay`	Return the total delay of the path at this point. This includes the incremental delay for the point itself
	`-type`	Return a string indicating the type of the point
	<*point_ref*>	Point object
Description		

```
Returns information about the referenced timing point object.
References to path objects can be generated using the get_path_info
function.

A point object is the equivalent of a row in a path in the output from
report_timing.

The -node option returns a node ID for the corrsponding node in the
```

continued...

```
path.  For points that do not have a corresponding node (such as
points for the lumped clock network delay, launch time, latch time,
individual routing elements, etc.), the node ID is an empty string.  A
non-empty node ID can be used in conjunction with the get_node_info
function to obtain additional information about the node.

The -edge option returns an edge ID for the corresponding edge in the
path.  Only points of type "ic", "cell", and "comp" may have edges.
For other point types, an empty string is returned.  A non-empty
edge ID can be used in conjunction with the get_edge_info function to
obtain additional information about the edge.

The -total_delay option returns the total delay along the path, up to
and including the current point.  The -incremental_delay option
returns the delay incurred by going through this point in the path.
Both delays are formated in terms of the current time units, excluding
the unit string.

The -number_of_fanout option returns the number of fanouts that the
corresponding node has in the timing netlist.  If there is no node for
this point, the return value is 0.

The -location option returns a string indicating the location of the
corresponding node in the part.  If there is no corresponding node,
this returns an empty string.

The -rise_fall option returns the transition type of this point.

Possible values for -rise_fall are:

    Value     Description
    -------   ------------------------------
    (empty)   Unknown transition
    r         Rising output
    f         Falling output
    rr        Rising input, rising output
    rf        Rising input, falling output
    fr        Falling input, rising output
    ff        Falling input, falling output

The -type option returns a string indicating the type of delay that
this point represents in the path.

Possible return values for -type are:

    Value     Description
    -------   ------------------------------------------------------
    borrow    Time borrowed (for level-sensitive latches)
    cell      Cell delay
    clknet    Lumped clock network delay
    clksrc    Clock source.  Used to ensure that the end-point of a
              clock segment is marked in the path when source latency
              is specified, or when the actual path cannot be found.
    comp      PLL clock network compensation delay
    ic        Interconnect delay
    iext      External input delay
    latch     Clock latch time
    launch    Clock launch time
    loop      Lumped combinational loop delay
    oext      External output delay
    re        Routing element (only for paths generated with the
              -show_routing option)
    srclat    Source latency for a clock segment. This appears
              if latency was specified between two clocks, or if a
              path could not be found between them.
    unc       Clock uncertainty
    utco      Register micro-Tco time
    utsu      Register micro-Tsu time
    uth       Register micro-Th time
```

Example Usage

```
# Define a few helper procedures to print out points
# on a path, and the path itself
proc get_clock_string { path clk } {
    set clk_str ""
    set clk_id [ get_path_info $path -${clk}_clock ]

    if { $clk_id ne "" } {
        set clk_str [ get_clock_info $clk_id -name ]

        if { [ get_path_info $path -${clk}_clock_is_inverted ] } {
            append clk_str " (INVERTED)"
        }
    }

    return $clk_str
}
```

continued...

```
proc print_point { point } {
    set total     [ get_point_info $point -total    ]
    set incr      [ get_point_info $point -incr     ]
    set node_id   [ get_point_info $point -node     ]
    set type      [ get_point_info $point -type     ]
    set rf        [ get_point_info $point -rise_fall]
    set node_name ""

    if { $node_id ne "" } {
        set node_name [ get_node_info $node_id -name ]
    }

    puts [format "%10s %8s %2s %-6s %s" $total $incr $rf $type $node_name ]
}

proc print_path { path } {
    puts "Slack      : [ get_path_info $path -slack]"
    puts "To Clock   : [ get_clock_string $path to ]"
    puts "From Clock : [ get_clock_string $path from]"
    puts ""
    puts [format "%10s %8s %-2s %-6s %s" "Total" "Incr" "RF" "Type" "Name"]
    puts "------------------------------------------------------------------"

    foreach_in_collection pt [ get_path_info $path -arrival_points ] {
        print_point $pt
    }
}

project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# And now simply iterate over the 10 worst setup paths, printing each path
foreach_in_collection path [ get_timing_paths -npaths 10 -setup ] {
    print_path $path
    puts ""
}

delete_timing_netlist
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Object with ID *<string>* is not an object of type *<string>*. Specify the ID of an object with the correct type.

3.1.37.32. get_port_info (::quartus::sta)

The following table displays information for the `get_port_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_port_info [-h | -help] [-long_help] [-edge_rate] [-is_inout_port] [-is_input_port] [-is_output_port] [-name] [-type]` *<port_object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-edge_rate`	Return the edge_rate value
	`-is_inout_port`	Return true if it is an inout port, or false otherwise
	`-is_input_port`	Return true if it is an input port, or false otherwise
	`-is_output_port`	Return true if it is an output port, or false otherwise
	`-name`	Return the port name
	`-type`	Return the port type

continued...

	<port_object>	Port object
Description		

```
Returns information about the specified port (referenced by port ID).
Port ID's can be obtained by Tcl commands such as get_ports. The -type
option returns "port". The -name, -type, -edge_rate, -is_input_port,
-is_output_port and is_inout_port options are mutually exclusive.
```

Example Usage

```
project_open chiptrip
create_timing_netlist
set ports [get_ports]
foreach_in_collection port $ports {
    set port_type ""
    if [get_port_info $port -is_inout_port] {
        set port_type "bidir"
    } elseif [get_port_info $port -is_input_port {
        set port_type "in"
    } else {
        set port_type "out"
    }
    puts "[get_port_info $port -name]: $port_type"
}
delete_timing_netlist
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Options are mutually exclusive: *<string>*. Specify only one of the these options.
TCL_ERROR	1	ERROR: Object with ID *<string>* is not an object of type *<string>*. Specify the ID of an object with the correct type.
TCL_ERROR	1	ERROR: Cannot find object of ID *<string>*. Specify an existing object ID.
TCL_ERROR	1	ERROR: Unsupported object type: *<string>*. Specify a supported object type.

3.1.37.33. get_register_info (::quartus::sta)

The following table displays information for the `get_register_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_register_info [-h | -help] [-long_help] [-asynch_edges] [-clock_edges] [-delay_type` *<max_rise|max_fall|min_rise|min_fall>* `] [-fanout_edges] [-is_latch] [-is_synchronizer] [-name] [-related_pin` *<related_pin_value>* `] [-synch_edges] [-tch] [-tcl] [-tco] [-th] [-tmin] [-tsu] [-type]` *<reg_object>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-asynch_edges`	Return a list of asynchronous edge IDs
	`-clock_edges`	Return a list of clock edge IDs
	`-delay_type` *<max_rise|max_fall|min_rise|min_fall>*	Specify which type of micro delay to query
	`-fanout_edges`	Return a list of fanout edge IDs
	`-is_latch`	Return "1" if this is a latch node, or "0" otherwise
	`-is_synchronizer`	Return which stage of a synchronizer chain this register is part of, or "0" if it is not part of a synchronizer chain

continued...

	`-name`	Return the object name
	`-related_pin` *<related_pin_value>*	Specify which register port you want the tsu/th/tco for
	`-synch_edges`	Return a list of synchronous edge IDs
	`-tch`	Return the Tch value
	`-tcl`	Return the Tcl value
	`-tco`	Return the Tco value
	`-th`	Return the Th value
	`-tmin`	Return the Tmin value
	`-tsu`	Return the Tsu value
	`-type`	Return the object type
	<reg_object>	Register object

Description

```
Gets information about the specified register (referenced by register
ID). Register IDs can be obtained by Tcl commands such as
get_registers.

The -type option returns "reg". The -name, -type, -tco, -tsu, -th,
-tch, -tcl, -tmin, -clock_edges, -synch_edges, -asynch_edges, -fanout_edges
and -is_latch options are mutually exclusive.
```

Example Usage

```
project_open chiptrip
create_timing_netlist
set registers [get_registers]
foreach_in_collection reg $registers {
    set name [get_register_info $reg -name]
    set tco [get_register_info $reg -tco]
    puts "Tco of $name is $tco"
}
delete_timing_netlist
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: delay_type is used without specifying the related_pin. These two options must be used together.
TCL_ERROR	1	ERROR: related_pin is used without specifying the delay_type. These two options must be used together.
TCL_ERROR	1	ERROR: Object with ID *<string>* is not an object of type *<string>*. Specify the ID of an object with the correct type.
TCL_ERROR	1	ERROR: Cannot find object of ID *<string>*. Specify an existing object ID.
TCL_ERROR	1	ERROR: Unsupported object type: *<string>*. Specify a supported object type.

3.1.37.34. get_timing_paths (::quartus::sta)

The following table displays information for the `get_timing_paths` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`get_timing_paths [-h | -help] [-long_help] [-asynch_clock] [-data_delay] [-detail <summary|path_only|path_and_clock|full_path>] [-fall_from <names>] [-fall_from_clock <names>] [-fall_through <names>] [-fall_to <names>] [-fall_to_clock <names>] [-false_path] [-from <names>] [-from_clock <names>] [-hold] [-inter_clock] [-intra_clock] [-less_than_slack <slack limit>] [-npaths <number>] [-nworst <number>] [-pairs_only] [-recovery] [-removal] [-rise_from <names>] [-rise_from_clock <names>] [-rise_through <names>] [-rise_to <names>] [-rise_to_clock <names>] [-setup] [-show_routing] [-through <names>] [-to <names>] [-to_clock <names>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-data_delay`	Report only paths that are covered by a data delay assignment
	`-detail <summary|path_only|path_and_clock|full_path>`	Option to determine how much detail should be shown in the path report
	`-fall_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-false_path`	Report only paths that are cut by a false path assignment
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-hold`	Option to report clock hold paths
	`-inter_clock`	Only report paths whose launch and latch clock are different
	`-intra_clock`	Only report paths whose launch and latch clock are the same
	`-less_than_slack <slack limit>`	Limit the paths reported to those with slack values less than the specified limit.
	`-npaths <number>`	Specifies the number of paths to report (default=1, or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported but be wary that this may be slow)

continued...

	`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same value as nworst
	`-pairs_only`	When set, paths with the same start and end points are considered equivalent. Only the worst case path for each unique combination is displayed.
	`-recovery`	Option to report recovery paths
	`-removal`	Option to report removal paths
	`-rise_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Option to report clock setup paths
	`-show_routing`	Option to display detailed routing in the path
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)

Description

```
Get a collection of path objects for the worst-case paths.

This command behaves the same as the report_timing command.
However, instead of reporting the paths, it returns a Tcl
collection of path objects. You can retrieve path object data
using the get_path_info and get_point_info commands.

For help on the options shared with report_timing, see the
report_timing help page.
```

Example Usage

```
# Define a few helper procedures to print out points
# on a path, and the path itself
proc get_clock_string { path clk } {
    set clk_str ""
    set clk_id [ get_path_info $path -${clk}_clock ]

    if { $clk_id ne "" } {
        set clk_str [ get_clock_info $clk_id -name ]

        if { [ get_path_info $path -${clk}_clock_is_inverted ] } {
            append clk_str " (INVERTED)"
        }
    }

    return $clk_str
}

proc print_point { point } {
    set total     [ get_point_info $point -total    ]
    set incr      [ get_point_info $point -incr     ]
    set node_id   [ get_point_info $point -node     ]
    set type      [ get_point_info $point -type     ]
    set rf        [ get_point_info $point -rise_fall]
    set node_name ""
```

continued...

```
    if { $node_id ne "" } {
        set node_name [ get_node_info $node_id -name ]
    }

    puts [format "%10s %8s %2s %-6s %s" $total $incr $rf $type $node_name ]
}

proc print_path { path } {
    puts "Slack      : [ get_path_info $path -slack]"
    puts "To Clock   : [ get_clock_string $path to ]"
    puts "From Clock : [ get_clock_string $path from]"
    puts ""
    puts [format "%10s %8s %-2s %-6s %s" "Total" "Incr" "RF" "Type" "Name"]
    puts "================================================================="

    foreach_in_collection pt [ get_path_info $path -arrival_points ] {
        print_point $pt
    }
}

project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# And now simply iterate over the 10 worst setup paths, printing each path
foreach_in_collection path [ get_timing_paths -npaths 10 -setup ] {
    print_path $path
    puts ""
}

delete_timing_netlist
project_close
```

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.
	TCL_ERROR	1	ERROR: Collection type '<*string*>' is not a valid type for a through collection. Valid collection types are 'pin' and 'net'
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.35. import_sdc (::quartus::sta)

The following table displays information for the `import_sdc` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`import_sdc [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	`Reads SDCs from synthesized database directly.`	
Example Usage	`project_new test` `create_timing_netlist` `# Read SDC commands` `import_sdc` `update_timing_netlist` `report_timing`	

continued...

	`` delete_timing_netlist project_close ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.36. is_post_syn_sta (::quartus::sta)

The following table displays information for the `is_post_syn_sta` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`is_post_syn_sta [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	``` Returns 1 when post synthesis delays are annotated on the timing netlist ```		
Example Usage	``` if {[is_post_syn_sta]} { create_clock ... } ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.37. locate (::quartus::sta)

The following table displays information for the `locate` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`locate [-h | -help] [-long_help] [-chip] [-classic_tmv] [-color` *<black|blue|brown|green|grey|light_grey|orange|purple|red|white>* `] [-dpp] [-label` *<label>* `] [-no_duplicates] [-rpe] [-rtm] [-tmv]` *<items>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-chip`	Locate the object in the Chip Planner
	`-classic_tmv`	Locate the object in the Classic Technology Map Viewer
	`-color` *<black|blue|brown|green|grey|light_grey|orange|purple|red|white>*	Specify the color to be used to identify the objects you are locating
	`-dpp`	Locate in the Design Partition Planner
	`-label` *<label>*	Specify a label used to identify the objects you are locating
	`-no_duplicates`	Do not locate duplicate objects

continued...

	-rpe	Locate in the Resource Property Editor
	-rtm	Locate in the Hyper-Retiming Viewer
	-tmv	Locate the object in the Technology Map Viewer
	<items>	Items to locate. Any collection or object (such as paths, points, nodes, nets, keepers, registers, etc) may be located by passing a reference to the corresponding collection or object.

Description

```
Locate an object from the Timing Analyzer in another Quartus Prime tool.

With this command, one or more objects, or collections of objects,
can be located in a supported Quartus tool from the Timing Analyzer.

The destination must be specified with one of the following
options:

    Option         Destination Tool
    ======         ============================
    -chip          Chip Planner
    -rpe           Resource Property Editor
    -rtl           RTL Viewer
    -classic_rtl   Classic RTL Viewer
    -tmv           Technology Map Viewer
    -classic_tmv   Classic Technology Map Viewer
    -rtm           Hyper-Retiming Viewer
    -dpp           Design Partition Planner

The -label option can be used to specify a label for the located
objects.  The -color command can be used to specify a color to
be used to identify the located objects in the destination tool.
```

Example Usage

```
proc prepare_design { } {
    set sleep_for 2000

    create_timing_netlist -risefall

    post_message -type info "Give the GUI some time to catch up to the new netlist. Sleep for
$sleep_for ms"
    after $sleep_for

    read_sdc
    update_timing_netlist
}

prepare_design

# Locate all of the nodes in the longest ten paths
# into the Resource Property Editor
locate [get_path -npaths 10] -rpe

# Locate ten paths into the chip planner, labelling
# each one individually.
set path_col [get_timing_paths -npaths 10]
set path_id 0

foreach_in_collection path $path_col {
    incr path_id

    locate -label "Path #$path_id" $path -chip
}

# locate all keepers that begin with the letter t
# to the Tech Map Viewer
locate [get_keepers t*] -tmv

# locate all nodes that begin with the letter a
#
# The Timing Analyzer GUI will prompt the user for the
# tool to which the nodes should be located.
#
# Pause first to allow the previous locations to
# appear, as the dialog that pops up, to ask
# the user for a location, will block the rest
# of the GUI until cleared.

after 5000

post_message -type info "Interactive locate"
locate a*
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Illegal color: <*string*>. Specify a color that is currently supported by the locate command.
	TCL_ERROR	1	ERROR: An object or collection matching <*string*> could not be found, or was of a type not supported by the locate command.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.38. print_total_sdc_processing_time (::quartus::sta)

The following table displays information for the `print_total_sdc_processing_time` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`print_total_sdc_processing_time [-h | -help] [-long_help]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
Description	`Returns the total processing time as a formatted string.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.39. query_collection (::quartus::sta)

The following table displays information for the `query_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`query_collection [-h | -help] [-long_help] [-all] [-limit` <*limit_value*> `] [-list_format] [-report_format]` <*collection*>	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-all`	Return all the collection objects.
	`-limit` <*limit_value*>	Set number of collection objects to return.
	`-list_format`	Return collection objects in a list format.
	`-report_format`	Return collection objects in a format of one element per line.
	<*collection*>	Object collection
Description	`Query collection objects.` `Collections can be obtained by Tcl commands such as get_clocks,`	

continued...

	get_ports, get_cells. If neither the -limit nor the -all option is specified, then first 20 objects (if the collection has more than 20 objects) or all objects (if the collection has less than or equal to 20 objects) are returned.		
Example Usage	project_open chiptrip create_timing_netlist set nodes [get_nodes Reg*] # Get the first 100 nodes in the collection. query_collection $nodes -limit 100 delete_timing_netlist project_close		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Cannot find specified collection. Specify an existing collection.

3.1.37.40. read_sdc (::quartus::sta)

The following table displays information for the `read_sdc` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`read_sdc [-h | -help] [-long_help] [-hdl] [-instance` *<instance_name>* `] [-no_import] [-no_sdc_promotion] [-post_fit] [-post_syn] [-project_relative] [` *<file_name>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-hdl`	Read SDC commands embedded in HDL
	`-instance` *<instance_name>*	Name of the block for which we are reading the SDC file
	`-no_import`	Do not automatically import RTL SDCs from the netlist
	`-no_sdc_promotion`	Disable instance-bound scoping of constraints
	`-post_fit`	For post_syn STA. Load SDCs as if in post_fit (Includes SDC_FILE, HDL Embedded, Entity)
	`-post_syn`	For use with STA on fitter snapshots. Load SDCs as if in post_syn mode
	`-project_relative`	If passing in a relative path to a file, interpret that path relative to the current project\'s directory instead of STA\'s current directory
	<file_name>	Name of the SDC file
Description	Reads an SDC file with all current constraints and exceptions. If an SDC file is specified, read_sdc only reads that SDC file. If the -hdl option is specified, read_sdc only reads SDC commands that were embedded in HDL. If no arguments are specified, read_sdc reads the default SDC files along with any SDC commands that were embedded in HDL. If one or more SDC_FILE assignments exists in the QSF, read_sdc reads all of them in order. Otherwise, read_sdc reads the file <revision>.sdc if it exists.	

continued...

Example Usage	<pre>project_new test create_timing_netlist # Read SDC commands from test_constraints.sdc read_sdc test_constraints.sdc # Read SDC commands embedded in HDL read_sdc -hdl update_timing_netlist report_timing delete_timing_netlist project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Can't find file <*string*>
	TCL_ERROR	1	ERROR: The provided instance name does not exist in the current design.
	TCL_ERROR	1	ERROR: instance_name must be specified when using no_sdc_promotion.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.41. register_delete_timing_netlist_callback (::quartus::sta)

The following table displays information for the `register_delete_timing_netlist_callback` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`register_delete_timing_netlist_callback [-h | -help] [-long_help] <body>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<*body*>	Body of the callback to run	
Description	<pre>Use this command to specify a TCL procedure that runs before the timing netlist is deleted. This command can be used to specify a procedure to run, like so: proc mycallback { # This is the body of the delete_timing_netlist callback. ... } register_delete_timing_netlist_callback mycallback Or, you may specify the procedure directly: register_delete_timing_netlist_callback { # This is the body of the delete_timing_netlist callback. ... }</pre>		
Example Usage	<pre>This command currently contains no example usage.</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.42. remove_from_collection (::quartus::sta)

The following table displays information for the `remove_from_collection` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`remove_from_collection [-h | -help] [-long_help] <base_collection> <items_to_remove>`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	<base_collection>	Collection to remove items from	
	<items_to_remove>	Items to be removed from base_collection	
Description	This command takes two collections and returns a new collection that is the difference of the two, effectively the second collection subtracted from the first collection. The second collection can be a string, but the first has to be previously-created collection: either by passing any of the "get_" functions directly, or by passing a variable that contains a collection (see code examples for this command). If a collection is used for the second argument, the types in the second collection must be the same as or a subset of the types in the first collection. If the first collection consists of keepers, the second collection can only consist of keepers, registers or ports. If the first collection consists of partitions, the second collection can only consist of partitions or cells. If the first collection consists of nodes, the second collection can only consist of nodes, keepers, registers, ports, pins, nets or combinational nodes.		
Example Usage	`set a_keeprs [get_keepers a*]` `set a1_regs [get_registers a1*]` `set keeprs_without_a1 [remove_from_collection $a_keeprs $a1_regs]` `#or:` `set keeprs_without_a1 [remove_from_collection $a_keeprs [get_registers a1*]]` `#or even:` `set keeprs_without_a1 [remove_from_collection $a_keeprs a1*]` `# - note that the last statement will actually remove all keepers with name a1*` `#  not only registers! (will remove IOs with name a1*, if any)` `# Get the first 100 nodes in the collection.` `query_collection $keeprs_without_a1 -limit 100`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Cannot find specified collection. Specify an existing collection.

3.1.37.43. report_advanced_io_timing (::quartus::sta)

The following table displays information for the `report_advanced_io_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_advanced_io_timing [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-stdout]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-append`		If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`		Sends the results to an ASCII or HTML file. Depending on the extension
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-stdout`		Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description	```This command creates a report containing all of the relevant signal integrity measurements computed during I/O buffer simulation. You must perform delay annotation with Advanced I/O Timing enabled before using this command. This option can be enabled from the Timing Analyzer Page of the Settings dialog box.```		
Example Usage	```project_open my_project # Always create the netlist first create_timing_netlist read_sdc my_project.sdc update_timing_netlist # Create "Advanced I/O Timing" report panel report_advanced_io_timing -panel_name "Advanced I/O Timing" # The following command is optional delete_timing_netlist project_close```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.44. report_asynch_cdc (::quartus::sta)

The following table displays information for the `report_asynch_cdc` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615
Syntax	`report_asynch_cdc [-h | -help] [-long_help] [-append] [-asynch_clock] [-detail <summary|full> ] [-fall_from_clock <names> ] [-fall_to_clock <names> ] [-file <name> ] [-from <names> ] [-from_clock <names> ] [-inter_clock] [-intra_clock]`

continued...

	`[-multi_bit_cdc] [-nentries <number> ] [-panel_name <name> ] [-reset_cdc] [-rise_from_clock <names> ] [-rise_to_clock <names> ] [-single_bit_cdc] [-stdout] [-to <names> ] [-to_clock <names> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-detail <summary|full>`	Option to specify how much detail should be shown in the CDC report
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-inter_clock`	Only report paths whose launch and latch clock are different
	`-intra_clock`	Only report paths whose launch and latch clock are the same
	`-multi_bit_cdc`	Report multi-bit CDC topologies found in the design
	`-nentries <number>`	Display up to this number of entries per CDC topology. Only applicable in full detail mode
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-reset_cdc`	Report reset CDC topologies found in the design
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-single_bit_cdc`	Report single-bit CDC topologies found in the design
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
Description		

```
This report displays all Clock-Domain-Crossings (CDC) between asynchronous
clocks in a design.

The report can be directed to the Tcl console ("-stdout", default), a file
```

continued...

	("-file"), the Timing Analyzer graphical user interface ("-panel_name"), or any combination of the three. By default, this command reports all CDC's in a design. You can limit the analysis performed by this command to specific CDC source and destination nodes, using the "-from" and "-to" options. The analysis can also be limited using clocks. Specify the CDC's source and destination clocks using the "-from_clock" and "-to_clock" options. Alternatively, specify edges of the clock using "-rise_from_clock", "-fall_from_clock", "-rise_to_clock", and "-fall_to_clock" options. To limit the report to only display specific categories of CDC's, use "-multi_bit_cdc" to report multi-bit CDC buses, "-single_bit_cdc" to report single-bit CDC synchronizers, and "-reset_cdc" to report asynchronous reset topologies. By default, all three CDC categories are reported. Use the "-nentries" option to limit the number of CDC's displayed per CDC topology type. The topology types fall into one of the three categories above. Run the report to see all the topology types that are supported. Use the "-detail" option to specify the desired level of reporting detail. "summary" generates a table listing only the number of CDC topologies recognized per category and the total number of crossings for that category. "full" reports every recognized CDC under each topology type, and is the default behaviour. In the full report, you can click on each individual CDC in the Timing Analyzer graphical user interface to view its statistics.
Example Usage	`# Report all bus CDC's, up to 10 buses per CDC topology type` `report_asynch_cdc -buses -nentries 10` `# Report all CDC's from clkA to clkB` `report_asynch_cdc -from_clock clkA -to_clock clkB` `# Report all synchronizer chains whose source node contains "transfer"` `report_asynch_cdc -from *transfer* -synch_chains`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: The current report cannot be run in XML mode. Use create_timing_netlist to create a non-XML timing netlist.

3.1.37.45. report_bottleneck (::quartus::sta)

The following table displays information for the `report_bottleneck` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_bottleneck [-h | -help] [-long_help] [-cmetric` *<cmetric_name>* `] [-details] [-metric` *<default|tns|num_paths|num_fpaths|num_fanins|num_fanouts>* `] [-nworst` *<number>* `] [-panel_name` *<panel_name>* `] [-stdout] [` *<paths>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-cmetric` *<cmetric_name>*	Custom metric function to evaluate individual nodes
	`-details`	Show the detailed information (number of failing edges, number of fanins, etc)
	`-metric` *<default|tns|num_paths|num_fpaths|num_fanins|num_fanouts>*	Indicate the metric to use to rate individual nodes
	`-nworst` *<number>*	Specifies the maximum number of nodes to report. If unspecified, there is no limit
	`-panel_name` *<panel_name>*	Sends the results to the panel and specifies the name of the new panel
	`-stdout`	Output the result onto stdout

continued...

	<paths>	Paths to be analyzed

Description

```
Reports bottleneck nodes in a design based on
user-specified criteria for rating each node.

The following considerations are pre-defined

    num_fpaths: (default) returns the number of paths
    that fail timing through the node.

    num_fanins: returns the number of fanin edges from
    the node.

    num_fanouts: returns the number of fanout edges from
    the node.

    num_paths: returns the number of paths through the node.

    tns: returns the total negative slack of all the paths
    through the node.

The paths to be analyzed can be specified by passing the result of any
get_timing_paths call as the last argument to report_bottleneck. If no
paths are specified, report_bottleneck analyzes the worst 1000 setup
paths in the design.

You can also create your own custom criteria for evaluating nodes
based on the combination of the number of fanouts, fanins, failing
paths, and total paths.

To use custom criteria, do the following:

    1. Create a Tcl procedure that takes one argument, "arg",
    for example.

    2. Use "upvar $arg metric" in the procedure.

    3. Calculate the rating based on $metric(tns),
    $metric(num_fanouts), $metric(num_fanins), and $metric(num_fpaths).

    4. Return the rating with "return $rating".

    5. Pass the name of your custom criteria procedure to
    report_bottleneck using the -cmetric option.

Reports can be directed to the Tcl console (-stdout), the Timing Analyzer
graphical interface (-panel), or a combination of the two.
```

Example Usage

```
project_open my_project
create_timing_netlist
read_sdc
update_timing_netlist

# use the worst 500 hold paths
set paths [ get_timing_paths -npaths 500 -hold ]
report_bottleneck -metric default -panel "Timing Analysis Bottleneck Report - Default Metric"
$paths
report_bottleneck -metric tns -panel "Timing Analysis Bottleneck Report - TNS" $paths
report_bottleneck -metric num_paths -panel "Timing Analysis Bottleneck Report - Number of
Paths" $paths
report_bottleneck -metric num_fpaths -panel "Timing Analysis Bottleneck Report - Number of
Failing Paths" $paths
report_bottleneck -metric num_fanouts -panel "Timing Analysis Bottleneck Report - Number of
Fanouts" $paths

# create custom metric and use the worst 2000 setup paths
proc report_bottleneck_custom_metric {arg} {
    # Description: use the number of fanins as the custom metric.
    upvar $arg metric
    set rating $metric(num_fanins)
    return $rating
}

set paths [ get_timing_paths -npaths 2000 -setup ]
report_bottleneck -cmetric report_bottleneck_custom_metric -panel "Timing Analysis Bottleneck
Report - Custom" $paths
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.37.46. report_cdc_viewer (::quartus::sta)

The following table displays information for the `report_cdc_viewer` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_cdc_viewer [-h | -help] [-long_help] [-append] [-clock_groups] [-file <name>] [-from_clock <names>] [-fully_cut] [-hierarchy] [-hold] [-inactive] [-less_than_slack <slack limit>] [-list] [-panel_name <name>] [-recovery] [-removal] [-setup] [-show_empty] [-show_non_crossing] [-stdout] [-summary] [-timed] [-to_clock <names>] [-tree]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-clock_groups`	Show transfers cut by clock groups
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fully_cut`	Include transfers where all paths are cut by false path assignments
	`-hierarchy`	When writing to a panel or a non-list file, show child clocks as nested within their parent clocks
	`-hold`	Option to report clock hold paths
	`-inactive`	Show transfers between inactive clocks
	`-less_than_slack <slack limit>`	Ignore paths with slack values greater or equal to the specified limit
	`-list`	When writing to file, output all clock transfers in a list instead of a grid
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-recovery`	Option to report recovery paths
	`-removal`	Option to report removal paths
	`-setup`	Option to report clock setup paths
	`-show_empty`	Include all clocks in the report, even ones that launch/latch no paths
	`-show_non_crossing`	Include transfers that do not cross a clock domain (i.e. transfers to/from the same clock)
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-summary`	Do not display slack information
	`-timed`	Include all timed transfers (i.e. those not fully cut by false paths or clock groups)

continued...

	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-tree`	Alias for the -hierarchy option
Description		

```
Generates a Clock Domain Crossing Viewer (CDC Viewer) report. It displays
all clock transfers (i.e., data paths between one clock domain and another
clock domain) in a design, as well as data on each transfer: the number of
uncut & cut paths in the transfer, the worst-case and total-negative
slack of the transfer, and the tightest setup/hold/removal/recovery
relationship between the clocks in the transfer.

The report indicates what clock transfers are cut ("false paths") by
set_clock_groups or clock-to-clock set_false_path commands, and which
clock transfers are ignored due to clocks marked as inactive by
set_active_clocks.

The report can be directed to the Tcl console ("-stdout", default), a
file ("-file"), the Timing Analyzer graphical user interface ("-panel_name"),
or any combination of the three. When directed to a panel or a file
without the -list option, the CDC Viewer report will be a grid, where
each cell represents paths transferring between a source and destination
clock. When directed to a panel or an HTML file, the grid is color-
coded to highlight passing, failing, cut, and inactive transfers, as
well as clock groups. When directed to the Tcl console or a file with
the -list option, a list of clock transfers are reported.

The -setup, -hold, -recovery, and -removal options determine the
analysis type of the report, particularly the reporting of false_paths
that apply to only one analysis type.  If you do not specify any of
these options, a report is generated for each analysis.

By default, the only transfers that are shown in the report are
ones that participate in clock domain crossings. This means that
transfers to/from the same clock are not shown, even if they fail
timing. To show all transfers, use the -show_all option.

The generated report can be customized by a variety of options. Refer
to the help text of those options for more information.
```

Example Usage

```
project_open top
create_timing_netlist -skip_dat
report_cdc_viewer -panel_name
delete_timing_netlist
project_close
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Netlist must be updated. Run update_timing_netlist

3.1.37.47. report_clock_fmax_summary (::quartus::sta)

The following table displays information for the `report_clock_fmax_summary` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_clock_fmax_summary [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-split_by_corner] [-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.

continued...

	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-split_by_corner`	When set, running this command with the -panel option creates a folder containing versions of this report for selected multiple operating conditions. This option has no effect when used with the -stdout or -file options.
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.

Description

```
Reports potential fmax for every clock in the design, regardless of
the user-specified clock periods. Fmax is only computed for paths
where the source and destination registers or ports are driven by the
same clock. Paths of different clocks, including generated clocks, are
ignored. For paths between a clock and its inversion, fmax is computed
as if the rising and falling edges of the clock are scaled along with
fmax, such that the duty cycle (in terms of a percentage) is
maintained.

Restricted fmax considers hold timing in addition to setup timing, as
well as minimum pulse and minimum period restrictions. Similar to
unrestricted fmax, the restricted fmax is computed as if the rising
and falling edges of the clock are scaled along with fmax, such that
the duty cycle (in terms of a percentage) is maintained. The "Note"
column reports which analyses restricted fmax.  Refer to hold timing
reports (e.g., report_timing with the -hold option) or minimum pulse
width reports generated by the report_min_pulse_width command for
details of specific paths, registers, or ports.
```

Example Usage

```
project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# Output results in the form of messages
report_clock_fmax_summary
# Create "Fmax" report panel
report_clock_fmax_summary -panel_name Fmax
# Report both with report panel and messages
report_clock_fmax_summary -panel_name Fmax -stdout

# The following command is optional
delete_timing_netlist

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.48. report_clock_network (::quartus::sta)

The following table displays information for the `report_clock_network` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615
Syntax	`report_clock_network [-h | -help] [-long_help] [-append] [-file <name> ] [-include_non_clock_paths] [-initial_depth <number> ] [-locate_with_routing] [-panel_name <name> ] [-show_full_paths] [-stdout] [-target <names> ]`

continued...

Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-include_non_clock_paths`	Show paths that are potentially in the clock network. These paths terminate at a register's clock pin but do not start at a clock source.
	`-initial_depth <number>`	Initial clock network depth to report.
	`-locate_with_routing`	When locating to a path, also show detailed routing. This is only shown in Chip Planner.
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-show_full_paths`	Show the full clock paths - do not reduce the table size by condensing multiple trivial nodes into one row.
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-target <names>`	Valid nodes in the clock network or clocks (string patterns are matched using Tcl string matching).
Description		

```
The clock network report shows the netlist topology of clock paths that make up the
clock network in a design. It allows the user to track a clock signal from its
source, through transformations such as PLL's, to the loads that the clock drives.
It also reveals clock relationships by indicating nodes where generated clocks are
defined.

The report can be directed to the Tcl console ("-stdout", default), a file
("-file"), the Timing Analyzer graphical user interface ("-panel_name"),
or any combination of the three.

Use the "-target" option to specify report targets. These can be clocks in the design or
nodes on the clock network, for example PLL outputs. If you specify clocks as targets, the
clock network report uses the clock's target nodes as report targets. The clock network
report then displays all nodes in the clock network that are in the fanin and fanout cones
of the report targets. If no report target is specified, all clock target nodes in the
design are used by default.

Each row in the report may include one node or multiple trivial nodes with singular fanin
and fanout edge. To disable the behaviour of collapsing down trivial nodes into one row, use
the "-show_full_paths" option.

When running the clock network report in the Timing Analyzer GUI, rows in the main table
corresponding to report target nodes are highlighted in light blue. You may click on each
row of the table to view information such as clock frequencies, relationships, and why
this node belongs in the clock network. You may right-click on any row to locate the
node in other Quartus tools such as RTL Viewer. You may also right-click on specific rows
to locate the shortest clock path from the clock source to that node in tools such as RTL
Viewer.

Use the "-initial_depth" option to reduce the height of the report table in the Timing Analyzer
GUI. Rows in the main table that are deeper than the set initial depth are collapsed by
default, but can be manually expanded. If this option is not set, the report determines an
appropriate initial depth to use.

Use the "-include_non_clock_paths" option if a node that you expect to be in the clock network
cannot be found. This option shows nodes that lead into a register's clock pin, but are not
downstream of any known clock source. Furthermore, registers that are not clocked by any clocks
will be shown using this option.

Use the "-locate_with_routing" option to show the routing elements that make up a path when
locating a path to Chip Planner. Only use this option if you use the "-target" option to specify
reporting targets, otherwise the report may be slow to generate.
```

continued...

Example Usage	<pre># Report the clock network that feeds into register regA report_clock_network -panel {Report} -target [get_registers regA] # Report the clock network starting from clk_100 clock, but show only the first 10 levels report_clock_network -panel {Report} -target [get_clocks clk_100] -initial_depth 10 # Report the clock network passing through the combinational node clk_mux|combout. # In the report, include paths that end at a register's clock pin but do not start # at a clock source. report_clock_network -panel {Report} -target [get_nodes clk_mux|combout] - include_non_clock_paths</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.49. report_clock_transfers (::quartus::sta)

The following table displays information for the `report_clock_transfers` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_clock_transfers [-h | -help] [-long_help] [-append] [-file <name> ] [-hold] [-panel_name <name> ] [-recovery] [-removal] [-setup] [-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-hold`	Creates a clock transfer summary for hold analysis
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-recovery`	Creates a clock transfer summary for recovery analysis
	`-removal`	Creates a clock transfer summary for removal analysis
	`-setup`	Creates a clock transfer summary for setup analysis
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description	<pre>Generates a timing report table showing all clock transfers (i.e., data paths between one clock domain and another clock domain). The from and to clocks are shown as well as the number of paths for each transfer: RR, RF, FR, FF. An RF transfer, for example, occurs when the source register of path is clocked by the rising edge of its clock and the destination register is clocked by the falling edge of its clock. The report also indicates what clock transfers are cut ("false paths") by set_clock_groups or clock-to-clock set_false_path commands. For transfers that are not cut, the number of paths reported does not take into account paths cut by path-specific set_false_path commands. Actual path counts may be lower than reported. The report can be directed to the Tcl console ("-stdout", default), a file ("-file"), the Timing Analyzer graphical user interface ("-panel_name"), or any combination of the three. The -setup, -hold, -recovery, and -removal options determine the</pre>	

continued...

	analysis type of the report, particularly the reporting of false_paths that apply to only one analysis type. If you do not specify any of these options, a report is generated for each analysis.
Example Usage	project_open top create_timing_netlist -skip_dat report_clock_transfers -panel_name delete_timing_netlist project_close

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Netlist must be updated. Run update_timing_netlist

3.1.37.50. report_clocks (::quartus::sta)

The following table displays information for the `report_clocks` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_clocks [-h | -help] [-long_help] [-append] [-desc] [-file <name> ] [-hierarchy] [-panel_name <name> ] [-stdout] [-summary] [-tree] [-waveform]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-desc`	Sort the clocks by name in descending order (ascending order is default)
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-hierarchy`	Display a tree view of the clocks
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-summary`	Create a single table with a summary of each clock
	`-tree`	alias for hierarchy
	`-waveform`	Display the clocks graphically as waveforms
Description	Report can be directed to the Tcl console ("-stdout", default), a file ("-file"), the Timing Analyzer graphical interface ("-panel_name"), or any combination of the three. For stdout/file output, the clock details are reported in two sections. The first section shows all clocks, their period, and their waveform. This includes generated clocks after an update_timing_netlist. The second section shows details for all generated clocks. For the panel report, both sections are combined into a single report.	

continued...

Example Usage	<pre>project_open top create_timing_netlist read_sdc update_timing_netlist report_clocks delete_timing_netlist project_close</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.51. report_datasheet (::quartus::sta)

The following table displays information for the `report_datasheet` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_datasheet [-h | -help] [-long_help] [-append] [-expand_bus] [-file` *<name>* `] [-panel_name` *<name>* `] [-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-expand_bus`	If set, bus is reported as individual ports
	`-file` *<name>*	Sends the results to an ASCII or HTML file. Depending on the extension
	`-panel_name` *<name>*	Sends the results to the panel and specifies the name of the new panel
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description	<pre>This function creates a datasheet report which summarizes the timing characteristics of the design as a whole. It reports setup (tsu), hold (th), clock-to-output (tco), minimum clock-to-output (mintco), output enable (tzx), minimum output enable (mintzx), output disable (txz), minimum output disable (mintxz), propagation delay (tpd), and minimum propagation delay (mintpd) times. These delays are reported for each clock or port for which they are relevant. If there is a case where there are multiple paths for a clock (for example if there are multiplexed clocks), then the maximum delay is reported for the tsu, th, tco, tzx, txz and tpd, and the minimum delay is reported for mintco, mintzx, mintxz and mintpd. The datasheet can be outputed to the Tcl console ("-stdout", default), a file ("-file"), or a report panel ("-panel_name"). Additionally if the "-file" option is used then the "-append" option can be used to specify that new data should be written to the end of the specified file.</pre>	
Example Usage	<pre>project_open proj1 create_timing_netlist read_sdc update_timing_netlist # Report the datasheet to a report panel report_datasheet -panel_name Datasheet # Report the datasheet to a file report_datasheet -file file1.txt</pre>	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.52. report_ddr (::quartus::sta)

The following table displays information for the `report_ddr` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_ddr [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-stdout]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-append`		If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`		Sends the results to an ASCII or HTML file. Depending on the extension
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-stdout`		Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description	`This command generates custom timing reports for EMIF instantiations.`		
Example Usage	`project_new test` `create_timing_netlist` `read_sdc` `update_timing_netlist` `report_ddr -panel "Report DDR"` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.53. report_exceptions (::quartus::sta)

The following table displays information for the `report_exceptions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_exceptions [-h | -help] [-long_help] [-append] [-asynch_clock] [-clock_groups] [-detail <summary|path_summary|path_only|path_and_clock|full_path>] [-fall_from_clock <names>] [-fall_to_clock <names>] [-false_path] [-file <name>] [-from <names>] [-from_clock <names>] [-hold] [-ignored] [-inter_clock] [-intra_clock] [-less_than_slack <slack limit>] [-max_delay] [-min_delay] [-multicycle_path] [-npaths <number>] [-num_exceptions <number>] [-nworst <number>] [-pairs_only] [-panel_name <name>] [-reachability] [-recovery] [-removal] [-report_clock_groups] [-rise_from_clock <names>] [-rise_to_clock <names>] [-setup] [-split_by_corner] [-stdout] [-through <names>] [-to <names>] [-to_clock <names>] [-valid]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-clock_groups`	Option to show clock groups in reports. This option also treats clock groups as timing exceptions
	`-detail <summary|path_summary|path_only|path_and_clock|full_path>`	Option to determine how much detail should be shown in the path report
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-false_path`	Option to report false path exceptions
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-hold`	Option to report clock hold paths
	`-ignored`	Option to report only exceptions that are partially or fully ignored
	`-inter_clock`	Only report paths whose launch and latch clock are different
	`-intra_clock`	Only report paths whose launch and latch clock are the same
	`-less_than_slack <slack limit>`	Limit the paths reported to those with slack values less than the specified limit.
	`-max_delay`	Option to report maximum delay exceptions
	`-min_delay`	Option to report mininum delay exceptions

continued...

	`-multicycle_path`	Option to report multicycle path exceptions
	`-npaths <number>`	Specifies the number of paths to report (default=1, or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported but be wary that this may be slow)
	`-num_exceptions <number>`	Option to only show a certain number of exceptions in the report
	`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same value as nworst
	`-pairs_only`	When set, paths with the same start and end points are considered equivalent. Only the worst case path for each unique combination is displayed.
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-reachability`	Option to report a percent value of how many nodes in an exception's targets are satisfied by the exception
	`-recovery`	Option to report recovery paths
	`-removal`	Option to report removal paths
	`-report_clock_groups`	Option to treat clock groups as exceptions
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Option to report clock setup paths
	`-split_by_corner`	When set, running this command with the -panel option creates a folder containing versions of this report for selected multiple operating conditions. This option has no effect when used with the -stdout or -file options.
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-valid`	Option to report only exceptions that cover valid paths and have been successfully applied
Description		

```
Reports status and timing analysis results for each timing
exception in your design. A timing exception is one of:
set_false_path, set_multicycle_path, set_min_delay, or
set_max_delay.

The status is relative to the paths covered by the -from, -to,
and other options. A complete timing exception relative to the
values specified for the -from and -to options may not actually
be complete with respect to the full design.

Complete: The exception has not been overridden and is valid
There are paths affected by this exception.
```

continued...

```
Partially overridden: The exception includes some paths that
have been overridden by one or more higher-precedence
exceptions.

Fully overridden: All paths affected by this exception have been
overridden by one or more higher-precedence exceptions.

Invalid: No paths are affected by this exception. This occurs
when a timing exception has valid -from, -to, or -through
collections, but there are no actual paths from the -from nodes
to the -to nodes.

Paths will not be analyzed: This exception has no paths for the
given analysis type (setup/hold/recovery/removal). This occurs when
a timing exception has paths, but only for analysis types other
than the type used for the current report.

Use the -valid option to show only exceptions that have the status
of "Complete" or "Partially overridden". These exceptions cover valid
paths and have been successfully applied.

Use the -ignored option to show only exceptions that are partially or
fully ignored. This includes exceptions with the status "Partially
Overridden", "Fully Overridden", "Invalid", as well as any exceptions
that have errors and were not included in the analysis.

Use the -setup (default), -hold, -recovery, or -removal options
to further filter the exceptions reported.

The report can be directed to the Tcl console using -stdout
(default), a file using -file, the Timing Analyzer graphical user
interface using -panel_name, or any combination of those three
options.

You can limit the reporting by this command to specific start
and end points, using the -from and -to options. You can further
limit reporting to clocks using the -from_clock and -to_clock
options, or to specific edges of the clock using the
-rise_from_clock, -fall_from_clock, -rise_to_clock, and
-fall_to_clock options. Additionally, the -through option can be
used to restrict reporting to paths which go through specified
pins or nets.

To determine which timing exceptions override other timing
exceptions, use the same -from and -to options that were used
with the overridden timing exception.

Use the -npaths option to limit the number of paths to report
per timing exception. If you do not specify this option, only
the single worst-case path per timing exception is provided. Use
the -less_than_slack option to limit output to all paths with
slack less than the specified value, up to the number specified
by -npaths.

Use the -nworst option to limit the number of paths reported for
each unique endpoint. If you do not specify this option, the
number of paths reported for each destination node is bounded
only by the -npaths option. If this option is used, but -npaths
is not specified, -npaths defaults to the same value specified
for -nworst.

Use the "-pairs_only" option to filter the output further, restricting
the results to only unique combinations of start and end points.

Use the "-num_exceptions" option to limit the report to only show
a certain number of exceptions. If the limit is not set or set too
high, the report may need to run for extended durations.

Use the "-reachability" option to determine a percentage of how many
start and endpoint pairs given by an exception's -from and -to filters
are satisfied by the exception. If an exception does not have a -from
target, the reachability ratio measures how many of the -to targets are
satisfied by the exception, and vise versa for exceptions without a -to
target. Exceptions without both -from and -to targets are not calculated
for reachability. An exception that targets clocks or uses wildcards that
match many nodes likely has a lower reachability than an exception that
targets specific nodes. To avoid overly-broad exception constraints,
you should use exceptions with a higher reachability value.

Reachability is calculated differently depending on whether the exception
is bus-type. A bus-type exception has both -from and -to options declared,
and they target the same number of nodes. As well, all nodes in the -from option
must be declared together, and all nodes in the -to option must be declared
together. Otherwise, the exception is non-bus type. Bus-type reachability
ratio assumes that each node in the -from target connects with one node in
the -to target. Non-bus type reachability ratio is typically more pessimistic
because it assumes that each node in the -from target connects with every
node in the -to target.
```

continued...

```
Use the -detail option to specify the desired level of report
detail. Specifiying "summary" generates a single table listing
only the highlights of each timing exception (status and
worst-case slack). Specifying "path_summary" generates a table,
per timing exception, listing only the highlights of each
path. Specifying "path_only" reports the path from the source to
the destination without any detail about the clock
path. Instead, the clock network delay is shown as a single
number. This is the default behavior. Specifying
"path_and_clock" extends the arrival and required paths back to
the launch and latch clocks. Specifying "full_path" causes
continued tracing back through generated clocks to the
underlying base clock.

To treat clock groups as timing exceptions (meaning that they
override exceptions with a lower priority), use the
"-report_clock_groups" option.

By default, all exception types are reported, and clock groups are
reported only if the "-report_clock_groups" option is used. Use a
combination of "-false_path", "-multicycle_path", "-max_delay",
"-min_delay", and "-clock_groups" to limit the analysis to specific
exception types. Note that "-clock_groups" option also treats clock
groups as timing exceptions.

False path exceptions (set_false_path) are reported as if the
false path was not applied, similar to the -false_path option
for report_timing.

The values of the "-from", "-to", and "-through" options are
either collections or a Tcl list of wildcards used to create
collections of appropriate types. The values used must follow
standard Tcl or Timing Analyzer-extension substitution rules. See the
help for use_timing_analyzer_style_escaping for details.

**************************
Clock-Domains-Crossing Verification:

To view the effects of clock groups on your design, run
"report_exceptions -report_clock_groups". If any pair of clocks
in your design are cut by a set_clock_groups command and NOT
cut by a set_false_path command, the Report shows paths between
that pair of clocks as a clock-to-clock exception.

Example:
set_clock_groups -logically_exclusive -group {clkA clkB} -group {clkC clkD}
report_exceptions -report_clock_groups -npaths 0 -detail path_only

You might want to run "report_exceptions -report_clock_groups" if:

1. You want to check which paths are cut by a set_clock_groups
SDC command.

2. You want to cut clock paths, but don't want to add
set_clock_groups because you're concerned that you might cut
a path incorrectly. With the "report_exceptions
-report_clock_groups" command, you can add set_clock_groups
to split clocks that are truly asynchronous, then add
set_false_path commands to manually cut what you want, and
then verify (in the Report) that no paths were found (since
false paths have priority). If a path is found by the report,
you can correct that without wasting a compilation concentrating
on some unrelated false paths.

3. If you use the set_clock_groups command and have some logic
that is behaving badly and think it's timing related, run the
"report_exceptions -report_clock_groups" command on that
hierarchy to verify whether the correct paths have been cut.
*******************************
```

Example Usage

```
# Reports all timing exceptions for a setup analysis.
report_exceptions

# Reports all timing exceptions for a hold analysis.
report_exceptions -hold

# Reports all timing exceptions affecting input paths for
# recovery analysis, reporting the 10 worst paths per exception.
report_exceptions -from [all_inputs] -to [all_registers] \
  -recovery -npaths 10 -detail full_path

# Reports all paths affected by timing exceptions, including
# all clock-to-clock-paths cut by clock groups.
report_exceptions -report_clock_groups -npaths 0 -detail path_only
```

continued...

	`` # Reports false path exceptions to determine which ones were overridden by clock groups report_exceptions -false_path -report_clock_groups -npaths 20 # Reports only clock groups, multicycle paths, and min delays report_exceptions -clock_groups -multicycle_path -min_delay # Generate a reachability report that shows 10 exceptions report_exceptions -reachability -num_exceptions 10 ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.54. report_ini_usage (::quartus::sta)

The following table displays information for the `report_ini_usage` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_ini_usage [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-stdout]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-append`		If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`		Sends the results to an ASCII or HTML file. Depending on the extension
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-stdout`		Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description	``` Reports the list of .ini files and variables used during timing analysis. ```		
Example Usage	``` project_open top create_timing_netlist read_sdc update_timing_netlist report_ini_usage -panel "INI Usage" ```		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.55. report_logic_depth (::quartus::sta)

The following table displays information for the `report_logic_depth` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_logic_depth [-h | -help] [-long_help] [-append] [-asynch_clock] [-detail <histogram|path>] [-fall_from <names>] [-fall_from_clock <names>] [-fall_through <names>] [-fall_to <names>] [-fall_to_clock <names>] [-file <name>] [-from <names>] [-from_clock <names>] [-hold] [-inter_clock] [-intra_clock] [-less_than_slack <slack limit>] [-npaths <number>] [-nworst <number>] [-pairs_only] [-panel_name <name>] [-recovery] [-removal] [-rise_from <names>] [-rise_from_clock <names>] [-rise_through <names>] [-rise_to <names>] [-rise_to_clock <names>] [-setup] [-stdout] [-through <names>] [-to <names>] [-to_clock <names>] [-topology]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-detail <histogram|path>`	Option to determine how much detail should be shown in the report
	`-fall_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-hold`	Option to report clock hold paths
	`-inter_clock`	Only report paths whose launch and latch clock are different
	`-intra_clock`	Only report paths whose launch and latch clock are the same
	`-less_than_slack <slack limit>`	Limit the paths reported to those with slack values less than the specified limit.
	`-npaths <number>`	Specifies the number of paths to report (default=1, or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported but be wary that this may be slow)

continued...

	`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same value as nworst
	`-pairs_only`	When set, paths with the same start and end points are considered equivalent. Only the worst case path for each unique combination is displayed.
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-recovery`	Option to report recovery paths
	`-removal`	Option to report removal paths
	`-rise_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Option to report clock setup paths
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-topology`	Topology Analysis (using arbitrary path analysis)
Description		

```
Reports a summary showing the distribution of logic depth among the critical
paths. Logic depth typically corresponds to the number of look-up tables (LUTs)
that a path passes through. If a path passes through non-LUT elements such
as RAM or DSP blocks, special rules may apply.

Use the "-setup", "-hold", "-recovery", "-removal", or "-topology" options to
specify which kind of analysis should be performed. In topology analysis,
paths are ordered by logic depth as opposed to slack.

The report can be directed to the Tcl console ("-stdout", default), a
file ("-file"), the Timing Analyzer graphical user interface
("-panel_name"), or any combination of the three.

You can limit the analysis performed by this command to specific
start and end points, using the "-from" and "-to" options. Use the
"-rise_from" and "-fall_from" options to limit the analysis to
endpoints with established high or low starting states. Use the
"rise_to" and "fall_to" options to limit the analysis to
destination points with high or low ending states.

The analysis can be further limited to clocks using the "-from_clock"
and "-to_clock"    options, or to specific edges of the clock using the
"-rise_from_clock", "-fall_from_clock", "-rise_to_clock", and
"-fall_to_clock" options.

Additionally, the "-through" option can be used to restrict analysis
to paths which go through specified pins or nets. Use the "rise_through"
and "fall_through" options to limit the analysis to intermediate points
```

continued...

```
with high or low ending states.

Use "-npaths" to limit the number of paths to report.  If you do not
specify this option, 1000 paths with the worst slack are included in
the report. Use the "-less_than_slack" option to limit output to all
paths with slack less than the specified value, up to the number specified
by "-npaths".

Use "-nworst" to limit the number of paths reported for each unique
endpoint.  If you do not specify this option, the number of paths
reported for each destination node is bounded only by the "-npaths"
option.  If this option is used, but "-npaths" is not specified, then
"-npaths" defaults to the same value specified for "-nworst".

Use the "-pairs_only" option to filter the output further, restricting
the results to only unique combinations of start and end points.

Use the "-detail" option to specify the desired level of report
detail. Specifiying "histogram" generates a summary showing the
distibution of logic depth among the critical paths. A row is
provided for each clock and a column for each logic depth.
Specifying "path" generates a table of paths as rows with a column
coresponding the paths' logic depths similar to the tables
reported by "report_timing" and "report_path".
The default behavior is to report a histogram.

The values of the "-from", "-to", and "-through" options are either
collections or a Tcl list of wildcards used to create collections
of appropriate types. The values used must follow standard Tcl or
Timing Analyzer-extension substitution rules.  See the help for
use_timing_analyzer_style_escaping for details.

The following options are not supported for this command:
--ccpp, --summary_view, --show_routing, --show_xtalk, --detail, --false_path

If any of the following filters are provided, then the logic depth distribution
occurs over "n critical paths" rather than "n critical paths per clock"
-from, -to, -through, -from_clock, -to_clock, -rise_from, -fall_from,
-rise_to, -fall_to, -rise_from_clock, -fall_from_clock, -rise_to_clock,
-fall_to_clock, -rise_through, -fall_through
```

Example Usage

```
project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

report_logic_depth -npaths 1000 -file "logic_depth.txt"

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.
TCL_ERROR	1	ERROR: Collection type '<*string*>' is not a valid type for a through collection. Valid collection types are 'pin' and 'net'
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Report database is not open

3.1.37.56. report_max_clock_skew (::quartus::sta)

The following table displays information for the `report_max_clock_skew` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_max_clock_skew [-h | -help] [-long_help] [-append] [-detail <summary|full_path>] [-file <name>] [-less_than_slack <slack limit>] [-npaths <number>] [-panel_name <name>] [-show_routing] [-stdout]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.	
	`-detail <summary|full_path>`	Option to determine how much detail should be shown in the path report	
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension	
	`-less_than_slack <slack limit>`	Limit the paths reported to those with slack values less than the specified limit.	
	`-npaths <number>`	Specifies the number of paths to report for each latest and earliest arrival skew result per set_max_skew assignment (default=1)	
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel	
	`-show_routing`	Option to display detailed routing in the path report	
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.	
Description	`TODO`		
Example Usage	`#TODO` `report_max_clock_skew`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.57. report_max_skew (::quartus::sta)

The following table displays information for the `report_max_skew` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615
Syntax	`report_max_skew [-h | -help] [-long_help] [-append] [-detail <summary|path_only|path_and_clock|full_path>] [-file <name>] [-less_than_slack <slack limit>] [-npaths <number>] [-panel_name <name>] [-show_routing] [-stdout]`

continued...

Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-detail` *<summary|path_only|path_and_clock|full_path>*	Option to determine how much detail should be shown in the path report
	`-file` *<name>*	Sends the results to an ASCII or HTML file. Depending on the extension
	`-less_than_slack` *<slack limit>*	Limit the paths reported to those with slack values less than the specified limit.
	`-npaths` *<number>*	Specifies the number of paths to report for each latest and earliest arrival skew result per set_max_skew assignment (default=1)
	`-panel_name` *<name>*	Sends the results to the panel and specifies the name of the new panel
	`-show_routing`	Option to display detailed routing in the path report
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description		

```
Reports max skew analysis results for all set_max_skew
commands in a single report. For each valid
set_max_skew constraint, this command computes skew
with respect to the latest and the earliest arrival of
each path.

By default, "Skew for the Latest Arrival" is computed
by comparing the latest arrival of each path with the
earliest arrival of the path that has the smallest
value for early arrival of all other paths included
in the constraint. Similarly, "Skew for the Earliest
Arrival" is computed by comparing the earliest arrival
of each path with the latest arrival of the path that
has the largest value for late arrival of all other
paths included in the constraint. No path is compared
with itself.

Use the -stdout option to direct the report to the Tcl
console (default), the -file option to write the
report to a file or the -panel_name option to direct
the report to the Timing Analyzer graphical user
interface. You can use these options in any
combination.

Use the -npaths option to limit the number of path
result pairs reported for each set_max_skew
constraint. If you do not specify this option,
report_max_skew only reports the result pair for the
single worst-case path. Use the -less_than_slack
option to limit output to all paths with slack less
than the specified value, up to the number specified
with -npaths.

Use the -detail option to specify the desired level of
report detail. The -detail summary option generates a
single table listing only the highlights of each path
(and is the same as -summary option, which this
replaces. "-detail path_only" (default) reports the
path from the source to the destination without any
detail about the clock path. Instead, the clock
network delay is shown as a single number. "-detail
path_and_clock" extends the arrival and required paths
back to the launch and latch clocks. "-detail
full_path" continues tracing back through generated
clocks to the underlying base clock.

The -show_routing option displays detailed routing
information in the path. Lines marked "IC" without the
option are shown, but only as a placeholder. The
routing elements for that line are broken out
```

continued...

```
individually and listed before the line.

The return value of this command is a two-element
list. The first number is the number of paths found in
the analysis. The second is the worst-case slack, in
terms of the current default time unit.

The "RF" column in the report output uses a two-letter
symbol to indicate the rise/fall transition that
occurs at that point in the path.

Possible "RF" values are:

    Value     Description
    -------   ------------------------------
    (empty)   Unknown transition
    R         Rising output
    F         Falling output
    RR        Rising input, rising output
    RF        Rising input, falling output
    FR        Falling input, rising output
    FF        Falling input, falling output

The "Type" column in the report uses a symbol to
indicate what type of delay occurs at that point in
the path.

Possible "Type" values are:

    Value     Description
    -------   ------------------------------------------------------------
    CELL      Cell delay
    COMP      PLL clock network compensation delay
    IC        Interconnect delay
    iExt      External input delay
    LOOP      Lumped combinational loop delay
    oExt      External output delay
    RE        Routing element (only for paths generated with the
              -show_routing option)
    uTco      Register micro-Tco time
    uTsu      Register micro-Tsu time
    uTh       Register micro-Th time
```

Example Usage

```
project_open my_project
create_timing_netlist
read_sdc
update_timing_netlist

# create max skew constraints
set_max_skew -from [get_ports data_ports[*]] -to [get_keepers *] 0.200
set_max_skew -from [get_keepers *] -to [get_ports output_ports[*]] 0.100

# show worst 10 paths for each earliest and latest arrival results
# per max_skew assignment assuming that their slack is less than 0.100
report_max_skew -panel_name "Report Max Skew" -npaths 10 -less_than_slack 0.100 -detail
full_path

delete_timing_netlist
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Report database is not open

3.1.37.58. report_metastability (::quartus::sta)

The following table displays information for the `report_metastability` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_metastability [-h | -help] [-long_help] [-append] [-file <name> ] [-length <number> ] [-max_length <number> ] [-min_length <number> ] [-nchains <number> ] [-panel_name <name> ] [-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-length <number>`	Reports only the synchronizer chains of an exact length
	`-max_length <number>`	Specifies the maximum length of a chain that appears in the report (default=no limit)
	`-min_length <number>`	Specifies the minimum length of a chain that appears in the report (default=0)
	`-nchains <number>`	Specifies the number of chains to report (default=1)
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description		

```
Report can be directed to the Tcl console ("-stdout", default), a file
("-file"), the Timing Analyzer graphical interface ("-panel_name"), or any
combination of the three.

The report_metastability function can be used to estimate the robustness
of asynchronous transfers in your design.

----------
Background
----------

Synchronization register chains should be used when transferring data
between unrelated clock domains to greatly reduce the probability of the
captured data signal becoming metastable.  A synchronization register
chain is a sequence of registers with the same clock, that is driven by a pin,
or logic from an unrelated clock domain.  The output of all but the last
register in the chain must connect only to the next register, either directly
or indirectly through logic.

When a register is metastable, its output hovers at a voltage between high
and low for a length of time beyond the normal Tco for the register.  The
design can fail if subsequent registers that use this metastable signal
latch different values.  Therefore, it is important to properly
synchronize data signals to prevent such occurrences.

------
Output
------

The report_metastability function generates a list of synchronization
register chains found in the design, and can provide estimates of the Mean
Time Between Failures (MTBF) of each chain.  The design MTBF is an estimate
of the overall robustness of the design, computed from the MTBF results from
all synchronization chains with calculated MTBFs.  The design MTBF metric is
reported only when the design meets timing.  Therefore, it is important to
fully timing constrain your design.
```

continued...

```
The typical MTBF result assumes typical silicon characteristics for the
selected device speed grade, with nominal operating conditions.

The worst case MTBF result uses the worst case silicon characteristics for
the selected device speed grade, with worst case operating conditions.

--------
Settings
--------

To get a list of possible synchronization chains, set "Synchronizer
Identification" to AUTO in the Timing Analyzer Page in the Settings
dialog box.  This sets the "Synchronizer Identification" QSF
assignment in your QSF file.  The Timing Analyzer uses timing constraints to
automatically detect synchronization chains in the design. Metastability
analysis checks for signal transfers between circuitry in unrelated or
asynchronous clock domains, so clock domains must be related correctly
with the timing constraints.

Set the maximum number of registers to consider as part of one
synchronization chain, via the "Synchronization Register Chain Length"
setting under Analysis and Synthesis Page in the Settings dialog box.
The default length is 2. All the registers in a chain (up to this length)
are protected from optimizations that can decrease MTBF.

Note that if you change the "Synchronizer Identification" setting, you
should rerun the Fitter, as this setting can impact some optimization
algorithms.

Use the -nchains option to limit the number of chains to report. If you
do not specify this option, only the single worst-case chain is reported.

To filter the synchronizer chains by their lengths, use the -min_length option
to set the minimum chain length to be reported, and use the -max_length
option to set the maximum chain length to be reported. Alternatively, use
the -length option to report only chains with a specific length.

-------------
Report Panels
-------------

The MTBF Summary report provides the estimated mean time between failure
for the design.  This is an estimate for the overall robustness of the
design in terms of metastability, and it is computed from all available
synchronization chain MTBFs present in the design.

The MTBF metric of automatically identified synchronization chains is not
computed.  To compute the MTBF of a synchronization chain, set
"Synchronizer Identification" to "Forced If Asynchronous" or "Forced"
for all registers of the synchronization chain.  By explicitly specifying
that this synchronization chain is valid, this chain will then be optimized
during the Fitter, and its MTBF will be computed.  Its MTBF will then be
included in the computation of the design MTBF.

The Synchronizer Summary table lists all the synchronization chains
found in your design.  It is possible that the analysis performed might
erroneously interpret certain structures, such as shift registers, as
synchronization chains.  If some synchronization chains are misidentified
and you wish to remove them from the report, you can turn off analysis of
these paths by making node-based assignments via the Assignment Editor,
set "Synchronizer Identification" to "Off" for the first register in these
synchronization chains.  Conversely, if there are synchronization
chains in your design that were not detected, you can set "Synchronizer
Identification" assignment to "Forced If Asynchronous" for all registers in
this chain through the Assignment Editor, and this chain will be
reported if it meets the criteria for being a synchronization chain.  This
can often occur if there is logic present between the registers of the
synchronization chain.  In the automatic mode of synchronizer identification,
these structures are not considered to be synchronizers. If you want to
force a register to be identified as the head of a synchronizer, set the
"Synchronizer Identification" assignment to "Forced" to the register,
and it will always be identified as the first of a synchronization chain.
This setting should not be applied to the entire design, since this
identifies every register in the design as a synchronizer.

The MTBF estimates assume the data being synchronized is switching at a
toggle rate of 12.5% of the source clock frequency.  That is, the estimates
assume that the arriving data signal switches once every 8 source clock
cycles.  If multiple clocks apply, the highest frequency is used.    If no
source clocks can be determined, then the data rate is taken as 12.5% of
the synchronization clock frequency.

If you know the approximate rate at which the data changes, and would like
to obtain a more accurate MTBF, use the "Synchronizer Toggle Rate" assignment
in the Assignment Editor.  Set the data toggle rate, in number of transitions
per second, on the first register of a synchronization chain.  The Timing
Analyzer then takes the specified rate into account when computing the MTBF
```

continued...

	of that particular chain. You can also apply this assignment to an entity or the entire design. Since a "Synchronizer Toggle Rate" assignment of 0 indicates that the data signal never toggles, the affected synchronization chain will not be reported since it does not affect the relibility of the design.
Example Usage	`project_open top` `create_timing_netlist` `read_sdc` `update_timing_netlist` `report_metastability` `# Reports only 10 chains that are between 2 and 4 registers long` `report_metastability -min_length 2 -max_length 4 -nchains 10` `delete_timing_netlist` `project_close`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.59. report_min_pulse_width (::quartus::sta)

The following table displays information for the `report_min_pulse_width` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_min_pulse_width [-h | -help] [-long_help] [-append] [-detail` *<summary|full_path>* `] [-file` *<name>* `] [-nworst` *<number>* `] [-panel_name` *<name>* `] [-show_routing] [-split_by_corner] [-stdout] [-type` *<all|min_period|clock_pulse>* `] [` *<targets>* `]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-detail` *<summary|full_path>*	Option to determine how much detail should be shown in the report
	`-file` *<name>*	Sends the results to an ASCII or HTML file. Depending on the extension
	`-nworst` *<number>*	Specifies the number of pulse width checks to report (default=1)
	`-panel_name` *<name>*	Sends the results to the panel and specifies the name of the new panel
	`-show_routing`	Option to display detailed routing in the path report
	`-split_by_corner`	When set, running this command with the -panel option creates a folder containing versions of this report for selected multiple operating conditions. This option has no effect when used with the -stdout or -file options.
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-type` *<all|min_period|clock_pulse>*	Option to determine the minimum pulse width analysis type
	<targets>	Either clocks or nodes such as ports and registers.

continued...

Description	Reports the results of minimum pulse width and minimum period checks. A minimum pulse width check verifies that a clock high ("High") or low ("Low") pulse sustains long enough to qualify as a recognizable change in the clock signal at a register or latch clock pin. A failed minimum pulse width check indicates that the register or latch may not recognize the clock transition. Each node in the design is reported twice per clock for minimum pulse width checks: once for the high pulse and once for the low pulse. While registers and latches may have a pulse width requirement greater than zero, other nodes are checked to ensure the pulse does not collapse entirely. A minimum period check verifies that the clock period ("Period") is large enough for the device to operate. Minimum period checks apply to registers and latches. The results of the minimum pulse width checks can be output to the Tcl console ("-stdout," the default), a report panel ("-panel"), a file ("-file"), or a combination of the three. Results are sorted from worst-case slack to best-case slack. To limit the number of checks reported, use the "-nworst" option. Results can be shown in summary ("-detail summary") or in detail ("-detail full_path," the default), showing the path details for the clock arrival times and how they affect the actual pulse width. The value of the targets is either a collection or a Tcl list of wildcards used to create a collection of the appropriate type. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See the help for use_timing_analyzer_style_escaping for details.
Example Usage	`project_open top` `create_timing_netlist` `read_sdc` `update_timing_netlist` `# Report the worst 100 minimum pulse width checks` `report_min_pulse_width -nworst 100` `# Report minimum pulse width checks for the register test_reg[*]` `report_min_pulse_width test_reg[*]` `# Output the previous results to a report panel and a file.` `report_min_pulse_width -panel_name "Min Pulse (test_reg)" test_reg[*]` `# Output the previous results to a file.` `report_min_pulse_width -file min_pulse_test_reg.txt test_reg[*]`

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Report database is not open

3.1.37.60. report_neighbor_paths (::quartus::sta)

The following table displays information for the `report_neighbor_paths` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615
Syntax	`report_neighbor_paths [-h | -help] [-long_help] [-append] [-asynch_clock] [-enable_complementary] [-extra_info <basic|all|none>] [-fall_from <names>] [-fall_from_clock <names>] [-fall_through <names>] [-fall_to <names>] [-fall_to_clock <names>] [-file <name>] [-from <names>] [-from_clock <names>] [-hold] [-inter_clock] [-intra_clock] [-less_than_slack <slack limit>] [-neighbor_path_num <number>] [-npaths <number>] [-nworst <number>] [-pairs_only] [-panel_name <name>] [-recovery] [-removal] [-rise_from <names>]`

continued...

	`[-rise_from_clock <names> ] [-rise_through <names> ] [-rise_to <names> ] [-rise_to_clock <names> ] [-setup] [-stdout] [-through <names> ] [-to <names> ] [-to_clock <names> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-enable_complementary`	Option to enable complementary analysis for the neighbor paths report
	`-extra_info <basic|all|none>`	Option to determine how much detail should be shown in the Extra Info report
	`-fall_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-hold`	Option to report clock hold paths
	`-inter_clock`	Only report paths whose launch and latch clock are different
	`-intra_clock`	Only report paths whose launch and latch clock are the same
	`-less_than_slack <slack limit>`	Limit the paths reported to those with slack values less than the specified limit.
	`-neighbor_path_num <number>`	Specifies the number of before and after paths for the neighbor paths report (default=1)
	`-npaths <number>`	Specifies the number of paths to report (default=1, or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported but be wary that this may be slow)
	`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same value as nworst

continued...

	`-pairs_only`	When set, paths with the same start and end points are considered equivalent. Only the worst case path for each unique combination is displayed.
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-recovery`	Option to report recovery paths
	`-removal`	Option to report removal paths
	`-rise_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Option to report clock setup paths
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
Description		

```
Reports the most timing-critical paths in the design, including
associated slack and additional path summary information, including
path bounding boxes. Additionally, for each path, shows its most
timing-critical neighbor paths: the path with the worst slack that
fans into the source of the current path (shown on the report as
"Critical Path Before") and the path with the worst slack that fans
out of the destination of the current path (shown as "Critical
Path After"). Note that if a register's output is connected to its
own input, one or both of the neighbor paths may be identical to
the current path. When looking for neighbor paths with the worst
slack, all operating conditions are considered, not just the operating
conditions of the main path itself.

Use the "-setup", "-hold", "-recovery", or "-removal" options to
specify which kind of analysis should be performed.

The report can be directed to the Tcl console ("-stdout", default), a
file ("-file"), the Timing Analyzer graphical user interface
("-panel_name"), or any combination of the three.

You can limit the analysis performed by this command to specific
start and end points, using the "-from" and "-to" options. Use the
"-rise_from" and "-fall_from" options to limit the analysis to
endpoints with established high or low starting states. Use the
"rise_to" and "fall_to" options to limit the analysis to
destination points with high or low ending states.

The analysis can be further limited to clocks using the "-from_clock"
and "-to_clock"    options, or to specific edges of the clock using the
"-rise_from_clock", "-fall_from_clock", "-rise_to_clock", and
"-fall_to_clock" options.

Additionally, the "-through" option can be used to restrict analysis
to paths which go through specified pins or nets. Use the "rise_through"
and "fall_through" options to limit the analysis to intermediate points
with high or low ending states.
```

continued...

```
Use "-npaths" to limit the number of paths to report.  If you do not
specify this option, only the single worst-case path is provided.  Use
the "-less_than_slack" option to limit output to all paths with slack
less than the specified value, up to the number specified by
"-npaths".

Use "-nworst" to limit the number of paths reported for each unique
endpoint.  If you do not specify this option, the number of paths
reported for each destination node is bounded only by the "-npaths"
option.  If this option is used, but "-npaths" is not specified, then
"-npaths" defaults to the same value specified for "-nworst".

Use the "-pairs_only" option to filter the output further, restricting
the results to only unique combinations of start and end points.

The values of the "-from", "-to", and "-through" options are either
collections or a Tcl list of wildcards used to create collections
of appropriate types. The values used must follow standard Tcl or
Timing Analyzer-extension substitution rules.  See the help for
use_timing_analyzer_style_escaping for details.

The following options are not supported for this command:
--ccpp, --summary_view, --show_routing, --show_xtalk, --false_path

Ideally you should use -panel_name or -file option to get the best
view of the report.

Here is description of each of the rows displayed for each path:

From: Source node in the path
To: Destination node in the path
Launch Clock: Source node clock
Latch Clock: Destination node clock
Relationship: Clock relationship
Slack: Slack on the path
Complement Analysis Slack: Slack of the same path via complement analysis
Number of Paths: Number of paths with the same source and destination nodes
Clock Skew: Clock Skew on the path
Data Delay: Data delay on the path ( ~ cell delay + interconnect delay
                                                    + misc. delay )
uTCO Delay: input delay
Cell Delay: Cell delay
Interconnect Delay: Wire Delay
Misc. Delay: Other kinds of delay
Uncertainty Delay: Clock uncertainty delay
uTSU Delay: Data stability delay - setup
uTH Delay: Data stability delay - hold
Logic Levels: Logic levels in the path
Max Fanout: Maximum fanout for any node on the path
I/O Crossings: Number of I/O crossings in the path
Number of wires: number of wires encountered on the path
Source/Destination Bounding Box: Co-ordinates covered by source destination blocks
Cell Bounding Box: Co-ordinates covered by cell blocks
Interconnect Bounding Box: Co-ordinates covered by wires
Source/Destination Relative Area: Normalized to 1.0
Cell Relative Area: Ratio of cell area against src/dst bounding area
Interconnect Relative Area: Ration of wire area against src/dst bounding area
Elements on Path: Displays the element type that make up that path
TDB Names Along Path: All the TDB elements that make up the path (only available with
                      developer license)
Corner: Describes the corner the path is found in
Complement Analysis Corner: Corner of the complement analysis path found
```

Example Usage

```
project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

report_neighbor_paths -npaths 10 -file "neighbor_path_analysis.txt"

# The following command is optional
delete_timing_netlist

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.

continued...

	TCL_ERROR	1	ERROR: Collection type '<*string*>' is not a valid type for a through collection. Valid collection types are 'pin' and 'net'
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.61. report_net_delay (::quartus::sta)

The following table displays information for the `report_net_delay` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_net_delay [-h | -help] [-long_help] [-append] [-file <name> ] [-nworst <number> ] [-panel_name <name> ] [-split_by_corner] [-stdout]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-append`		If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`		Sends the results to an ASCII or HTML file. Depending on the extension
	`-nworst <number>`		Specifies the maximum number of paths to report for each analysis. If unspecified, there is no limit.
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-split_by_corner`		When set, running this command with the -panel option creates a folder containing versions of this report for selected multiple operating conditions. This option has no effect when used with the -stdout or -file options.
	`-stdout`		Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description			
Example Usage			
Return Value	**Code Name**	**Code**	**String Return**

```
Reports net delay analysis results based on set_net_delay
commands. Each set_net_delay command is treated as a separate analysis
and report_net_delay reports the results of all set_net_delay
commands in a single report.

The report contains each set_net_delay command with the worst case
slack result followed by the results of each edge matching the
criteria set by that set_net_delay command. These results are ordered
based on the slack value.

Use -nworst option to limit the number of lines reported for a
set_net_delay command.
```

```
project_open my_project
create_timing_netlist
read_sdc
update_timing_netlist

set_net_delay -min 0.160 -from [get_pins inst9|combout] -to [get_pins *|dataf]
set_net_delay -max 0.500 -from inst8|combout

report_net_delay -panel "Net Delay"
```

continued...

TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Report database is not open

3.1.37.62. report_net_timing (::quartus::sta)

The following table displays information for the `report_net_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_net_timing [-h | -help] [-long_help] [-append] [-file <name> ] [-nworst_delay <number> ] [-nworst_fanout <number> ] [-panel_name <name> ] [-stdout] [ <name> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-nworst_delay <number>`	Report worst net delays
	`-nworst_fanout <number>`	Report worst fanout nets
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	<name>	Signal or collection name
Description		

```
Reports delay and fanout information about a net in the design. A net
corresponds to a cell output pin.

Report can be directed to the Tcl console ("-stdout", default), a file
("-file"), the Timing Analyzer graphical interface ("-panel_name"), or any
combination of the three.

The value of the name is either a collection or a Tcl list of
wildcards used to create a collection of the appropriate type.  The
values used must follow standard Tcl or Timing Analyzer-extension
substitution rules.  See the help for use_timing_analyzer_style_escaping for
details.
```

Example Usage

```
project_open <design>
create_timing_netlist

# Show delay and fanout information for all nets
# that match "abc*"
report_net_timing [get_nets abc*]

# Report delay and fanout information for the 10
# nets showing higher delays
report_net_timing -nworst_delay 10

# Report delay and fanout information for the 10
# nets showing higher fanout
report_net_timing -nworst_fanout 10

project_close
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Options -<*string*> and -<*string*> are mutually exclusive. Specify only one of the two options.
	TCL_ERROR	1	ERROR: You must open a project before you can use this command.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Neither of options -<*string*> or -<*string*> is specified. Specify one of the two options.

3.1.37.63. report_partitions (::quartus::sta)

The following table displays information for the `report_partitions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_partitions [-h | -help] [-long_help] [-append] [-file <name> ] [-from_clock <names> ] [-nworst <number> ] [-panel_name <name> ] [-stdout] [-to_clock <names> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-nworst <number>`	Specifies the maximum number of paths to report between partitions. If unspecified, the limit defaults to 1000
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
Description		

```
Reports timing information related to design partitions.

The report_partitions command analyzes the worst 1000 failing setup paths
in the design by default, but you can optionally set the nworst option to
increase or decrease this number.  The from_clock and to_clock arguments
can be used to control how this function finds the paths to be analyzed.

This function reports the number of failing paths within each partition and
the worst-case slack of the paths that are entirely contained within a single partition
in a Partition Timing Overview table.

The function also creates a Partition Timing Details table that lists
the number of failing paths and worst-case slack of paths that feed from
one partition to another partition. This information provides more details
on where the critical paths in the design are with respect to design partitions.

The function also creates a Partition Timing Breakdown table. This table bins the
worst failing paths by all partitions that the path spans, and reports the
```

continued...

	number of failing paths and worst case slack for each group of partitions. This report can be directed to the Tcl console ("-stdout", default), a file ("-file"), the Timing Analyzer graphical user interface ("-panel_name"), or any combination of the three.
Example Usage	project_open my_project create_timing_netlist read_sdc update_timing_netlist # Report a maximum of 500 failing paths between partitions to the # Timing Analyzer graphical interface and to the Tcl console. report_partitions -panel_name "Partition Timing Report" -nworst 500 -stdout

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Design partitions are not supported in this project.

3.1.37.64. report_path (::quartus::sta)

The following table displays information for the `report_path` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_path [-h | -help] [-long_help] [-append] [-fall_from <names>] [-fall_through <names>] [-fall_to <names>] [-file <name>] [-from <names>] [-list_clocks] [-logic_depth] [-min_path] [-npaths <number>] [-nworst <number>] [-pairs_only] [-panel_name <name>] [-rise_from <names>] [-rise_through <names>] [-rise_to <names>] [-show_routing] [-split_by_corner] [-stdout] [-summary] [-through <names>] [-to <names>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-fall_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-list_clocks`	Includes the driving clocks associated with the from and to nodes of each path in the Path Summary table
	`-logic_depth`	Option to display the logic depth instead of path delay
	`-min_path`	Find the minimum delay path(s)

continued...

	`-npaths <number>`	Specifies the number of paths to report (default=1, or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported but be wary that this may be slow)
	`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same as nworst
	`-pairs_only`	When set, paths with the same start and end points are considered to be equivalent. Only the longest delay path for each unique combination is displayed.
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-rise_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-show_routing`	Option to display detailed routing in the path
	`-split_by_corner`	When set, running this command with the -panel option creates a folder containing versions of this report for selected multiple operating conditions. This option has no effect when used with the -stdout or -file options.
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-summary`	Create a single table with a summary of each path found
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)

Description

```
Reports the longest delay paths and the corresponding delay value.

The report can be directed to the Tcl console ("-stdout", default),
a file ("-file"), the Timing Analyzer graphical user interface
("-panel_name"), or any combination of the three.

You can limit the analysis performed by this command to specific
start and end points, using the "-from" and "-to" options.  Any
node or cell in the design is considered a valid endpoint.
Additionally, the "-through" option can be used to restrict analysis
to paths which go through specified pins or nets.  Paths that are
reported can not start before or go beyond a keeper node (register
or port); this restriction considers register pins as combinational
nodes in the design.

Use "-npaths" to limit the number of paths to report.  If this
option is not specified, only the single longest delay path is
provided.

Use "-nworst" to limit the number of paths reported for each
unique endpoint.  If you do not specify this option, the number
of paths reported for each destination node is bounded only by the
"-npaths" option.  If this option is used, but "-npaths" is not
specified, then "-npaths" defaults to the same value specified
for  "-nworst".

Use the "-pairs_only" option to filter the output further, restricting
the results to only unique combinations of start and end points.

Use the "-summary" option to generate a single table listing only
the highlights of each path.
```

continued...

```
The "-min_path" option finds the minimum delay path(s) rather
than the maximum delay paths which is the default behavior.

The "-show_routing" option displays detailed routing information
in the path.  Lines that were marked as "IC" without the option
are still shown, but only as a placeholder.  The routing elements
for that line are broken out individually and listed before
the line.

The return value of this command is a two-element list.  The
first number is the number of paths found in the analysis.  The
second is the longest delay, in terms of the current default
time unit.

The values of the "-from", "-to", "-through" options are either collections
or a Tcl list of wildcards used to create collections of appropriate types.
The values used must follow standard Tcl or Timing Analyzer-extension
substitution rules.  See the help for use_timing_analyzer_style_escaping for
details.
```

Example Usage

```
project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# Report path delay between nodes "foo" and "bar",
# reporting the longest delay if a path is found.

set my_list [report_path -from foo -to bar]
set num_paths [lindex $my_list 0]
set longest_delay [lindex $my_list 1]
if { $num_paths > 0 } {
    puts "Longest delay -from foo -to bar is $longest_delay"
}

# The following command is optional
delete_timing_netlist

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.
TCL_ERROR	1	ERROR: Collection type '<*string*>' is not a valid type for a through collection. Valid collection types are 'pin' and 'net'
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Report database is not open

3.1.37.65. report_pipelining_info (::quartus::sta)

The following table displays information for the `report_pipelining_info` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_pipelining_info [-h | -help] [-long_help] [-append] [-bus_name <names> ] [-file <name> ] [-max_rows <number> ] [-min_depth <number> ] [-min_width <number> ] [-panel_name <name> ] [-stdout]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values

continued...

	-append	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	-bus_name <*names*>	When set, pipelining report gives detailed information on the specific bus
	-file <*name*>	Sends the results to an ASCII or HTML file. Depending on the extension
	-max_rows <*number*>	Specifies the maximum number of rows to report
	-min_depth <*number*>	Specifies the minimum average bus depth
	-min_width <*number*>	Specifies the minimum bus width
	-panel_name <*name*>	Sends the results to the panel and specifies the name of the new panel
	-stdout	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.

Description

```
This command reports back-to-back pipelining registers information and collapses
pipelining chains based on which bus they are in.

Use the "-min_depth" option to specify the minimum average depths of pipelining
buses to report. The default value for this field is "3".

Use the "-min_width" option to specify the minimum width of pipelining buses to
report. The default value for this field is "16".

By default, the command reports all of the pipelining buses with average depth
and width over the value specified by "-min_depth" and "-min_width" options.
To help check one pipelining bus in detail, this command also provides detailed
mode. Use the "bus_name" to specify which bus to focus.
```

Example Usage

```
This command currently contains no example usage.
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful

3.1.37.66. report_register_spread (::quartus::sta)

The following table displays information for the `report_register_spread` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_register_spread [-h | -help] [-long_help] [-append] [-file` <*name*> `] [-from_clock` <*names*> `] [-min_sinks` <*number*> `] [-num_registers` <*number*> `] [-panel_name` <*name*> `] [-sink_type` <*endpoint*|*immediate*> `] [-spread_type` <*tension*|*span*|*area*|*angle*|*count*> `] [-stdout] [-to_clock` <*names*> `]`	
Arguments	-h | -help	Short help
	-long_help	Long help with examples and possible return values
	-append	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	-file <*name*>	Sends the results to an ASCII or HTML file. Depending on the extension
	-from_clock <*names*>	Valid source clocks (string patterns are matched using Tcl string matching)

continued...

	`-min_sinks` <*number*>	Specifies the minimum number of sinks per register (must be at least 2)
	`-num_registers` <*number*>	Specifies the top N registers with highest spread
	`-panel_name` <*name*>	Sends the results to the panel and specifies the name of the new panel
	`-sink_type` <*endpoint*|*immediate*>	Determines which sink type [endpoint|immediate] is analyzed
	`-spread_type` <*tension*|*span*|*area*|*angle*|*count*>	Determines which spread type is analyzed [tension|span|area|angle|count]
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-to_clock` <*names*>	Valid sink clocks (string patterns are matched using Tcl string matching)

Description

```
This report analyzes the final placement of a design and strive to
identify registers with sinks that are pulling them in various directions.
These registers are then recommended as candidates for duplication
There are two types of sink: "Immediate Fan-Out" and "Timing Path
Endpoint". There are two types of pull: "Tension" and "Span"

The report can be directed to the Tcl console ("-stdout", default), a
file ("-file"), the Timing Analyzer graphical user interface
("-panel_name"), or any combination of the three.

Use "-sink_type" to specify between endpoints and immediate
fanouts to report on. If not specified, the default reports on
endpoint fanouts.
    Timing Path Endpoints: the nodes (usually registers) that terminate
        timing paths from a register
    Immediate Fanouts: the immediately connected nodes (lookup tables,
        other registers, RAM or DSP blocks, etc.) of the register.

Use "-spread_type" to specify the type of spread to report on, a user
can choose between:
    Tension: the sum over each sink of the distance from it to the
        centroid of all the sinks.
    Span: the maximum 1-dimensional delta between the left/bottom-most
        sink and the right/top-most sink.
    Area: the area covered by a box drawn around the left/bottom-most
        sink and the right/top-most sink.
    Angle: the angular span of the sinks around to the source, defined
        as 360 minus the largest angle between any two angularly-adjacent
        sinks that are each a sufficiently large distance from the source.
    Count: the number of each sink type associated with the source register.

The analysis can be limited to clocks using the "-from_clock" and "-to_clock"
options.

Use "-min_sinks" to filter out any registers with a number of sinks below
the threshold. Span, Area, and Angle have a low sensititivity to sink
count, which can result in registers with a small sink count having a larger
spread score than other registers with more sinks. The minimum number of
sinks must be at least 2 for the spread calculations to work properly.

Use "-num_registers" to limit the number source registers reported.
Registers are reported in decreasing order of the spread type selected
so the top n registers are reported. If you do not specify this
option, the number of source registers displayed is limited to a
maximum of 10.
```

Example Usage

```
project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

report_register_spread -num_registers 20 -spread_type "tension" -sink_type "endpoint"

project_close
```

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Invalid value %u for argument <*string*>. Specify a value of at least %u.

3.1.37.67. report_register_statistics (::quartus::sta)

The following table displays information for the `report_register_statistics` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_register_statistics [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-registers_without_clocks] [-stdout] [-unique_clocks_feeding_registers]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.	
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension	
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel	
	`-registers_without_clocks`	Include a column showing registers with no defined clock on their clock pin	
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.	
	`-unique_clocks_feeding_registers`	Include a column showing the number of unique clocks feeding registers clock pins	
Description	`This command reports reset statistics.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.68. report_retiming_restrictions (::quartus::sta)

The following table displays information for the `report_retiming_restrictions` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_retiming_restrictions [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-stdout]`	
Arguments	`-h | -help`	Short help

continued...

	`-long_help`		Long help with examples and possible return values
	`-append`		If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`		Sends the results to an ASCII or HTML file. Depending on the extension
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-stdout`		Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description			
	`This command reports retiming restrictions in a hierarchical form. The report helps users identify various types of retiming restrictions in each entity. Consider removing these retiming restrictions to allow retiming optimization to improve performance.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.69. report_route_net_of_interest (::quartus::sta)

The following table displays information for the `report_route_net_of_interest` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_route_net_of_interest [-h | -help] [-long_help] [-append] [-file <name> ] [-num_nets <number> ] [-panel_name <name> ] [-stdout]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-append`		If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`		Sends the results to an ASCII or HTML file. Depending on the extension
	`-num_nets <number>`		Specifies the number of nets of interest to report
	`-panel_name <name>`		Sends the results to the panel and specifies the name of the new panel
	`-stdout`		Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
Description			
	`This command reports nets that router works hardest on.`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.70. report_rskm (::quartus::sta)

The following table displays information for the `report_rskm` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_rskm [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-stdout]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.	
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension	
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel	
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.	
Description			
Example Usage			
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

```
Reports RSKM for dedicated LVDS circuitry.

In designs that use dedicated LVDS circuitry, receiver input skew
margin (RSKM) is the time margin available before the LVDS receiver
megafunction fails to operate. RSKM is defined as the total time
margin that remains after subtracting the sampling window (SW) size
and the receiver channel-to-channel skew (RCCS) from the time unit
interval (TUI), as expressed in the following formula:

RSKM = (TUI - SW - RCCS) /2

The time unit interval is the LVDS clock period (1/fmax). The sampling
window is the period of time that the input data must be stable to
ensure that the data is successfully sampled by the LVDS receiver
megafunction. The sampling window size varies by device speed
grade. RCCS is the difference between the fastest and slowest data
output transitions, including the tco variation and clock skew. To
obtain an accurate analysis of an LVDS circuit, you should assign an
appropriate input delay to the LVDS receiver megafunction. RCCS is
equal to the difference between maximum input delay and minimum input
delay. If no input delay is set, RCCS defaults to zero.
```

```
project_open top
create_timing_netlist
read_sdc
update_timing_netlist

# Ensure a tccs of 1ns
set_input_delay -max -clock lvds_clk 2ns [get_ports lvds_input]
set_input_delay -min -clock lvds_clk 1ns [get_ports lvds_input]

# Show lvds information
report_rskm
```

3.1.37.71. report_sdc (::quartus::sta)

The following table displays information for the `report_sdc` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615		
Syntax	`report_sdc [-h | -help] [-long_help] [-append] [-file <name> ] [-ignored] [-panel_name <name> ] [-stdout]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.	
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension	
	`-ignored`	Reports full history of assignments to locate ignored ones	
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel	
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.	
Description	`Reports all SDC constraints used in the design. Use the -ignored option to report SDC constraints that were ignored and the reason they were ignored.`		
Example Usage	`project_new test` `create_timing_netlist` `create_clock -period 10 -name clk10 clk` `set_multicycle_path -from [get_cells a] -to [get_cells b]` `update_timing_netlist` `report_sdc -panel_name sdc_report_panel` `report_timing` `delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.72. report_skew (::quartus::sta)

The following table displays information for the `report_skew` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615
Syntax	`report_skew [-h | -help] [-long_help] [-append] [-asynch_clock] [-detail <summary|path_only|path_and_clock|full_path> ] [-fall_from_clock <names> ] [-fall_to_clock <names> ] [-file <name> ] [-from <names> ] [-from_clock <names> ] [-greater_than_skew <slack limit> ] [-inter_clock] [-intra_clock] [-`

continued...

	`npaths <number> ] [-panel_name <name> ] [-rise_from_clock <names> ] [-rise_to_clock <names> ] [-show_routing] [-stdout] [-through <names> ] [-to <names> ] [-to_clock <names> ]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-detail <summary|path_only|path_and_clock|full_path>`	Option to determine how much detail should be shown in the path report
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-greater_than_skew <slack limit>`	Limit the paths reported to those with skew values greater than the specified limit.
	`-inter_clock`	Only report paths whose launch and latch clock are different
	`-intra_clock`	Only report paths whose launch and latch clock are the same
	`-npaths <number>`	Specifies the number of paths to report for each latest and earliest arrival skew result (default=1)
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-show_routing`	Option to display detailed routing in the path report
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)

continued...

Description

```
This report performs skew analysis on selected
paths. As opposed to report generated by
report_max_skew command, this command does not depend
on the existance of set_max_skew assignments. This
report computes skew with respect to the latest and
the earliest arrival of each selected path.

By default, "Skew for the Latest Arrival" is computed
by comparing the latest arrival of each path with the
earliest arrival of the path that has the smallest
value for early arrival of all other paths included
in the constraint. Similarly, "Skew for the Earliest
Arrival" is computed by comparing the earliest arrival
of each path with the latest arrival of the path that
has the largest value for late arrival of all other
paths included in the constraint. No path is compared
with itself.

Use the -stdout option to direct the report to the Tcl
console (default), the -file option to write the
report to a file or the -panel_name option to direct
the report to the Timing Analyzer graphical user
interface. You can use these options in any
combination.

Report skew includes data arrival times, clock arrival times,
register micro parameters, clock uncertainty, on-die variation and
ccpp removal.

Use the -npaths option to limit the number of path
result pairs reported for each set_max_skew
constraint. If you do not specify this option,
report_skew only reports the result pair for the
single worst-case path. Use the -less_than_slack
option to limit output to all paths with skew greater
than the specified value, up to the number specified
with -npaths.

Use the -detail option to specify the desired level of
report detail. "-detail path_only" (default) reports the
path from the source to the destination without any
detail about the clock path. Instead, the clock
network delay is shown as a single number. "-detail
path_and_clock" extends the arrival and required paths
back to the launch and latch clocks. "-detail
full_path" continues tracing back through generated
clocks to the underlying base clock. The "-detail summary"
option is deprecated.

The -show_routing option displays detailed routing
information in the path. Lines marked "IC" without the
option are shown, but only as a placeholder. The
routing elements for that line are broken out
individually and listed before the line.

The return value of this command is a two-element
list. The first number is the number of paths found in
the analysis. The second is the worst-case skew, in
terms of the current default time unit.

The "RF" column in the report output uses a two-letter
symbol to indicate the rise/fall transition that
occurs at that point in the path.

Possible "RF" values are:

    Value     Description
    -------   ------------------------------
    (empty)   Unknown transition
    R         Rising output
    F         Falling output
    RR        Rising input, rising output
    RF        Rising input, falling output
    FR        Falling input, rising output
    FF        Falling input, falling output

The "Type" column in the report uses a symbol to
indicate what type of delay occurs at that point in
the path.

Possible "Type" values are:

    Value     Description
    -------   ------------------------------
    CELL      Cell delay
    COMP      PLL clock network compensation delay
    IC        Interconnect delay
```

continued...

```
    iExt      External input delay
    LOOP      Lumped combinational loop delay
    oExt      External output delay
    RE        Routing element (only for paths
              generated with the -show_routing option)
    uTco      Register micro-Tco time
    uTsu      Register micro-Tsu time
    uTh       Register micro-Th time
```

Example Usage

```
project_open my_project
create_timing_netlist
read_sdc
update_timing_netlist

# show worst 10 paths for each earliest and latest arrival results
report_skew -from [get_ports input[*]] -to [get_registers *] -panel_name "Report Skew" -npaths
10 -greater_than_skew 0.100 -detail full_path

delete_timing_netlist
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Report database is not open

3.1.37.73. report_tccs (::quartus::sta)

The following table displays information for the `report_tccs` Tcl command:

Tcl Package and Version

Belongs to ::quartus::sta on page 615

Syntax

`report_tccs [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-stdout]`

Arguments

`-h | -help`	Short help
`-long_help`	Long help with examples and possible return values
`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.

Description

```
Reports TCCS for dedicated LVDS transmitters.

In designs that implement the LVDS I/O standard, transmitter
channel-to-channel skew (TCCS) is the timing difference between the
fastest and slowest output transitions, including tco variations and
clock skew.
```

Example Usage

```
project_open top
create_timing_netlist
read_sdc
update_timing_netlist
```

continued...

| | ```
# Show lvds information
report_tccs
``` | | |
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Report database is not open

3.1.37.74. report_timing (::quartus::sta)

The following table displays information for the `report_timing` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_timing [-h | -help] [-long_help] [-append] [-asynch_clock] [-data_delay] [-detail <summary|path_only|path_and_clock|full_path>] [-extra_info <basic|all|none>] [-fall_from <names>] [-fall_from_clock <names>] [-fall_through <names>] [-fall_to <names>] [-fall_to_clock <names>] [-false_path] [-file <name>] [-from <names>] [-from_clock <names>] [-hold] [-inter_clock] [-intra_clock] [-less_than_slack <slack limit>] [-npaths <number>] [-nworst <number>] [-pairs_only] [-panel_name <name>] [-recovery] [-removal] [-rise_from <names>] [-rise_from_clock <names>] [-rise_through <names>] [-rise_to <names>] [-rise_to_clock <names>] [-setup] [-show_routing] [-split_by_corner] [-stdout] [-through <names>] [-to <names>] [-to_clock <names>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-data_delay`	Report only paths that are covered by a data delay assignment
	`-detail <summary|path_only|path_and_clock|full_path>`	Option to determine how much detail should be shown in the path report
	`-extra_info <basic|all|none>`	Option to determine how much detail should be shown in the Extra Info report
	`-fall_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-false_path`	Report only paths that are cut by a false path assignment

continued...
```

`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
`-hold`	Option to report clock hold paths
`-inter_clock`	Only report paths whose launch and latch clock are different
`-intra_clock`	Only report paths whose launch and latch clock are the same
`-less_than_slack <slack limit>`	Limit the paths reported to those with slack values less than the specified limit.
`-npaths <number>`	Specifies the number of paths to report (default=1, or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported but be wary that this may be slow)
`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same value as nworst
`-pairs_only`	When set, paths with the same start and end points are considered equivalent. Only the worst case path for each unique combination is displayed.
`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
`-recovery`	Option to report recovery paths
`-removal`	Option to report removal paths
`-rise_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
`-rise_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
`-setup`	Option to report clock setup paths
`-show_routing`	Option to display detailed routing in the path
`-split_by_corner`	When set, running this command with the -panel option creates a folder containing versions of this report for selected multiple operating conditions. This option has no effect when used with the -stdout or -file options.
`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)

continued...

	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)

Description

```
Reports the worst-case paths and associated slack.

Use the "-setup", "-hold", "-recovery", or "-removal" options to
specify which kind of analysis should be performed.

The report can be directed to the Tcl console ("-stdout", default), a
file ("-file"), the Timing Analyzer graphical user interface
("-panel_name"), or any combination of the three.

You can limit the analysis performed by this command to specific
start and end points, using the "-from" and "-to" options. Use the
"-rise_from" and "-fall_from" options to limit the analysis to
endpoints with established high or low starting states. Use the
"-rise_to" and "-fall_to" options to limit the analysis to
destination points with high or low ending states.

The analysis can be further limited to clocks using the "-from_clock"
and "-to_clock"    options, or to specific edges of the clock using the
"-rise_from_clock", "-fall_from_clock", "-rise_to_clock", and
"-fall_to_clock" options.

Additionally, the "-through" option can be used to restrict analysis
to paths which go through specified pins or nets. Use the "-rise_through"
and "-fall_through" options to limit the analysis to intermediate points
with high or low ending states.

Use "-npaths" to limit the number of paths to report.  If you do not
specify this option, only the single worst-case path is provided.  Use
the "-less_than_slack" option to limit output to all paths with slack
less than the specified value, up to the number specified by
"-npaths".

Use "-nworst" to limit the number of paths reported for each unique
endpoint.  If you do not specify this option, the number of paths
reported for each destination node is bounded only by the "-npaths"
option.  If this option is used, but "-npaths" is not specified, then
"-npaths" defaults to the same value specified for "-nworst".

Use the "-pairs_only" option to filter the output further, restricting
the results to only unique combinations of start and end points.

Use the "-detail" option to specify the desired level of report
detail.  "-summary" generates a single table listing only the
highlights of each path (and is the same as "-summary" option, which
this replaces). "-path_only" reports the path from the source to the
destination without any detail about the clock path.  Instead, the
clock network delay is shown as a single number.  This is the default
behavior. "-path_and_clock" extends the arrival and required paths back
to the launch and latch clocks.  "-full_path" continues tracing
back through generated clocks to the underlying base clock.

The "-show_routing" option displays detailed routing information in
the path.  Lines that were marked as "IC" without the option are still
shown, but only as a placeholder.  The routing elements for that line
are broken out individually and listed before the line.

The "-false_path" option reports only those paths that are normally
hidden by false_path assignments or clock to clock cuts.  Like the
default report, this option only reports constrained paths.

The "-data_delay" option reports only those paths that are constrained
by set_data_delay. Without this option, such paths are excluded from
the report.

The return value of this command is a two-element list.  The first
number is the number of paths found in the analysis.  The second is
the worst-case slack, in terms of the current default time unit.

The "RF" column in the report output uses a two-letter symbol to
indicate the rise/fall transition that occurs at that point in the
path.

Possible "RF" values are:

    Value     Description
    -------   ------------------------------
    (empty)   Unknown transition
    R         Rising output
    F         Falling output
    RR        Rising input, rising output
```

continued...

```
    RF        Rising input, falling output
    FR        Falling input, rising output
    FF        Falling input, falling output

The "Type" column in the report uses a symbol to indicate what type
of delay occurs at that point in the path.

Possible "Type" values are:

    Value     Description
    -------   -----------------------------------------------------------
    CELL      Cell delay
    COMP      PLL clock network compensation delay
    IC        Interconnect delay
    iExt      External input delay
    LOOP      Lumped combinational loop delay
    oExt      External output delay
    RE        Routing element (only for paths generated with the
              -show_routing option)
    uTco      Register micro-Tco time
    uTsu      Register micro-Tsu time
    uTh       Register micro-Th time

The values of the "-from", "-to", and "-through" options are either
collections or a Tcl list of wildcards used to create collections
of appropriate types. The values used must follow standard Tcl or
Timing Analyzer-extension substitution rules.  See the help for
use_timing_analyzer_style_escaping for details.
```

Example Usage

```
project_open my_project

# Always create the netlist first
create_timing_netlist
read_sdc my_project.sdc
update_timing_netlist

# Run a setup analysis between nodes "foo" and "bar",
# reporting the worst-case slack if a path is found.

set my_list [report_timing -from foo -to bar]
set num_paths [lindex $my_list 0]
set wc_slack [lindex $my_list 1]
if { $num_paths > 0 } {
    puts "Worst case slack -from foo -to bar is $wc_slack"
}

# The following command is optional
delete_timing_netlist

project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Option <*string*> has illegal value: <*string*>. Specify a legal option value.
TCL_ERROR	1	ERROR: Collection type '<*string*>' is not a valid type for a through collection. Valid collection types are 'pin' and 'net'
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Report database is not open

3.1.37.75. report_timing_by_source_files (::quartus::sta)

The following table displays information for the `report_timing_by_source_files` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_timing_by_source_files [-h | -help] [-long_help] [-append] [-asynch_clock] [-fall_from <names>] [-fall_from_clock <names>] [-fall_through <names>] [-fall_to <names>] [-fall_to_clock <names>] [-false_path] [-file <name>] [-from <names>] [-from_clock <names>] [-hold] [-inter_clock] [-intra_clock] [-less_than_slack <slack limit>] [-npaths <number>] [-nworst <number>] [-pairs_only] [-panel_name <name>] [-recovery] [-removal] [-rise_from <names>] [-rise_from_clock <names>] [-rise_through <names>] [-rise_to <names>] [-rise_to_clock <names>] [-setup] [-stdout] [-through <names>] [-to <names>] [-to_clock <names>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-fall_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-fall_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-false_path`	Report only paths that are cut by a false path assignment
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension
	`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-hold`	Option to report clock hold paths
	`-inter_clock`	Only report paths whose launch and latch clock are different
	`-intra_clock`	Only report paths whose launch and latch clock are the same
	`-less_than_slack <slack limit>`	Limit the paths reported to those with slack values less than the specified limit.
	`-npaths <number>`	Specifies the number of paths to report (default=1, or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported but be wary that this may be slow)

continued...

	`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same value as nworst
	`-pairs_only`	When set, paths with the same start and end points are considered equivalent. Only the worst case path for each unique combination is displayed.
	`-panel_name <name>`	Sends the results to the panel and specifies the name of the new panel
	`-recovery`	Option to report recovery paths
	`-removal`	Option to report removal paths
	`-rise_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
	`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-rise_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
	`-setup`	Option to report clock setup paths
	`-stdout`	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
	`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
	`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
Description		

```
The command groups the most timing critical paths to identify the most timing
critical entities and source files.

Use the "-setup", "-hold", "-recovery", or "-removal" options to
specify which kind of analysis should be performed.

The report can be directed to the Tcl console ("-stdout", default), a
file ("-file"), the Timing Analyzer graphical user interface
("-panel_name"), or any combination of the three.

You can limit the analysis performed by this command to specific
start and end points, using the "-from" and "-to" options. Use the
"-rise_from" and "-fall_from" options to limit the analysis to
endpoints with established high or low starting states. Use the
"-rise_to" and "-fall_to" options to limit the analysis to
destination points with high or low ending states.

The analysis can be further limited to clocks using the "-from_clock"
and "-to_clock"    options, or to specific edges of the clock using the
"-rise_from_clock", "-fall_from_clock", "-rise_to_clock", and
"-fall_to_clock" options.

Additionally, the "-through" option can be used to restrict analysis
to paths which go through specified pins or nets. Use the "-rise_through"
and "-fall_through" options to limit the analysis to intermediate points
with high or low ending states.

Use "-npaths" to limit the number of paths to report.  If you do not
specify this option, 1000 worst-case paths are provided.  Use
the "-less_than_slack" option to limit output to all paths with slack
```

continued...

	less than the specified value, up to the number specified by "-npaths". Use "-nworst" to limit the number of paths reported for each unique endpoint. If you do not specify this option, the number of paths reported for each destination node is bounded only by the "-npaths" option. If this option is used, but "-npaths" is not specified, then "-npaths" defaults to the same value specified for "-nworst". Use the "-pairs_only" option to filter the output further, restricting the results to only unique combinations of start and end points. The "-false_path" option reports only those paths that are normally hidden by false_path assignments or clock to clock cuts. Like the default report, this option only reports constrained paths. The values of the "-from", "-to", and "-through" options are either collections or a Tcl list of wildcards used to create collections of appropriate types. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See the help for use_timing_analyzer_style_escaping for details. The return value of this command is a three-element list. The first number is the number of paths found in the analysis. The second is the worst-case slack, in terms of the current default time unit. The third is the total negative slack across all paths found in the analysis.
Example Usage	`set my_list [report_timing_by_source_files -setup -npaths 1000 -panel_name {Report Timing by Source Files}]` `set num_paths [lindex $my_list 0]` `set wc_slack [lindex $my_list 1]` `set tns [lindex $my_list 2]` `if { $num_paths > 0 } {` `  puts "Analyzed $num_paths paths against their source file(s)."` `  puts "Worst-case slack found is $wc_slack."` `  puts "Total Negative Slack found is $tns."` `}`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.76. report_timing_tree (::quartus::sta)

The following table displays information for the `report_timing_tree` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_timing_tree [-h | -help] [-long_help] [-asynch_clock] [-fall_from <names>] [-fall_from_clock <names>] [-fall_through <names>] [-fall_to <names>] [-fall_to_clock <names>] [-from <names>] [-from_clock <names>] [-hold] [-inter_clock] [-intra_clock] [-less_than_slack <slack limit>] [-npaths <number>] [-nworst <number>] [-pairs_only] -panel_name <name> [-recovery] [-removal] [-rise_from <names>] [-rise_from_clock <names>] [-rise_through <names>] [-rise_to <names>] [-rise_to_clock <names>] [-setup] [-through <names>] [-to <names>] [-to_clock <names>]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-asynch_clock`	Only report paths whose launch and latch clock do not share a common ancestor clock, or were explicitly marked as asynchronous via clock groups
	`-fall_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
	`-fall_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)

continued...

`-fall_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
`-fall_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
`-fall_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
`-from <names>`	Valid sources (string patterns are matched using Tcl string matching)
`-from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
`-hold`	Option to report clock hold paths
`-inter_clock`	Only report paths whose launch and latch clock are different
`-intra_clock`	Only report paths whose launch and latch clock are the same
`-less_than_slack <slack limit>`	Limit the paths reported to those with slack values less than the specified limit.
`-npaths <number>`	Specifies the number of paths to report (default=1, or the same value as nworst, if nworst is specified. Value of 0 causes all paths to be reported but be wary that this may be slow)
`-nworst <number>`	Specifies the maximum number of paths to report for each endpoint. If unspecified, there is no limit. If nworst is specified, but npaths is not, npaths defaults to the same value as nworst
`-pairs_only`	When set, paths with the same start and end points are considered equivalent. Only the worst case path for each unique combination is displayed.
`-panel_name <name>`	Report panel_name
`-recovery`	Option to report recovery paths
`-removal`	Option to report removal paths
`-rise_from <names>`	Valid sources (string patterns are matched using Tcl string matching)
`-rise_from_clock <names>`	Valid source clocks (string patterns are matched using Tcl string matching)
`-rise_through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
`-rise_to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
`-rise_to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)
`-setup`	Option to report clock setup paths
`-through <names>`	Valid through nodes (string patterns are matched using Tcl string matching)
`-to <names>`	Valid destinations (string patterns are matched using Tcl string matching)
`-to_clock <names>`	Valid destination clocks (string patterns are matched using Tcl string matching)

continued...

Description	Reports the worst-case paths and associated slack grouped by design entity. Each entity level indicates the worst-case slack and number of paths at that level. Use the "-setup", "-hold", "-recovery", or "-removal" options to specify which kind of analysis should be performed. Use the "-panel_name" option to direct the report to the Timing Analyzer graphical user interface. You can limit the analysis performed by this command to specific start and end points, using the "-from" and "-to" options. Use the "-rise_from" and "-fall_from" options to limit the analysis to endpoints with established high or low starting states. Use the "rise_to" and "fall_to" options to limit the analysis to destination points with high or low ending states. The analysis can be further limited to clocks using the "-from_clock" and "-to_clock" options, or to specific edges of the clock using the "-rise_from_clock", "-fall_from_clock", "-rise_to_clock", and "-fall_to_clock" options. Additionally, the "-through" option can be used to restrict analysis to paths which go through specified pins or nets. Use the "rise_through" and "fall_through" options to limit the analysis to intermediate points with high or low ending states. Use "-npaths" to limit the number of paths to report. If you do not specify this option, only the single worst-case path is provided. Use the "-less_than_slack" option to limit output to all paths with slack less than the specified value, up to the number specified by "-npaths". Use "-nworst" to limit the number of paths reported for each unique endpoint. If you do not specify this option, the number of paths reported for each destination node is bounded only by the "-npaths" option. If this option is used, but "-npaths" is not specified, then "-npaths" defaults to the same value specified for "-nworst". Use the "-pairs_only" option to filter the output further, restricting the results to only unique combinations of start and end points. The values of the "-from", "-to", and "-through" options are either collections or a Tcl list of wildcards used to create collections of appropriate types. The values used must follow standard Tcl or Timing Analyzer-extension substitution rules. See the help for use_timing_analyzer_style_escaping for details.		
Example Usage	This command currently contains no example usage.		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.77. report_ucp (::quartus::sta)

The following table displays information for the `report_ucp` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`report_ucp [-h | -help] [-long_help] [-append] [-file <name> ] [-panel_name <name> ] [-stdout] [-summary]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-append`	If output is sent to a file, this option appends the result to that file. Otherwise, the file is overwritten. This option is not supported for HTML files.
	`-file <name>`	Sends the results to an ASCII or HTML file. Depending on the extension

continued...

	-panel_name <*name*>	Sends the results to the panel and specifies the name of the new panel
	-stdout	Send output to stdout, via messages. You only need to use this option if you have selected another output format, such as a file, and would also like to receive messages.
	-summary	Generate everything except for the detailed paths panels.
Description		
	Reports unconstrained paths.	
Example Usage		
	project_open chiptrip create_timing_netlist read_sdc update_timing_netlist report_ucp delete_timing_netlist project_close	

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.78. set_operating_conditions (::quartus::sta)

The following table displays information for the set_operating_conditions Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	set_operating_conditions [-h | -help] [-long_help] [-force_dat] [-grade <*c|i|m|e|a*>] [-model <*fast|slow*>] [-speed <*speed*>] [-temperature <*value_in_C*>] [-voltage <*value_in_mV*>] [<*list_of_operating_conditions*>]	
Arguments	-h | -help	Short help
	-long_help	Long help with examples and possible return values
	-force_dat	Option to force delay annotation (only done when selecting an unanalyzed corner)
	-grade <*c|i|m|e|a*>	Option to specify temperature grade
	-model <*fast|slow*>	Option to specify timing model
	-speed <*speed*>	Speed grade
	-temperature <*value_in_C*>	Operating temperature
	-voltage <*value_in_mV*>	Operating voltage
	<*list_of_operating_conditions*>	list or collection of Operating conditions Tcl objects or names
Description		

```
Use this command to specify operating conditions different from the
initial conditions used to create the timing netlist. When a timing
model is not specified, the slow model is used.

Voltage and temperature options must be used together. These two
options are not available for all devices. The
get_available_operating_conditions command returns the list of
available operating conditions for your device.

Use the -speed option to analyze the design at a different speed grade
of the selected device.

Use the -grade option to analyze the design at a different temperature
grade. This option is provided to support what-if analysis and is not
```

continued...

	recommended for final sign-off analysis. By default, delay annotation is skipped if previously performed. Use -force_dat to rerun delay annotation.
Example Usage	#do report timing for different operating conditions one by one foreach_in_collection op [get_available_operating_conditions] { set_operating_conditions $op update_timing_netlist report_timing } #set aggregated report timing for all operating conditions when corner aggregation is enabled set_operating_conditions [get_available_operating_conditions] report_timing #manually set operating conditions set_operating_conditions -model fast -temperature 85 -voltage 1200 update_timing_netlist #change device speed grade and set operating conditions set_operating_conditions -speed 3 -model slow -temperature 0 -voltage 1100 update_timing_netlist

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Cannot set operating conditions for timing netlist created from XML file. Run create_timing_netlist.
TCL_ERROR	1	ERROR: Both the -temperature and -voltage options and their values are required.
TCL_ERROR	1	ERROR: The ability to select multiple operating conditions at once has been disabled.
TCL_ERROR	1	ERROR: Values entered did not match any valid operating conditions. Available operating conditions are: <*string*>
TCL_ERROR	1	ERROR: The <*string*> device family does not support set_operating_conditions command.
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
TCL_ERROR	1	ERROR: Illegal value: <*string*>. Specify an integer ranging from -999999999 to 999999999 for the option -voltage
TCL_ERROR	1	ERROR: Unsupported option: <*string*>.

3.1.37.79. timing_netlist_exist (::quartus::sta)

The following table displays information for the `timing_netlist_exist` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`timing_netlist_exist [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Checks if the timing netlist exists. Returns 1, if the timing netlist exists. Returns 0, otherwise.	

continued...

Example Usage	<pre>if {![timing_netlist_exist]} { create_timing_netlist }</pre>		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful

3.1.37.80. update_timing_netlist (::quartus::sta)

The following table displays information for the `update_timing_netlist` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`update_timing_netlist [-h | -help] [-long_help] [-dynamic_borrow] [-full] [-loop_aware_dynamic_borrow] [-no_borrow] [-recompute_borrow]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-dynamic_borrow`	Use time borrowing values that are correct for the current clock constraints
	`-full`	Forces creation of an updated timing netlist to ensure correctness
	`-loop_aware_dynamic_borrow`	Determine time borrowing values as with the -dynamic_borrow option with an addition loop-detection layer to prevent excessive time borrowing in sequential latch loops
	`-no_borrow`	Turn off all time borrowing
	`-recompute_borrow`	Recompute optimal time borrowing values
Description		

```
Updates and applies SDC commands to the timing netlist. The
update_timing_netlist command expands and validates generated clocks, warns
about sources in the design that require clock settings, identifies and removes
combinational loops, and warns about undefined input/output delays.

Most Tcl commands (e.g., report_timing) automatically update the timing netlist
when necessary. You can use the update_timing_netlist command explicitly to
control when updating occurs, or to force a full update using the -full option.

The update_timing_netlist command can also be used to control time borrowing
behavior. Time borrowing is a technique whereby certain flip-flops in certain
device families are allowed to have signals that arrive late (thus improving
upstream slack), at the expense of downstream slack. The amount of time
borrowing allowed at each flip-flop is hardware-dependent.

By default, optimal time borrowing values are computed at the end of the Fitter
(Finalize) stage (if enabled by your compilation settings), and these are the
values you will see in the timing reports. To turn off time borrowing support,
use the -no_borrow option. This is not recommended,    as it may result in
significantly pessimistic timing results.

If you have changed clock constraints after compiling your design, pre-computed
optimal time borrowing values may no longer be valid. Time borrowing is turned
off for the changed clocks, resulting in pessimistic timing. To get optimal
results once again, run update_timing_netlist with the -recompute_borrow option.
This may take significant time on large designs, but the results are saved and
available the next time you run update_timing_netlist without any time borrowing
options.

The time borrowing optimization algorithm has a few limitation - for example, it
never borrows any time on sources of cross-clock transfers, sources of paths
with set_max_delay or set_max_skew constraints, or in any clock domain
containing at least one level-sensitive latch. If your design is correctly
constrained, you may overcome these limitations and get better timing results
with the -dynamic_borrow option, which calculates time borrowing amounts based
on your actual clock constraints (rather than optimizing for highest FMax within
```

continued...

	each clock domain). Note that -dynamic_borrowing is not recommended for overconstrained designs. Using the -dynamic_borrow option may result in excessive time borrowing in sequential loops of level-sensitive latches, potentially leading to worse downstream slack. If your design contains such loops and you encounter this behavior, use the -loop_aware_dynamic_borrow option to detect sequential latch loops when evaluating their borrow amounts. Note that enabling this loop-detection feature may result in increased memory consumption and significantly increased runtime for large designs.
Example Usage	`project_open top` `create_timing_netlist` `read_sdc` `update_timing_netlist` `report_timing -to_clock clk1` `report_timing -to_clock clk2` `delete_timing_netlist` `project_close`

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.81. use_timing_analyzer_style_escaping (::quartus::sta)

The following table displays information for the `use_timing_analyzer_style_escaping` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`use_timing_analyzer_style_escaping [-h | -help] [-long_help] [-off] [-on]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-off`	Disable this setting.
	`-on`	Enable this setting.
Description	Use Timing Analyzer-style escaping. (Timing Analyzer-style escaping is enabled by default. The values used to create a collection, whether explicitly using a collection command or implicitly as a value specified as a "-from", "-to", or similar option to various SDC and report commands, are a Tcl list of wildcards. This includes a single name with an exact match. The value must follow standard Tcl substitution rules for Tcl lists and "string match" as described below, unless using Timing Analyzer-style escaping (default). For special characters such as '$', the character must be escaped using a single '\' character to prevent Tcl from interpreting the word after '$' as a Tcl variable, such as: Clk\$Signal. A '\' character itself must be escaped with another '\' as in the '$' case, must be escaped again for the Tcl list, and must be escaped yet again for Tcl "string match." The final result is eight '\' characters, such as: Clk\\\\\\\\Signal. Using Tcl "list" eliminates one level of escaping, since it escapes any '\' characters automatically for the Tcl list, such as: [list Clk\\\\Signal] Using '{' and '}' characters also eliminates the need for one or two levels of escaping, since '{' and '}' prevent string substitution in the contents, such as: [List {Clk\\Signal}]	

continued...

```
    {{Clk\\Signal}}

The use_timing_analyzer_style_escaping option, which is on by default,
allows the user to specify a name containing '\' characters with only
two '\' characters in all cases, such as: Clk\\Signal.  The extra '\'
characters required for Tcl list string substitution and "string
match" are added automatically by the Timing Analyzer.

To disable Timing Analyzer style string escaping, call
"use_timing_analyzer_style_escaping -off" before adding any timing
constraints or exceptions.
```

Example Usage

```
project_open top
use_timing_analyzer_style_escaping -on
create_timing_netlist
set res [get_cells my_test|special_\\reg]
query_collection $res -all

delete_timing_netlist
project_close
```

Return Value

Code Name	Code	String Return
TCL_OK	0	INFO: Operation successful
TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.

3.1.37.82. write_sdc (::quartus::sta)

The following table displays information for the `write_sdc` Tcl command:

Tcl Package and Version	Belongs to ::quartus::sta on page 615	
Syntax	`write_sdc [-h | -help] [-long_help] [-expand] [-history] [-valid_exceptions]` *<file_name>*	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
	`-expand`	Generate SDC file by expanding the macros
	`-history`	Reports full history of assignments
	`-valid_exceptions`	Generate SDC file containing only valid timing exceptions for debugging purposes
	<file_name>	Name of output file

Description

```
Generates an SDC file with all current constraints and
exceptions. When you use the -expand option, derive_clocks,
derive_pll_clocks, derive_lvds_clocks and derive_clock_uncertainty
macros are be expanded to corresponding sdc assignments before they
are written to a file. If you do not use the -expand option, these
macros are preserved.

Use the -valid_exceptions option to generate an SDC file that contains
only valid timing exceptions. Run the report_exceptions command with
the -valid option to see all the valid timing exceptions in your design.
```

Example Usage

```
project_new test
create_timing_netlist
create_clock -period 10 -name clk10 clk
set_multicycle_path -from [get_cells a] -to [get_cells b]
update_timing_netlist

report_timing

write_sdc my_sdc_file.sdc
```

continued...

	`delete_timing_netlist` `project_close`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: Clock manager is not up-to-date. Run update_timing_netlist to generate the latest clock manager.
	TCL_ERROR	1	ERROR: Timing netlist does not exist. Use create_timing_netlist to create a timing netlist.
	TCL_ERROR	1	ERROR: Open failed: *<string>*

3.1.38. ::quartus::stp

The following table displays information for the **::quartus::stp** Tcl package:

Tcl Package and Version	::quartus::stp 1.0
Description	This package contains the set of Tcl functions for acquiring Signal Tap data from the Intel device.
Availability	This package is loaded by default in the following executables: quartus_stp quartus_stp_tcl
Tcl Commands	close_session (::quartus::stp) on page 726 export_data_log (::quartus::stp) on page 727 open_session (::quartus::stp) on page 728 run (::quartus::stp) on page 729 run_multiple_end (::quartus::stp) on page 730 run_multiple_start (::quartus::stp) on page 731 stop (::quartus::stp) on page 731

3.1.38.1. close_session (::quartus::stp)

The following table displays information for the `close_session` Tcl command:

Tcl Package and Version	Belongs to ::quartus::stp on page 726	
Syntax	`close_session [-h | -help] [-long_help]`	
Arguments	`-h | -help`	Short help
	`-long_help`	Long help with examples and possible return values
Description	Saves the current session to the existing Signal Tap File (.stp).	
Example Usage	`#opens signaltap session` `open_session -name stp1.stp` `#capture data to log named log1, timeout after 5 seconds if no trigger occurs` `if { [catch {run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger trigger_1 -data_log log_1 -timeout 5} err_msg} {` `  # Timeout event is thrown as TCL exception` `  puts "ERROR: $err_msg"` `}` `#close signaltap session` `close_session`	

continued...

Return Value	Code Name	Code	String Return
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Session has been saved in Signal Tap File and closed
	TCL_ERROR	1	ERROR: Can't open Signal Tap File for writing. Make sure Signal Tap File exists, has write permission, and is not currently being used by another program.
	TCL_ERROR	1	ERROR: Session has not been opened. Make sure a session is open before attempting to close it.

3.1.38.2. export_data_log (::quartus::stp)

The following table displays information for the `export_data_log` Tcl command:

Tcl Package and Version	Belongs to ::quartus::stp on page 726		
Syntax	`export_data_log [-h | -help] [-long_help] [-clock_period` *<clock period>* `] [-data_log` *<data log>* `] -filename` *<export file name>* `[-format` *<export format>* `] [-instance` *<instance>* `] [-signal_set` *<signal set>* `] [-trigger` *<trigger>* `]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-clock_period` *<clock period>*		The file name of the exported file
	`-data_log` *<data log>*		Name of data log to be exported
	`-filename` *<export file name>*		The file name of the exported file
	`-format` *<export format>*		File format of the exported file
	`-instance` *<instance>*		Name of instance that defines data log
	`-signal_set` *<signal set>*		Name of signal set that defines data log
	`-trigger` *<trigger>*		Name of trigger that defines data log
Description			

```
       Exports the specified data log from the current open
       session into another file in different format.

If a data log is not explicitly specified, the last
       active one is used.

The supported file formats are Comma Separated Value
file (.csv), Value Change Dump file (.vcd), and Table
file (.tbl).
```

Example Usage

```
#opens signaltap session
open_session -name stp1.stp

#capture data to log named log1, timeout after 5 seconds if no trigger occurs
if { [catch {run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger trigger_1 -
data_log log_1 -timeout 5} err_msg} {
    # Timeout event is thrown as TCL exception
    puts "ERROR: $err_msg"
}

#export data into a VCD file
export_data_log -instance auto_signaltap_0 -signal_set signal_set_1 -trigger trigger_1 -
data_log log_1 -filename log_1.vcd -format vcd

#close signaltap session
close_session
```

Return Value	Code Name	Code	String Return

continued...

	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Data has been acquired successfully
	TCL_ERROR	1	ERROR: Error occured when the specified data log was exported to a file.
	TCL_ERROR	1	ERROR: Instance, signal set, or trigger does not exist. Make sure the instance, signal set, and trigger exist in the Signal Tap File.
	TCL_ERROR	1	ERROR: Session has not been opened. Make sure a session is open before attempting to close it.

3.1.38.3. open_session (::quartus::stp)

The following table displays information for the open_session Tcl command:

Tcl Package and Version	Belongs to ::quartus::stp on page 726		
Syntax	`open_session [-h | -help] [-long_help] -name <.stp file name>`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
	`-name <.stp file name>`		Signal Tap File (.stp) name
Description	`Opens a session from the specified Signal Tap File (.stp).`		
Example Usage	`#opens signaltap session` `open_session -name stp1.stp` `#capture data to log named log1, timeout after 5 seconds if no trigger occurs` `if { [catch {run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger trigger_1 -data_log log_1 -timeout 5} err_msg} {` `  # Timeout event is thrown as TCL exception` `  puts "ERROR: $err_msg"` `}` `#close signaltap session` `close_session`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Session has been opened from Signal Tap File
	TCL_ERROR	1	ERROR: Can't open Signal Tap File for reading. Make sure the Signal Tap File exists and has read permission.
	TCL_ERROR	1	ERROR: Session already open. Close session before attempting to open it again.
	TCL_ERROR	1	ERROR: Signal Tap File contains syntax error. Make sure the Signal Tap File is formatted correctly before opening. Intel recommends that you do not manually edit Signal Tap Files, but use the Signal Tap dialog boxes in the Quartus Prime GUI.

3.1.38.4. run (::quartus::stp)

The following table displays information for the `run` Tcl command:

Tcl Package and Version	Belongs to ::quartus::stp on page 726		
Syntax	`run [-h | -help] [-long_help] [-bridge <bridge> ] [-data_log <data log> ] [-device_name <device name> ] [-hardware_name <hardware name> ] [-instance <instance> ] [-signal_set <signal set> ] [-timeout <timeout> ] [-trigger <trigger> ]`		
Arguments	`-h | -help`	Short help	
	`-long_help`	Long help with examples and possible return values	
	`-bridge <bridge>`	Bridge to use instead of the one specified in the stp file	
	`-data_log <data log>`	Name of data log to be recorded	
	`-device_name <device name>`	Device to use instead of the one specified in the stp file. Tcl command, get_device_names, can be used to obtain the valid hardware names	
	`-hardware_name <hardware name>`	JTAG programming hardware to use instead of the one specified in the stp file. Tcl command, get_hardware_names, can be used to obtain the valid hardware name	
	`-instance <instance>`	Name of instance that defines data acquisition	
	`-signal_set <signal set>`	Name of signal set that defines data acquisition	
	`-timeout <timeout>`	Timeout period for data acquisition in seconds	
	`-trigger <trigger>`	Name of trigger that defines data acquisition	
Description			

```
Starts data acquisition with the specified conditions in
the session and saves data into the specified data log
within the timeout period.
```

Example Usage

```
#opens signaltap session
open_session -name stp1.stp

#capture data to log named log1, timeout after 5 seconds if no trigger occurs
if { [catch {run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger trigger_1 -
data_log log_1 -timeout 5} err_msg} {
    # Timeout event is thrown as TCL exception
    puts "ERROR: $err_msg"
}

#close signaltap session
close_session
```

Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Data has been acquired successfully
	TCL_ERROR	1	ERROR: JTAG chain in use. Wait for JTAG communication to finish and run again.
	TCL_ERROR	1	ERROR: Data acquisition stopped unexpectedly. Make sure device is stable and run again.
	TCL_ERROR	1	ERROR: Trigger not compatible with device. Download a design with the current SRAM Object File after recompiling.

continued...

	TCL_ERROR	1	ERROR: Instance, signal set, or trigger does not exist. Make sure the instance, signal set, and trigger exist in the Signal Tap File.
	TCL_ERROR	1	ERROR: Session has not been opened. Make sure a session is open before attempting to close it.
	TCL_ERROR	1	ERROR: Trigger did not occur in timeout period. Make sure trigger conditions are valid and/or increase timeout period.

3.1.38.5. run_multiple_end (::quartus::stp)

The following table displays information for the `run_multiple_end` Tcl command:

Tcl Package and Version	Belongs to ::quartus::stp on page 726		
Syntax	`run_multiple_end [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Defines the end of a set of "run" commands. This command is used when multiple instances of data acquisition are started simultaneously. Add "run_multiple_start" before the set of "run" commands that specify data acquisition. Add this command after the set of commands.` `If "run_multiple_end" is not included, the "run" commands do not execute.`		
Example Usage	`#opens signaltap session` `open_session -name stp1.stp` `#start acquisition of instance auto_signaltap_0 and auto_signaltap_1 at the same time` `#calling run_multiple_end will start all instances run after run_multiple_start call run_multiple_start` `run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger trigger_1 -data_log log_1 -timeout 5` `run -instance auto_signaltap_1 -signal_set signal_set_1 -trigger trigger_1 -data_log log_1 -timeout 5` `if { [catch {run_multiple_end} err_msg} {` `  # Timeout event is thrown as TCL exception` `  puts "ERROR: $err_msg"` `}` `#close signaltap session` `close_session`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Multiple instances of data acquisition ended successfully
	TCL_ERROR	1	ERROR: Data acquisition stopped unexpectedly. Make sure device is stable and run again.
	TCL_ERROR	1	ERROR: Run multiple instances has not been started. Use run_multiple_start before using run_multiple_end.
	TCL_ERROR	1	ERROR: Session has not been opened. Make sure a session is open before attempting to close it.

3.1.38.6. run_multiple_start (::quartus::stp)

The following table displays information for the `run_multiple_start` Tcl command:

Tcl Package and Version	Belongs to ::quartus::stp on page 726		
Syntax	`run_multiple_start [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Defines the start of a set of "run" commands. This command is used when multiple instances of data acquisition are started simultaneously. Add this command before the set of "run" commands that specify data acquisition. Add "run_multiple_end" after the set of commands.` `If "run_multiple_end" is not included, the "run" commands do not execute.`		
Example Usage	`#opens signaltap session` `open_session -name stp1.stp` `#start acquisition of instance auto_signaltap_0 and auto_signaltap_1 at the same time` `#calling run_multiple_end will start all instances run after run_multiple_start call` `run_multiple_start` `run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger trigger_1 -data_log log_1 -timeout 5` `run -instance auto_signaltap_1 -signal_set signal_set_1 -trigger trigger_1 -data_log log_1 -timeout 5` `if { [catch {run_multiple_end} err_msg} {` `  # Timeout event is thrown as TCL exception` `  puts "ERROR: $err_msg"` `}` `#close signaltap session` `close_session`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Multiple instances of data acquisition started
	TCL_ERROR	1	ERROR: Run multiple instances has not ended. Use run_multiple_end to complete an active call to run_multiple_start before using run_multiple_start again.
	TCL_ERROR	1	ERROR: Session has not been opened. Make sure a session is open before attempting to close it.

3.1.38.7. stop (::quartus::stp)

The following table displays information for the `stop` Tcl command:

Tcl Package and Version	Belongs to ::quartus::stp on page 726		
Syntax	`stop [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Stops all data acquisition.`		

continued...

Example Usage	`stop`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_OK	0	INFO: Data acquisition has stopped
	TCL_ERROR	1	ERROR: No data acquisition was started. Data acquisition must be in progress before stopping.

3.1.39. ::quartus::tdc

The following table displays information for the **::quartus::tdc** Tcl package:

Tcl Package and Version	::quartus::tdc 1.0
Description	This package contains the set of Tcl functions for obtaining information from the Timing Analyzer.
Availability	This package is loaded by default in the following executables: qpro quartus quartus_fit quartus_map quartus_pow quartus_sta quartus_syn
Tcl Commands	is_place (::quartus::tdc) on page 732 is_plan (::quartus::tdc) on page 733 is_post_route (::quartus::tdc) on page 733

3.1.39.1. is_place (::quartus::tdc)

The following table displays information for the `is_place` Tcl command:

Tcl Package and Version	Belongs to ::quartus::tdc on page 732		
Syntax	`is_place [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	Returns true when called from Fitter during the placer. (Only supported in Quartus Prime Pro Edition)		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: This command is not supported in the current software edition.

3.1.39.2. is_plan (::quartus::tdc)

The following table displays information for the `is_plan` Tcl command:

Tcl Package and Version	Belongs to ::quartus::tdc on page 732		
Syntax	`is_plan [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns true when called from Fitter during the plan.` `(Only supported in Quartus Prime Pro Edition)`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: This command is not supported in the current software edition.

3.1.39.3. is_post_route (::quartus::tdc)

The following table displays information for the `is_post_route` Tcl command:

Tcl Package and Version	Belongs to ::quartus::tdc on page 732		
Syntax	`is_post_route [-h | -help] [-long_help]`		
Arguments	`-h | -help`		Short help
	`-long_help`		Long help with examples and possible return values
Description	`Returns true when called from Fitter after router has completed or when post-fitting delays are annotated.` `(Only supported in Quartus Prime Pro Edition)`		
Example Usage	`This command currently contains no example usage.`		
Return Value	**Code Name**	**Code**	**String Return**
	TCL_OK	0	INFO: Operation successful
	TCL_ERROR	1	ERROR: No project is currently open. Open an existing project or create a new project.
	TCL_ERROR	1	ERROR: This command is not supported in the current software edition.

3.2. Tcl Commands and Packages Revision History

The following revision history applies to this chapter:

Table 13. Document Revision History

Document Version	Quartus Prime Version	Changes
2024.04.01	24.1	• Applied phase I Altera rebranding throughout. • Updated for latest Tcl commands and packages support.
2023.12.04	23.4	• Updated for latest Tcl commands and packages support.
2023.10.02	23.3	• Updated for latest Tcl commands and packages support.
2023.06.26	23.2	• Updated for latest Tcl commands and packages support.
2023.04.03	23.1	• Updated name of Intel Agilex 7 device family. • Updated for latest Tcl commands and packages support.
2022.12.12	22.4	• Updated for latest Tcl commands and packages support.
2022.09.26	22.3	• Updated for latest Tcl commands and packages support.
2022.06.20	22.2	• Updated for latest Tcl commands and packages support.
2022.03.28	22.1	• Updated for latest Tcl commands and packages support.
2021.12.13	21.4	• Updated for latest Tcl commands and packages support.
2021.10.04	21.3	• Updated for latest Tcl commands and packages support.
2021.03.29	21.1	• Updated for latest Tcl commands and packages support.
2020.12.14	20.4	Initial release of *Tcl Commands and Packages* reference chapter.

4. Quartus Prime Pro Edition User Guide Scripting Archives

For the latest and previous versions of this user guide, refer to *Quartus Prime Pro Edition User Guide: Scripting*. If a software version is not listed, the guide for the previous software version applies.

A. Quartus Prime Pro Edition User Guides

Refer to the following user guides for comprehensive information on all phases of the Quartus Prime Pro Edition FPGA design flow.

Related Information

- Quartus Prime Pro Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Pro Edition software, including managing Quartus Prime Pro Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Pro Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer, a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Pro Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Pro Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Pro Edition synthesis optimally implements your design in hardware.
- Quartus Prime Pro Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Pro Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Pro Edition User Guide: Design Optimization
  Describes Quartus Prime Pro Edition settings, tools, and techniques that you can use to achieve the highest design performance in Intel® FPGAs. Techniques include optimizing the design netlist, addressing critical chains that limit retiming and timing closure, optimizing device resource usage, device floorplanning, and implementing engineering change orders (ECOs).
- Quartus Prime Pro Edition User Guide: Programmer
  Describes operation of the Quartus Prime Pro Edition Programmer, which allows you to configure Intel® FPGA devices, and program CPLD and configuration devices, via connection with an Intel® FPGA download cable.
- Quartus Prime Pro Edition User Guide: Block-Based Design
  Describes block-based design flows, also known as modular or hierarchical design flows. These advanced flows enable preservation of design blocks (or logic that comprises a hierarchical design instance) within a project, and reuse of design blocks in other projects.

- Quartus Prime Pro Edition User Guide: Partial Reconfiguration
  Describes Partial Reconfiguration, an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. Define multiple personas for a particular design region, without impacting operation in other areas.
- Quartus Prime Pro Edition User Guide: Third-party Simulation
  Describes RTL- and gate-level design simulation support for third-party simulation tools by Aldec*, Cadence*, Siemens EDA, and Synopsys that allow you to verify design behavior before device programming. Includes simulator support, simulation flows, and simulating Intel® FPGA IP.
- Quartus Prime Pro Edition User Guide: Third-party Synthesis
  Describes support for optional synthesis of your design in third-party synthesis tools by Siemens EDA, and Synopsys. Includes design flow steps, generated file descriptions, and synthesis guidelines.
- Quartus Prime Pro Edition User Guide: Third-party Logic Equivalence Checking Tools
  Describes support for optional logic equivalence checking (LEC) of your design in third-party LEC tools by OneSpin*.
- Quartus Prime Pro Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Pro Edition in-system design debugging tools for real-time verification of your design. These tools provide visibility by routing (or "tapping") signals in your design to debugging logic. These tools include System Console, Signal Tap logic analyzer, system debugging toolkits, In-System Memory Content Editor, and In-System Sources and Probes Editor.
- Quartus Prime Pro Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Pro Edition Timing Analyzer, a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology.
- Quartus Prime Pro Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Pro Edition Power Analysis tools that allow accurate estimation of device power consumption. Estimate the power consumption of a device to develop power budgets and design power supplies, voltage regulators, heat sink, and cooling systems.
- Quartus Prime Pro Edition User Guide: Design Constraints
  Describes timing and logic constraints that influence how the Compiler implements your design, such as pin assignments, device options, logic options, and timing constraints. Use the Interface Planner to prototype interface implementations, plan clocks, and quickly define a legal device floorplan. Use the Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Pro Edition User Guide: PCB Design Tools
  Describes support for optional third-party PCB design tools by Siemens EDA and Cadence*. Also includes information about signal integrity analysis and simulations with HSPICE and IBIS Models.
- Quartus Prime Pro Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Pro Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.