Auto-generated Date: August 18, 2025

Intel® Quartus® Prime Standard 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.

- Getting Started (PDF | HTML)
- Platform Designer (PDF | HTML)
- Design Recommendations (PDF | HTML)
- Design Compilation (PDF | HTML)
- Design Optimization (PDF | HTML)
- Programmer (PDF | HTML)
- Partial Reconfiguration (PDF | HTML)
- Third-party Simulation (PDF | HTML)
- Third-party Synthesis (PDF | HTML)
- Debug Tools (PDF | HTML)
- Timing Analyzer (PDF | HTML)
- Power Analysis and Optimization (PDF | HTML)
- Design Constraints (PDF | HTML)
- PCB Design Tools (PDF | HTML)
- Scripting (PDF | HTML)

Intel® Quartus® Prime Standard Edition User Guide

Getting Started

Updated for Intel® Quartus® Prime Design Suite: **19.4**

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

Contents

1. Introduction to Intel® Quartus® Prime Standard Edition.....5
1.1. Selecting an Intel Quartus Prime Software Edition.....7
1.2. Introduction to Intel Quartus Prime Standard Edition Revision History.....8
2. Managing Intel Quartus Prime Projects..... 9
2.1. Viewing Basic Project Information..... 10
2.1.1. Viewing Project Reports..... 11
2.1.2. Viewing Project Messages..... 12
2.1.3. Automated Problem Reports..... 13
2.2. Intel Quartus Prime Project Contents.....13
2.2.1. Project File Best Practices.....14
2.3. Managing Project Settings.....14
2.3.1. Specifying the Target Device or Board.....15
2.3.2. Optimizing Project Settings..... 17
2.4. Managing Logic Design Files..... 19
2.4.1. Including Design Libraries..... 20
2.4.2. Creating a Project Copy..... 20
2.5. Managing Timing Constraints.....20
2.6. Integrating Other EDA Tools..... 21
2.7. Exporting Compilation Results..... 22
2.7.1. Exporting a Version-Compatible Compilation Database 23
2.7.2. Importing a Version-Compatible Compilation Database 23
2.7.3. Exporting a Design Partition..... 23
2.7.4. Clearing Compilation Results..... 24
2.8. Migrating Projects Across Operating Systems.....25
2.8.1. Migrating Design Files and Libraries.....25
2.8.2. Design Library Migration Guidelines..... 26
2.9. Archiving Projects..... 27
2.9.1. Manually Adding Files To Archives..... 27
2.9.2. Archiving Compilation Results.....28
2.9.3. Archiving Projects for Service Requests.....28
2.9.4. Using External Revision Control.....29
2.10. Command-Line Interface.....30
2.10.1. Project Revision Commands.....30
2.10.2. Project Archive Commands..... 31
2.10.3. Project Database Commands..... 31
2.10.4. Project Library Commands.....32
2.11. Managing Projects Revision History.....33
3. Design Planning..... 35
3.1. Design Planning.....35
3.2. Create a Design Specification and Test Plan..... 35
3.3. Plan for the Target Device..... 35
3.3.1. Device Migration Planning..... 37
3.4. Plan for Intellectual Property Cores..... 37
3.5. Plan for Standard Interfaces.....38
3.6. Plan for Device Programming.....38
3.7. Plan for Device Power Consumption.....39

3.8. Plan for Interface I/O Pins....41
3.8.1. Simultaneous Switching Noise Analysis....42
3.9. Plan for other EDA Tools....43
3.9.1. Third-Party Synthesis Tools....43
3.9.2. Third-Party Simulation Tools....43
3.10. Plan for On-Chip Debugging Tools....43
3.11. Plan HDL Coding Styles....44
3.11.1. Design Recommendations....45
3.11.2. Recommended HDL Coding Styles....45
3.11.3. Managing Metastability....45
3.12. Plan for Hierarchical and Team-Based Designs....46
3.12.1. Flat Compilation without Design Partitions....46
3.12.2. Incremental Compilation with Design Partitions....47
3.12.3. Planning Design Partitions and Floorplan Location Assignments....47
3.13. Design Planning Revision History....48

4. Introduction to Intel FPGA IP Cores....51
4.1. IP Catalog and Parameter Editor....52
4.1.1. The Parameter Editor....52
4.2. Installing and Licensing Intel FPGA IP Cores....53
4.2.1. Intel FPGA IP Evaluation Mode....54
4.3. IP General Settings....57
4.4. Adding Your Own IP to IP Catalog....57
4.5. Best Practices for Intel FPGA IP....59
4.6. Generating IP Cores (Intel Quartus Prime Standard Edition)....60
4.6.1. IP Core Generation Output (Intel Quartus Prime Standard Edition)....61
4.7. Modifying an IP Variation....62
4.8. Upgrading IP Cores....63
4.8.1. Upgrading IP Cores at Command-Line....65
4.8.2. Migrating IP Cores to a Different Device....66
4.8.3. Troubleshooting IP or Platform Designer System Upgrade....67
4.9. Simulating Intel FPGA IP Cores....69
4.9.1. Generating IP Simulation Files....69
4.9.2. Using NativeLink Simulation (Intel Quartus Prime Standard Edition)....70
4.10. Synthesizing IP Cores in Other EDA Tools....72
4.11. Instantiating IP Cores in HDL....72
4.11.1. Example Top-Level Verilog HDL Module....72
4.11.2. Example Top-Level VHDL Module....72
4.12. Introduction to Intel FPGA IP Cores Revision History....73

5. Migrating to Intel Quartus Prime Pro Edition....74
5.1. Keep Pro Edition Project Files Separate....74
5.2. Upgrade Project Assignments and Constraints....74
5.2.1. Modify Entity Name Assignments....75
5.2.2. Resolve Timing Constraint Entity Names....75
5.2.3. Verify Generated Node Name Assignments....76
5.2.4. Replace Logic Lock (Standard) Regions....76
5.2.5. Modify Signal Tap Logic Analyzer Files....78
5.2.6. Remove References to .qip Files....79
5.2.7. Remove Unsupported Feature Assignments....79
5.3. Upgrade IP Cores and Platform Designer (Standard) Systems....80

5.4. Upgrade Non-Compliant Design RTL..... 81
5.4.1. Verify Verilog Compilation Unit 81
5.4.2. Update Entity Auto-Discovery..... 82
5.4.3. Ensure Distinct VHDL Namespace for Each Library.....83
5.4.4. Remove Unsupported Parameter Passing.....83
5.4.5. Remove Unsized Constant from WYSIWYG Instantiation..... 83
5.4.6. Remove Non-Standard Pragmas.....84
5.4.7. Declare Objects Before Initial Values..... 84
5.4.8. Confine SystemVerilog Features to SystemVerilog Files..... 84
5.4.9. Avoid Assignment Mixing in Always Blocks.....85
5.4.10. Avoid Unconnected, Non-Existent Ports..... 85
5.4.11. Avoid Illegal Parameter Ranges..... 85
5.4.12. Update Verilog HDL and VHDL Type Mapping.....86
5.5. Migrating to Intel Quartus Prime Pro Edition Revision History..... 86

A. Intel Quartus Prime Pro Edition User Guide: Getting Started Documentation Archive...87

B. Intel Quartus Prime Standard Edition User Guides.....88

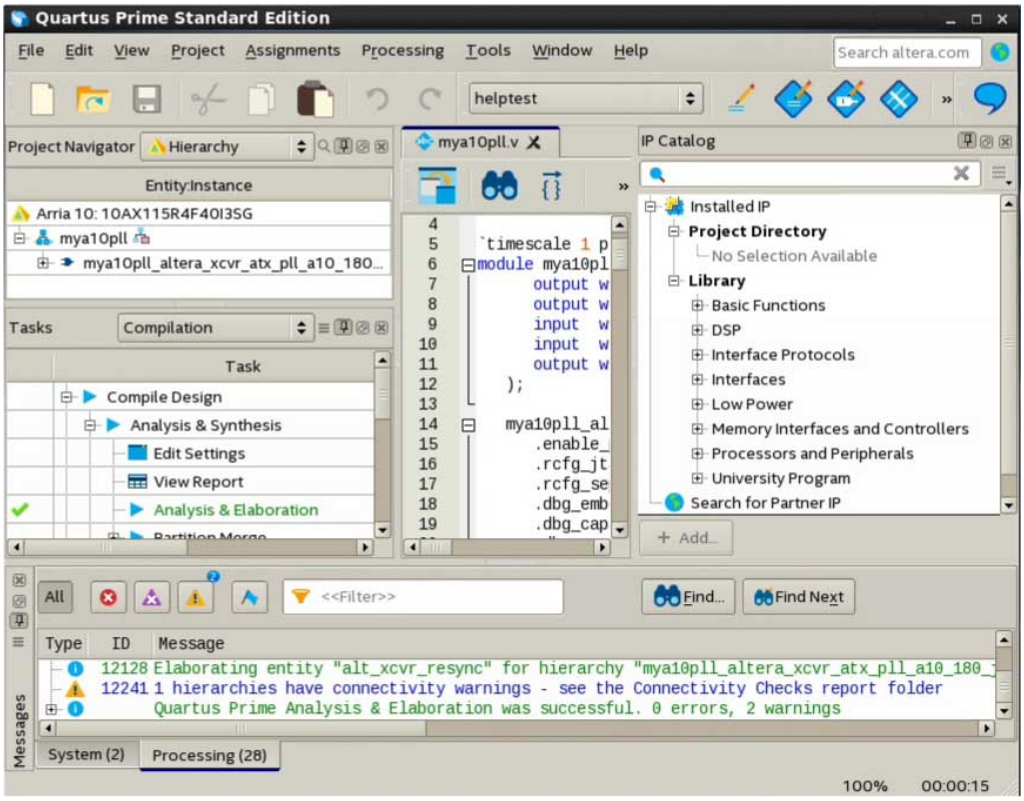
1. Introduction to Intel® Quartus® Prime Standard Edition

This user guide describes basic concepts and operation of the Intel® Quartus® Prime Standard Edition design software, including GUI and project structure basics, initial design planning, use of Intel FPGA IP, and migration to Intel Quartus Prime Pro Edition. The Intel Quartus Prime Standard Edition software provides a complete design environment for the following device families:

- Intel Arria® 10, Arria V, and Arria II
- Intel Cyclone® 10 LP, Cyclone IV, and Cyclone V
- MAX® series

The Intel Quartus Prime software GUI supports easy design entry, 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.

Figure 1. Intel Quartus Prime Standard Edition Software GUI



Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

The Intel Quartus Prime Standard Edition software offers a full range of features at each phase of the design flow to shorten your design cycle and achieve the highest performance:

- Easy Project Setup—quickly create a new project, add design files, and specify the target Intel device with the New Project Wizard. Create different revisions of your project to compare results with different settings. Save the current state of your project and project files as a single, compressed file. Refer to Managing Intel Quartus Prime Projects on page 9 for more information.
- Design Planning Tools— plan for initial I/O pin layout, power consumption, and area utilization in the Early Power Estimator, the Power Analyzer Tool, and the Pin Planner. Refer to Design Planning on page 35 for more information.
- Design Constraint Entry—specify timing, placement, and other constraints with the **Settings** dialog box, Assignment Editor, Pin Planner, and Timing Analyzer. Visualize and modify logic placement within a view of the device floorplan in the Chip Planner and Timing Closure Floorplan. Refer to Intel Quartus Prime Standard Edition User Guide: Design Constraints for more information.
- Integrated Synthesis—provides efficient synthesis support for VHDL (1987, 1993, 2008), Verilog HDL (1995, 2001), and SystemVerilog (2005) design entry languages. Refer to Intel Quartus Prime Standard Edition User Guide: Compiler for more information.
- Incremental Compilation—preserve the results and performance for unchanged logic in your design as you make changes elsewhere, facilitating top-down or bottom-up team-based design methodologies. Refer to Intel Quartus Prime Standard Edition User Guide: Compiler for more information.
- Optimizing Results—Design Space Explorer automatically determines the best combination of settings for your design. Design Assistant validates your project against predetermined design rules for gated clocks, reset signals, asynchronous design practices, and signal race conditions. Refer to Intel Quartus Prime Standard Edition User Guide: Design Optimization for more information.
- Design Debugging—The Signal Tap logic analyzer captures and displays real-time signal behavior in an FPGA design, allowing to examine the behavior of internal signals during normal device operation without the need for extra I/O pins or external lab equipment. The Transceiver Toolkit provides real-time control, monitoring, and debugging of the transceiver links running on your board. Refer to Intel Quartus Prime Standard Edition User Guide: Debug Tools for more information.
- System and IP Integration—define and generate a complete system in much less time than using traditional, manual integration methods with Platform Designer (Standard). Refer to Introduction to Intel FPGA IP Cores on page 51 and Intel Quartus Prime Standard Edition User Guide: Platform Designer for more information.
- Third-party EDA Tool Support—integrate with supported versions of third-party EDA synthesis, simulation, and board-level timing analysis tools. Refer to Third-Party Simulation and Third-Party Synthesis user guides for more information.

The Intel Quartus Prime Pro Edition software expands on these capabilities of the Intel Quartus Prime Standard Edition, and provides unique features that support the latest Intel FPGAs. Select the Intel Quartus Prime software edition that provides the device support and features you require, as Selecting an Intel Quartus Prime Software Edition on page 7 describes.

1.1. Selecting an Intel Quartus Prime Software Edition

Depending on your device support and software feature requirements, you can choose either the Intel Quartus Prime Pro Edition or Intel Quartus Prime Standard Edition software for your design. Consider the requirements and timeline of your project in determining whether to select the Intel Quartus Prime Standard Edition or Intel Quartus Prime Pro Edition software:

- Select the Intel Quartus Prime Pro Edition if you are beginning a new Intel Arria 10, Intel Cyclone 10 GX, Intel Stratix® 10 or Intel Agilex™ design, or to take advantage of the unique features of Intel Quartus Prime Pro Edition.

Figure 2. Intel Quartus Prime Feature Support Matrix

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

- Select the Intel Quartus Prime Standard Edition software if your design must target Arria V, Arria, Intel Cyclone 10 LP, Cyclone IV, Cyclone V, or MAX series devices, and you do not want to migrate you design to a device that Intel Quartus Prime Pro Edition supports.
- Intel Quartus Prime Pro Edition software does not support the following Intel 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`

Note: Intel replaces the following Altera tool names in the Intel Quartus Prime software:

Table 1. Intel Quartus Prime Tool Name Updates

Altera Name	Intel Name
Qsys	Platform Designer
TimeQuest	Timing Analyzer
EyeQ	Eye Viewer
JNEye	Advanced Link Analyzer

Related Information

Migrating to Intel Quartus Prime Pro Edition on page 74

1.2. Introduction to Intel Quartus Prime Standard Edition Revision History

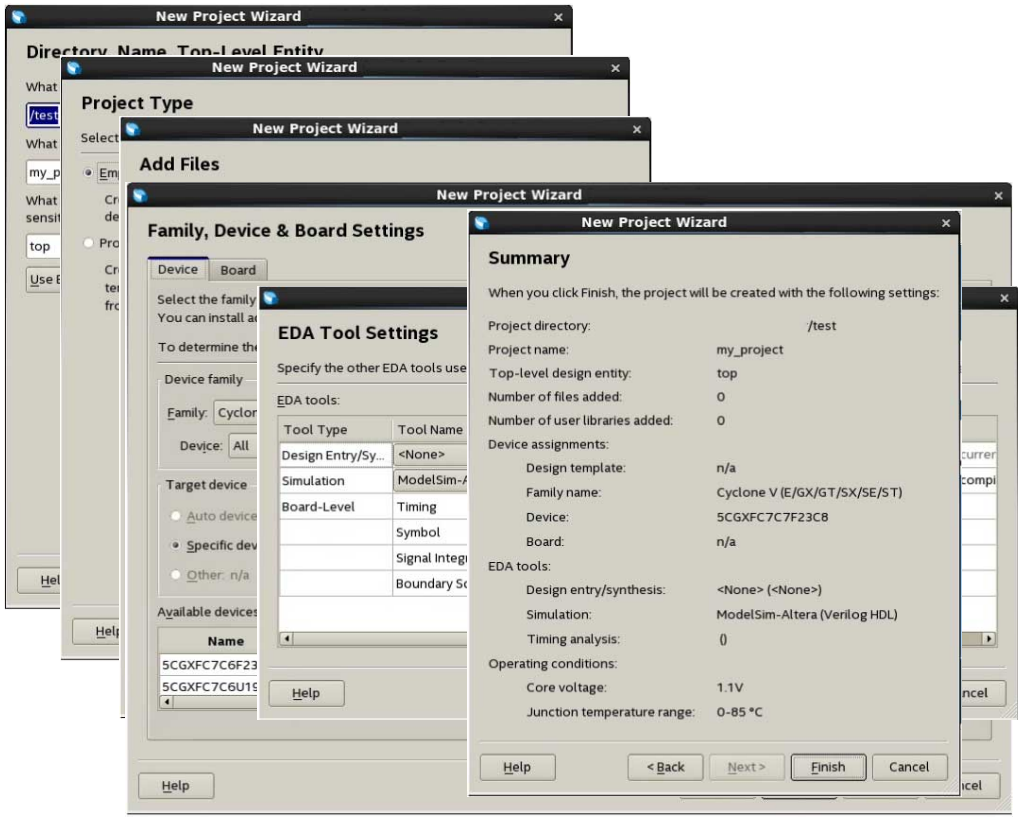
Document Version	Intel Quartus Prime Version	Changes
2019.12.16	19.4.0	• Added programming file generation support for Intel Agilex devices.
2018.09.24	18.1.0	Initial release for Intel Quartus Prime Standard Edition.

2. Managing Intel Quartus Prime Projects

The Intel 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 Intel Quartus Prime software projects, including initial project setup, viewing project information, adding design files and constraints, and exporting compilation results.

Click **File > New Project Wizard** to quickly setup and open a new project.

Figure 3. New Project Wizard



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

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

2.1. Viewing Basic Project Information

You can view basic information about your project in the **Tasks** pane, Project Navigator, Report panel, and **Messages** window.

Project Tasks Pane

The **Tasks** pane (**View ➤ Tasks**) provides one-click launch of common project tasks, such as creating design files, specifying project settings, running compilation, debug and timing closure, and device programming.

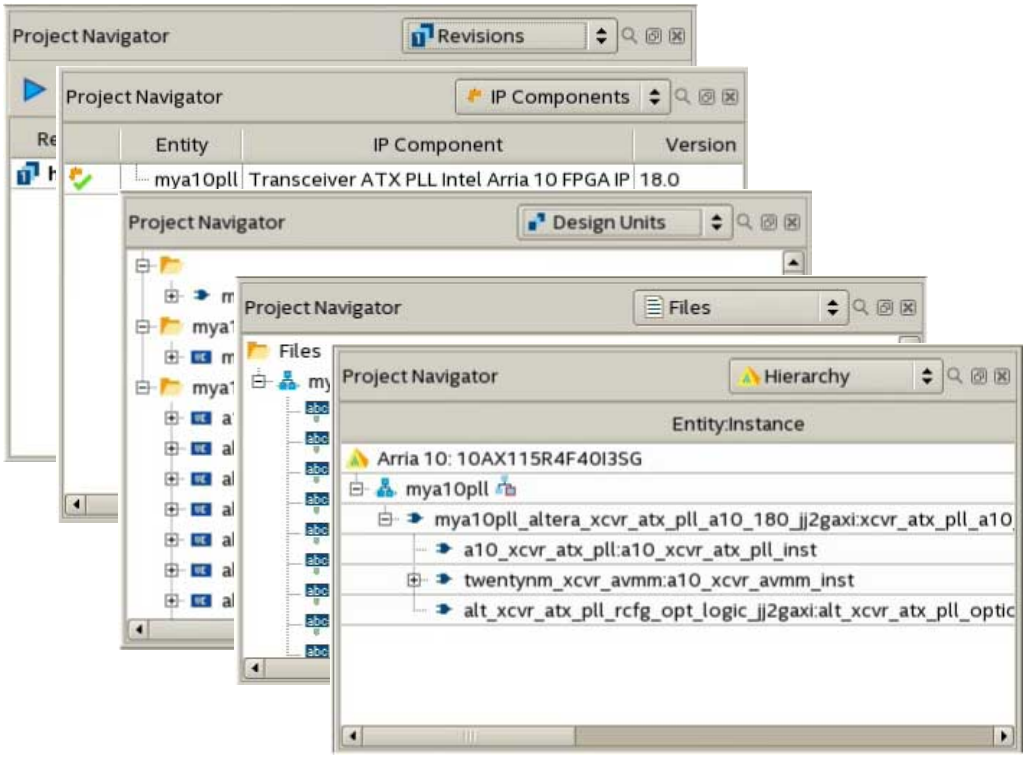
The Project Navigator

The **Project Navigator** (**View ➤ Utility Windows ➤ Project Navigator**) displays information about the elements of your project, such as the design files, IP components, and your project hierarchy (after elaboration). You can right-click items in the **Project Navigator** to locate or perform actions on the elements of your project. The Project Navigator organizes information on the **Files**, **Hierarchy**, **Design Units**, and **IP Components** tabs.

Table 2. 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** • **Create AHDL Include Files for Current File** • **Create Symbol Files for Current File** • **Create Verilog Instantiation Template Files for Current File** • **Create VHDL Component Declaration Files for Current File**
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 (Standard) components, and third-party IP. Click **Launch IP Upgrade Tool** from this tab to upgrade outdated IP components. Right-click any IP component to **Edit in Parameter Editor**.
Revisions	Displays the current project revisions.

Figure 4. Project Navigator Hierarchy, Files, Design Units, IP Components, and Revisions Tabs

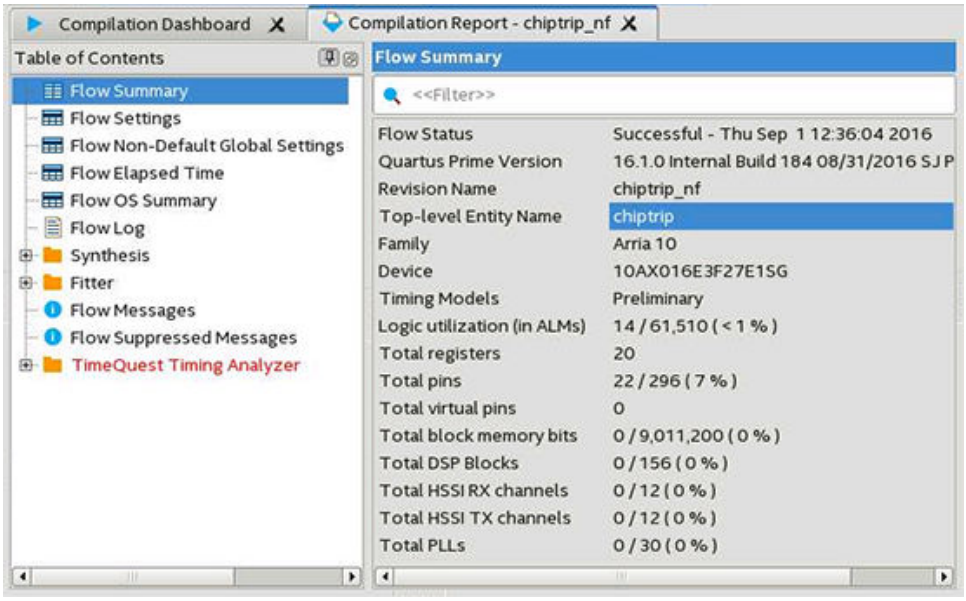


2.1.1. Viewing Project Reports

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 5. Compilation Report

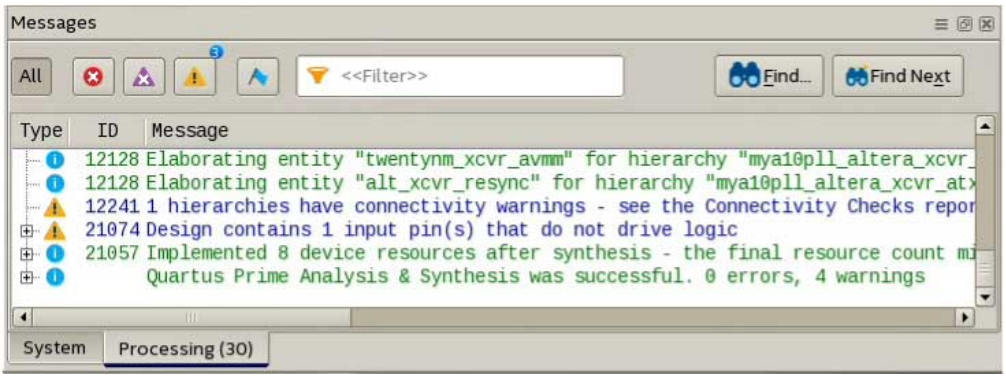


2.1.2. Viewing Project Messages

The Messages window (**View ➤ Utility Windows ➤ Messages**) displays information, warning, and error messages about Intel 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
- **Search**—locates specific messages

Figure 6. Messages Window



2.1.2.1. 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.

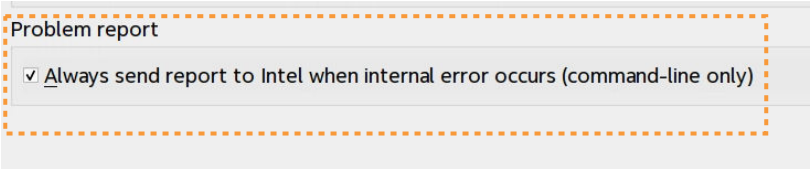
2.1.3. Automated Problem Reports

By default, the **Problem report** feature automatically sends a text based problem report to Intel whenever an internal error occurs in the Intel Quartus Prime software. You can disable **Problem report** to stop sending problem reports.

To disable or enable automatic sending of problem reports, follow these steps:

1. Click **Tools ➤ Options**.
2. Click the **Internet Connectivity** tab.
3. Under **Problem report**, turn on or off **Always send report to Intel when internal error occurs (command-line only)**.

Figure 7. Problem Report Option



2.2. Intel Quartus Prime Project Contents

The Intel 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 Intel Quartus Prime software automatically creates an Intel Quartus Prime Project File (`.qpf`) for that project. The `.qpf` references the Intel 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 Intel Quartus Prime software creates and updates these files automatically as you make changes in the GUI.

Table 3. Intel Quartus Prime Project Files

File Type	Contains	To Edit	Format
Project file	Project and revision name	**File ➤ New Project Wizard**	Intel Quartus Prime Project File (`.qpf`)
Settings file	Lists design files, entity settings, target device, synthesis directives, placement constraints	**Assignments ➤ Settings**	Intel Quartus Prime Settings File (`.qsf`)
Quartus database	Project compilation results	**Project ➤ Export Database**	`export_db` directory

continued...

File Type	Contains	To Edit	Format
Partition database	Partition compilation results	**Project ➤ Export Design Partition**	Exported Partition File (`.qxp`)
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**	Intel Quartus Prime IP File (`.qip`)
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**	Intel Quartus Prime Archive File (`.qar`)

2.2.1. Project File Best Practices

The Intel 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 Intel Quartus Prime data files, such as the Intel Quartus Prime Project File (`.qpf`), Intel Quartus Prime Settings File (`.qsf`), Quartus IP File (`.ip`), or Platform Designer (Standard) 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 Intel 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.

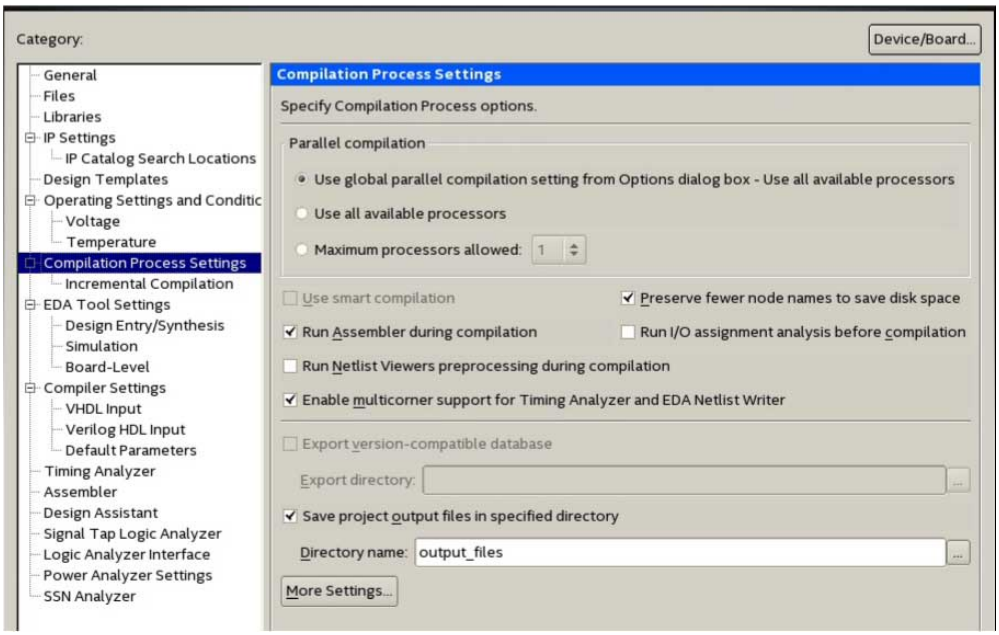
2.3. Managing Project Settings

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.

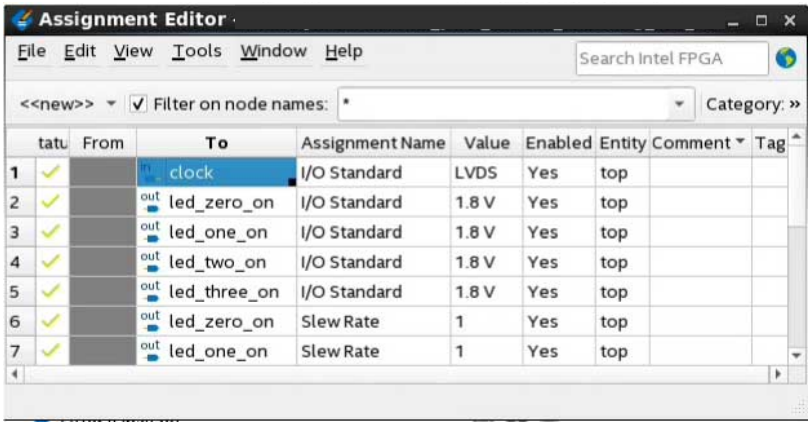
Figure 8. Settings Dialog Box for Global Project Settings



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 9. Assignment Editor

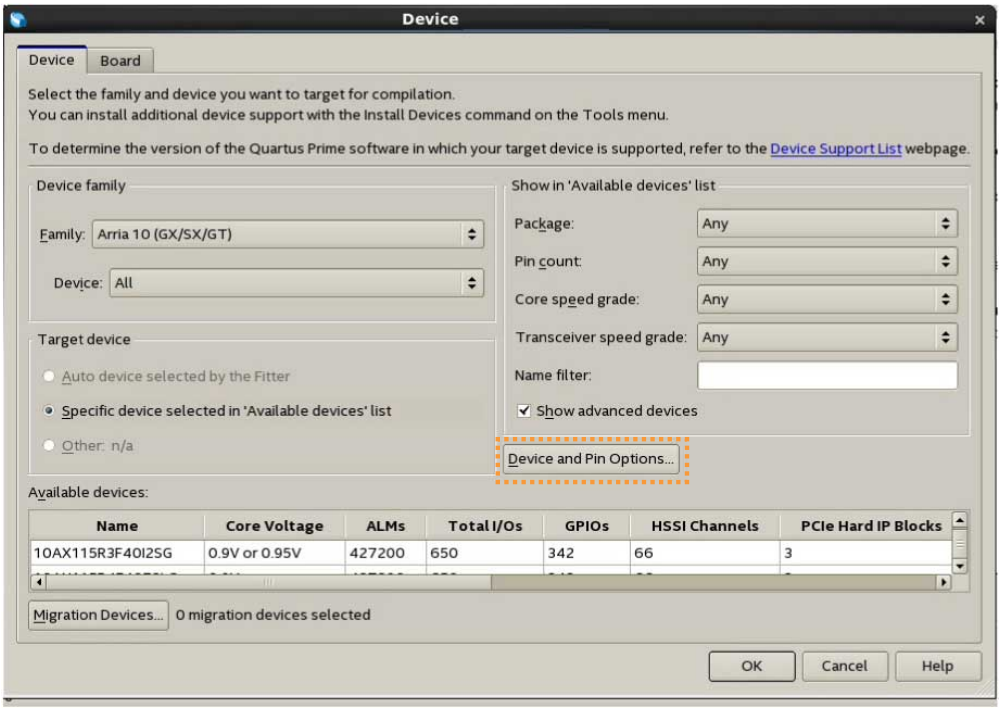


2.3.1. Specifying the Target Device or Board

Specify the target Intel device or board for your project in the **Device** dialog box. Click the **Device and Pin Options** button in the dialog to specify preferences for the device configuration scheme, programming file generation, I/O timing, voltage, and other options.

1. Open a project in the Intel Quartus Prime software.
2. Click **Assignments ➤ Device**.

Figure 10. Device Dialog Box



3. Specify either a target Intel FPGA board or device for your project. When you specify a board, the Intel Quartus Prime software generates the appropriate pin assignments script for that board automatically.
   - To specify an Intel FPGA board or development kit for your project:
     a. Click the **Board** tab.
     b. Select the target device **Family** and a supported **Development Kit**. Click **Yes** if prompted to remove existing **Location** and **I/O Standard** pin assignments. The Intel Quartus Prime software creates the kit's baseline design and stores the design in *<current_project_dir>*/devkits/*<design_name>*. To retain all your existing pin assignments, click **No**.
     c. Select the desired development kit and click **OK**.
   - To specify a device family for your project:
     a. On the **Device** tab, select the **Family** and **Device** name. The list of **Available devices** reflects your selections.
     b. To further refine your selection, specify options for the **Package**, **Pin count**, **Core speed grade**, **Name filter**, and **Show advanced devices** filters.
     c. From the **Available devices**, select your target device **Name** and click **OK**.

2.3.2. Optimizing Project Settings

Optimize project settings to meet your design goals. The Intel 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.

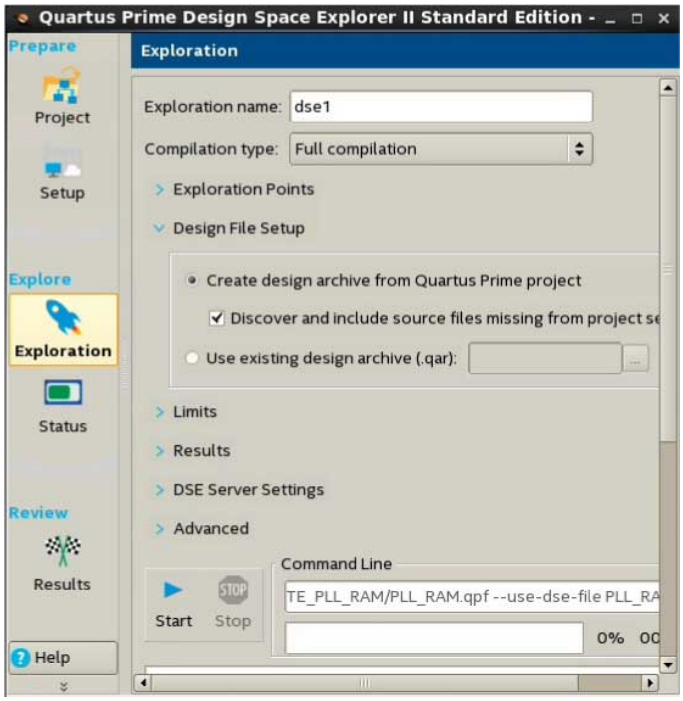
The Intel Quartus Prime software includes several advisors to help you optimize your design and reduce compilation time. The advisors listed in the **Tools ➤ Advisors** menu can provide recommendations based on your project settings and design constraints.

2.3.2.1. Optimize Settings with Design Space Explorer II

Use Design Space Explorer II (**Tools > Launch Design Space Explorer II**) to find optimal project settings for resource, performance, or power optimization goals. Design Space Explorer II (DSE II) processes your design using various setting and constraint combinations, and reports the best settings for your design.

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 11. Design Space Explorer II



2.3.2.2. Optimize Settings with Project Revisions

You can save multiple, named project revisions within your Intel 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.

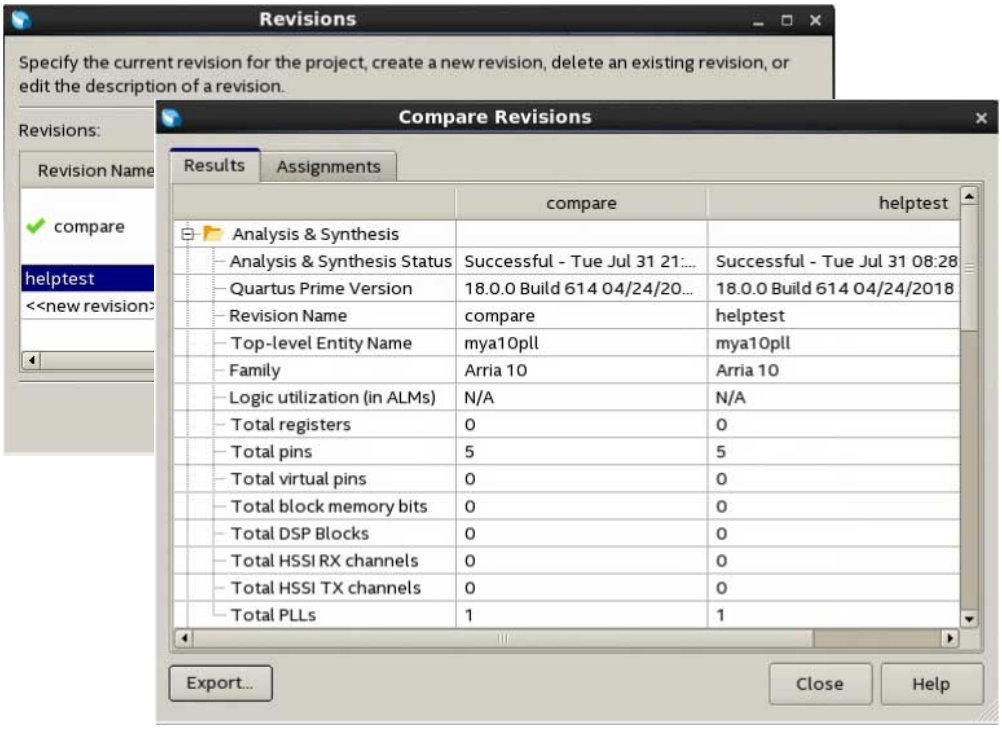
Use revisions to experiment with different settings while preserving the original. Optimize different revisions for separate applications:

- 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 Intel 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 Intel 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 12. Comparing Project Revisions



Related Information

Project Revision Commands on page 30

2.3.2.3. Back-Annotating Compiler 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 device and resource assignments to the project `.qsf` if you want to preserve that same implementation in subsequent compilations.

Click **Assignments ➤ Back-Annotate Assignments** to copy the device resource assignments from the last compilation to the `.qsf` for use in the next compilation. Select the back-annotation type in the **Back-annotation type** list.

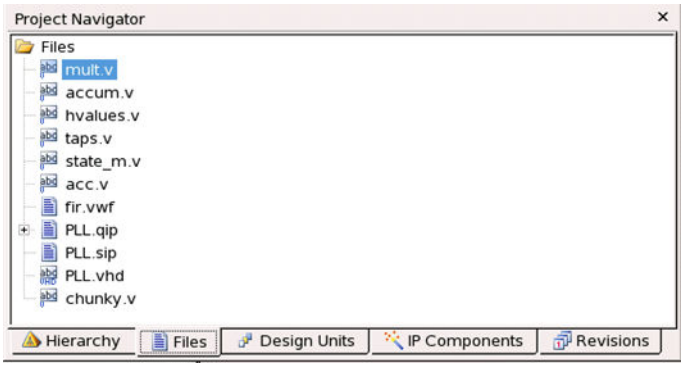
2.4. Managing Logic Design Files

The Intel 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 Intel Quartus Prime software includes full-featured schematic and text editors, as well as HDL templates to accelerate your design work. The Intel Quartus Prime software supports VHDL Design Files (`.vhd`), Verilog HDL Design Files (`.v`), SystemVerilog (`.sv`) and schematic Block Design Files (`.bdf`). The Intel Quartus Prime software also supports Verilog Quartus Mapping (`.vqm`) design files generated by other design entry and synthesis tools. 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 (Standard) system (`.qsys`).

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 13. 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**

2.4.1. Including Design Libraries

Include design files libraries in your project. Specify libraries for a single project, or for all Intel 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 26

2.4.2. 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.

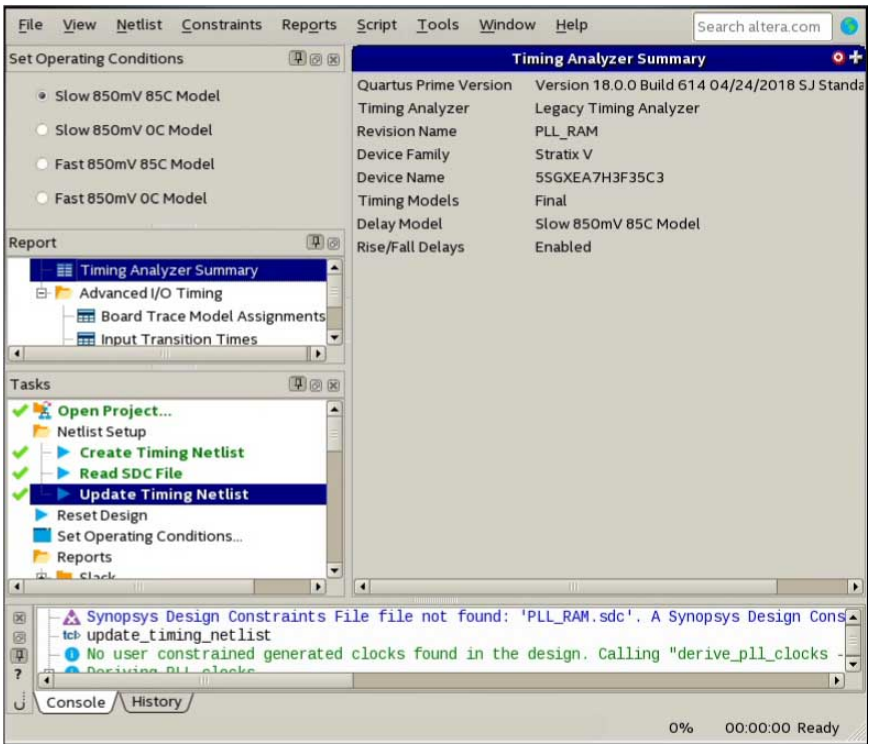
2.5. Managing Timing Constraints

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.

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.

Figure 14. Timing Analyzer



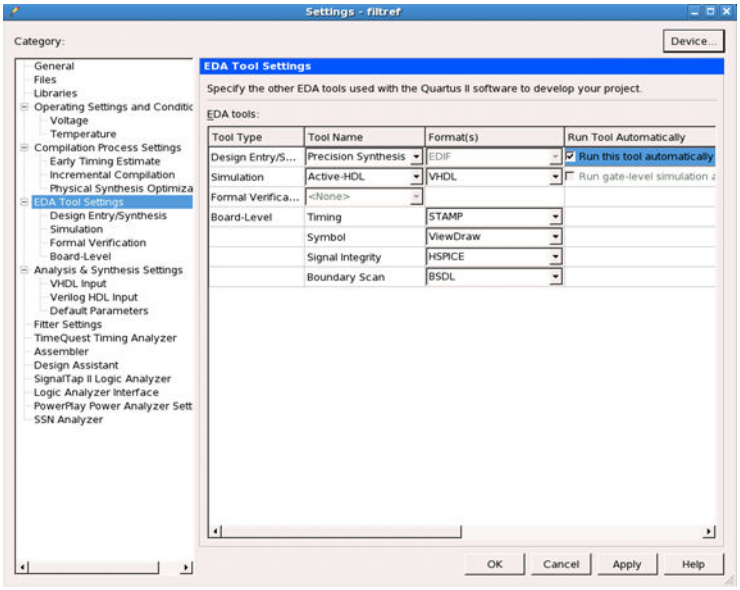
2.6. Integrating Other EDA Tools

Optionally integrate supported EDA design entry, synthesis, simulation, physical synthesis, and formal verification tools into the Intel Quartus Prime design flow. The Intel Quartus Prime software supports netlist files from other EDA design entry and synthesis tools. The Intel Quartus Prime software optionally generates various files for use in other EDA tools.

The Intel Quartus Prime software manages EDA tool files and provides the following integration capabilities:

- Automatically generate files for synthesis and simulation and automatically launch other EDA tools (**Assignments > Settings > EDA Tool Settings > NativeLink Settings**). The Intel Quartus Prime Pro Edition software does not support NativeLink.
- Compile all RTL and gate-level simulation model libraries for your device, simulator, and design language automatically (**Tools > Launch Simulation Library Compiler**).
- Include files generated by other EDA design entry or synthesis tools in your project as synthesized design files (**Project > Add/Remove File from Project**) .
- Automatically generate optional files for board-level verification (**Assignments > Settings > EDA Tool Settings**).

Figure 15. EDA Tool Settings



2.7. Exporting Compilation Results

When you run compilation, the Compiler preserves a database of results in a Quartus Database File (`.qdb`). The `.qdb` contains the data to reproduce similar 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 Intel 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.

Table 4. Exporting Compilation Results

To Export Compilation Results For	Method	Description
Complete Design	Click **Project ➤ Export Database**	Saves compilation results for the entire project in a version-compatible format. You can migrate the results to a later version of the Intel Quartus Prime software.
Design Partition	Click **Project ➤ Export Design Partition**	Saves compilation results for a design partition as a Quartus Prime Exported Partition File (`.qxp`) that you can import to another project. You can export the results for the post-fit or post-synthesis netlist.

Related Information

Project Database Commands on page 31

2.7.1. Exporting a Version-Compatible Compilation Database

To export a project compilation database to a format that ensures version-compatibility with a later version of the Intel Quartus Prime software:

1. In the Intel 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 Database**, specify the **Export directory** name, and click **OK**. The database files export to the location you specify. You can now import this exported database into a later version of the Intel Quartus Prime software.

Note: You can turn on **Assignments > Settings > Compilation Process Settings > Export version-compatible database** if you want to always export a version-compatible database following compilation.

2.7.2. Importing a Version-Compatible Compilation Database

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

1. Export a version-compatible compilation database for a complete design, as Exporting a Version-Compatible Compilation Database on page 23 describes.
2. In a newer version of the Intel Quartus Prime software, open the original project. Click **Yes** if prompted to open a project created with a different software version.
3. If you have previously compiled the design you want to export, click **Project ➤ Clean Project** to clean the old compilation database before import.
4. Click **Project > Import Database** and select the exported database directory (by default, `<project directory>/export_db/`). **Timing analysis mode** is available to disable legality checks for certain configuration rule changes from prior versions of the Intel Quartus Prime software. Enable this option only if your design does not successfully import with the option disabled. After you import a design with **Timing analysis mode**, you cannot the project to generate programming files.
5. Click **OK**.

2.7.3. Exporting a Design Partition

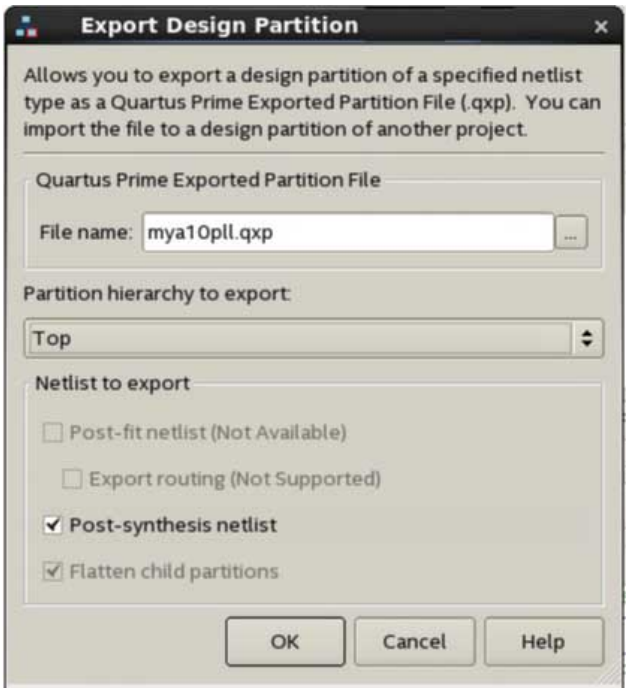
The following steps describe export of design partitions that you create in your project.

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.
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 16. Export Design Partition Dialog Box



   - Under **Quartus Prime Exported Partition File**, specify a file name.
   - Select the **Partition hierarchy** to export.
   - Under **Netlist to export**, select the **Post-fit netlist** or **Post-synthesis netlist** for export.
4. Click **OK**. The compilation results for the design partition exports to the file that you specify.

2.7.4. 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 17. Clean Project Dialog Box Cleans the Project Database

2.8. Migrating Projects Across Operating Systems

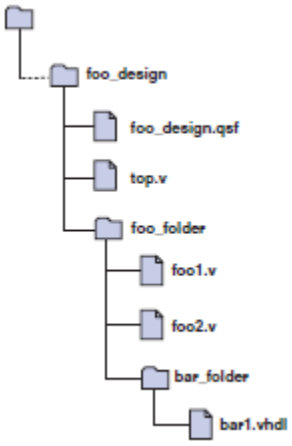
Consider the following cross-platform issues when moving your project from one operating system to another (for example, from Windows* to Linux*).

2.8.1. 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 Intel 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 18. All Inclusive Project Directory Structure

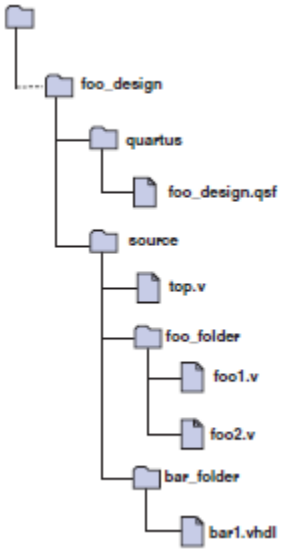


2.8.1.1. Use Relative Paths

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 19. Intel Quartus Prime Project Directory Separate from Design Files



2.8.2. 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 Intel 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 Intel 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 Intel 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:\`

2.9. Archiving Projects

You can optionally save the elements of a project in a single, compressed Intel Quartus Prime Archive File (`.qar`) by clicking **Project > Archive Project**. The `.qar` preserves logic 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 Intel Quartus Prime software, or to Intel support. Optionally add compilation results, Platform Designer system files, and third-party EDA tool files to the archive.

If you restore the archive in a different version of the Intel Quartus Prime software, you must include the original `.qdf` in the archive to preserve original compilation results.

Related Information

Project Archive Commands on page 31

2.9.1. Manually Adding Files To Archives

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

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**.

2.9.2. Archiving Compilation Results

Optionally include compilation results in a project archive to avoid recompilation and preserve original results in the restored project. To archive compilation results, export the post-synthesis or post-fit version compatible database and include this file in the archive.

1. Export the project database.
2. Click **Project > Archive Project** and specify the archive file name.
3. Click **Advanced**.
4. Under **File subsets**, turn on **Version-compatible database files** and click **OK**.
5. Click **Archive**.

To restore an archive containing a version-compatible database, follow these steps:

1. Click **Project > Restore Archived Project**.
2. Select the archive name and destination folder and click **OK**.
3. After restoring the archived project, click **Project > Import Database** and import the version-compatible database.

Related Information

Exporting a Version-Compatible Compilation Database on page 23

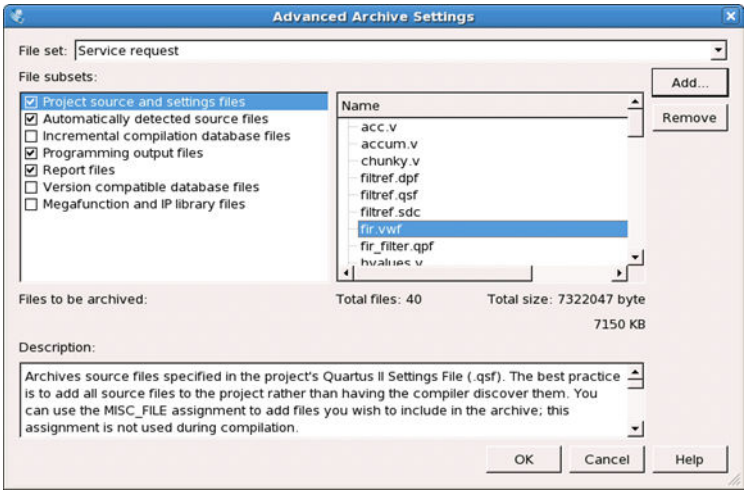
2.9.3. Archiving Projects for Service Requests

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`)
   - Platform Designer system and IP files (`.qsys`, `.qip`)
4. Click **OK**, and then click **Archive**.

Figure 20. Archiving Project for Service Request



2.9.4. Using 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 Intel 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.

2.9.4.1. Files to Include In External Revision Control

Include the following project file types in external revision control systems:

- Logic design files (`.v`, `.vdh`, `.bdf`, `.edf`, `.vqm`)
- Timing constraint files (.sdc)
- Quartus project settings and constraints (`.qdf`, `.qpf`, `.qsf`)
- IP files (`.ip`, `.v`, `.sv`, `.vhd`, `.qip`, `.sip`, `.qsys`)
- Platform Designer (Standard)-generated files (`.qsys`, `.ip`, `.sip`)
- EDA tool files (`.vo`, `.vho`)

Generate or modify these files manually if you use a scripted design flow. If you use an external source code control system, check-in project files anytime you modify assignments and settings.

2.10. 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>`.

2.10.1. Project Revision Commands

create_revision Command

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

```
create_revision <name> -based_on <revision_name> -copy_results -set_current
```

Table 5. 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.
`copy_results`	Copies the results from the `based_on` 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

Optimize Settings with Project Revisions on page 17

2.10.2. Project Archive Commands

project_archive Command

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

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

Table 6. project_archive Command Options

Options	Description
`-all_revisions`	Includes all revisions of the current project in the archive.
`-auto_common_directory`	Preserves original project directory structure in 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.

Note: Version-compatible databases are not available for some device families. If you require the database files to reproduce the compilation results in the same Intel Quartus Prime software version, use the `-use_file_set full_db` option to archive the complete database.

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 27

2.10.3. Project Database Commands

Related Information

Exporting Compilation Results on page 22

2.10.3.1. Import and Export Version-Compatible Databases from a Flow Package

The following are Tcl commands from the `flow` package to import or export version-compatible databases. If you use the `flow` package, you must specify the database directory variable name. `flow` and `database_manager` packages contain commands to manage version-compatible databases.

- `set_global_assignment -name VER_COMPATIBLE_DB_DIR <directory>`
- *`execute_flow –flow export_database`*
- execute_flow –flow import_database

2.10.3.2. quartus_cdb and quartus_sh Executables to Manage Version-Compatible Databases

Use the following commands to manage version-compatible databases:

- `quartus_cdb <project> -c <revision> --export_database=<directory>`
- `quartus_cdb <project> -c <revision> --import_database=<directory>`
- `quartus_sh –flow export_database <project> -c \ <revision>`
- `quartus_sh –flow import_database <project> -c \ <revision>`

2.10.4. Project Library Commands

Use the following commands to script project library changes.

2.10.4.1. Specify Project Libraries With SEARCH_PATH Assignment

In Tcl, use commands in the `:: quartus ::project` package to specify project libraries, and the `set_global_assignment` command.

Use the following commands to script project library changes:

- `set_global_assignment -name SEARCH_PATH "../other_dir/library1"`
- `set_global_assignment -name SEARCH_PATH "../other_dir/library2"`
- `set_global_assignment -name SEARCH_PATH "../other_dir/library3"`

2.10.4.2. Report Specified Project Libraries Commands

To report any project libraries specified for a project and any global libraries specified for the current installation of the Quartus software, use the `get_global_assignment` and `get_user_option` Tcl commands.

Use the following commands to report specified project libraries:

- `get_global_assignment -name SEARCH_PATH`
- `get_user_option -name SEARCH_PATH`

2.10.4.3. Generate Version-Compatible Database After Compilation

Use the following commands to generate a version-compatible database after compilation:

- `set_global_assignment -name AUTO_EXPORT_VER_COMPATIBLE_DB ON`
- `set_global_assignment-name VER_COMPATIBLE_DB_DIR <directory>`

2.11. Managing Projects Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Subdivided "Exporting, Archiving, and Migrating Projects" into separate sections. • 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 Standard Edition User Guide*.
2018.02.11	18.0.0	• Added description of | as root partition hierarchy path in Design Partitions Window. • Removed "Scripting IP Simulation" and "Generating a Combined Simulation Script" topics. These features are supported only for Intel Arria 10 devices in Intel Quartus Prime Standard Edition. • Added link to "Scripting IP Simulation" in the *Introduction to Intel FPGA IP Cores*.

Date	Version	Changes
2017.11.06	17.1.0	• Revised product branding for Intel standards. • Changed instances of *Qsys* to *Platform Designer (Standard)* • Revised topics on Intel FPGA IP Evaluation Mode (formerly OpenCore). • Removed `-compatible` attribute from `export_design` command content. • Updated IP Core Upgrade Status table with new icons, and added row for IP Component Outdated status.
2017.05.08	17.0.0	• Added topic on Back-Annotate Assignments command.
2016.10.31	16.1.0	• Updated screenshots.
2016.05.03	16.0.0	Removed statements about serial equivalence when using multiple processors.
2016.02.09	15.1.1	• Clarified instructions for Generating a Combined Simulator Setup Script. • Clarified location of **Save project output files in specified directory** option.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2015.05.04	15.0.0	• Added description of design templates feature. • Updated screenshot for DSE II GUI. • Added qsys_script IP core instantiation information. • Described changes to generating and processing of instance and entity names.

continued...

Date	Version	Changes
		• Added description of upgrading IP cores at the command line. • Updated procedures for upgrading and migrating IP cores. • Gate level timing simulation supported only for Cyclone IV and Stratix IV devices.
2014.12.15	14.1.0	• Updated content for DSE II GUI and optimizations. • Added information about new **Assignments ➤ Settings ➤ IP Settings** that control frequency of synthesis file regeneration and automatic addition of IP files to the project.
2014.08.18	14.0a10.0	• Added information about specifying parameters for IP cores targeting Arria 10 devices. • Added information about the latest IP output for version 14.0a10 targeting Arria 10 devices. • Added information about individual migration of IP cores to the latest devices. • Added information about editing existing IP variations.
2014.06.30	14.0.0	• 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.
November 2013	13.1.0	• Conversion to DITA format
May 2013	13.0.0	• Overhaul for improved usability and updated information.
June 2012	12.0.0	• Removed survey link. • Updated information about `VERILOG_INCLUDE_FILE`.
November 2011	10.1.1	Template update.
December 2010	10.1.0	• Changed to new document template. • Removed Figure 4–1, Figure 4–6, Table 4–2. • Moved "Hiding Messages" to Help. • Removed references about the `set_user_option` command. • Removed Classic Timing Analyzer references.

3. Design Planning

3.1. 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 Intel 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.

3.2. Create a Design Specification and Test Plan

Before you create your design logic or complete your system design, it is best practice to create detailed design specifications that define the system, specify the I/O interfaces for the FPGA, identify the different clock domains, and include a block diagram of basic design functions.

In addition, creating a test plan helps you to design for verification and ease of manufacture. For example, your test plan can include validation of interfaces incorporated in your design. To perform any built-in self-test functions to drive interfaces, you can use a UART interface with a Nios® II processor inside the FPGA device.

If more than one designer contributes to the design, consider a common design directory structure or source control system to make design integration easier. Consider whether you want to standardize on an interface protocol for each design block.

3.3. Plan for the Target Device

Intel offers a broad portfolio of FPGA and PLD devices. The Intel device that you select determines factors of performance, density, and board layout. To avoid costly design changes, it is best to carefully consider and determine the target device family early in

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

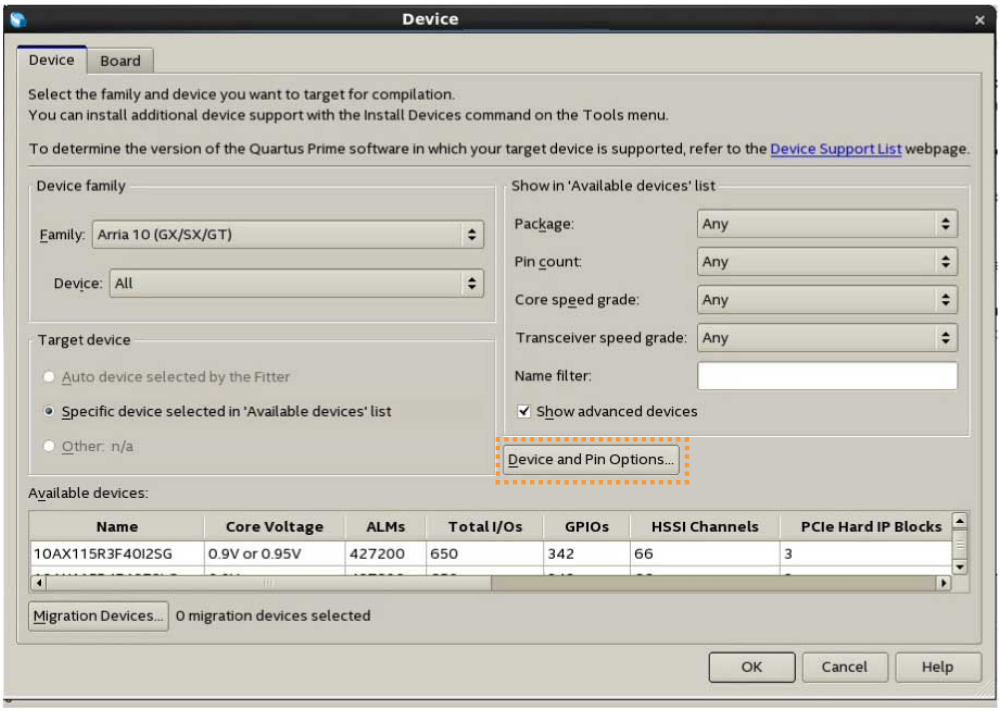
ISO 9001:2015 Registered

the design cycle. Intel FPGA device families differ in cost, size, density, performance, power consumption, packaging, I/O standards, and other factors. Select the device family that best suits your most critical design requirements.

Device Family Selection Guidelines

- Refer to the Product Selector tool on the Intel website to quickly find and compare the specifications and features of Intel FPGA devices and development kits.
- Once you identify the target device family, refer to the device family technical documentation for detailed device characteristics. Each device family includes complete documentation, including a datasheet and user guide or handbook. You can also view a summary of each device's resources by selecting a device in the **Device** dialog box (**Assignments ➤ Device**)

Figure 21. Device Dialog Box



- Consider whether the device family meets any requirements you have 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.

Related Information

Product Selector Guide Tool
To help you choose your device.

3.3.1. Device Migration Planning

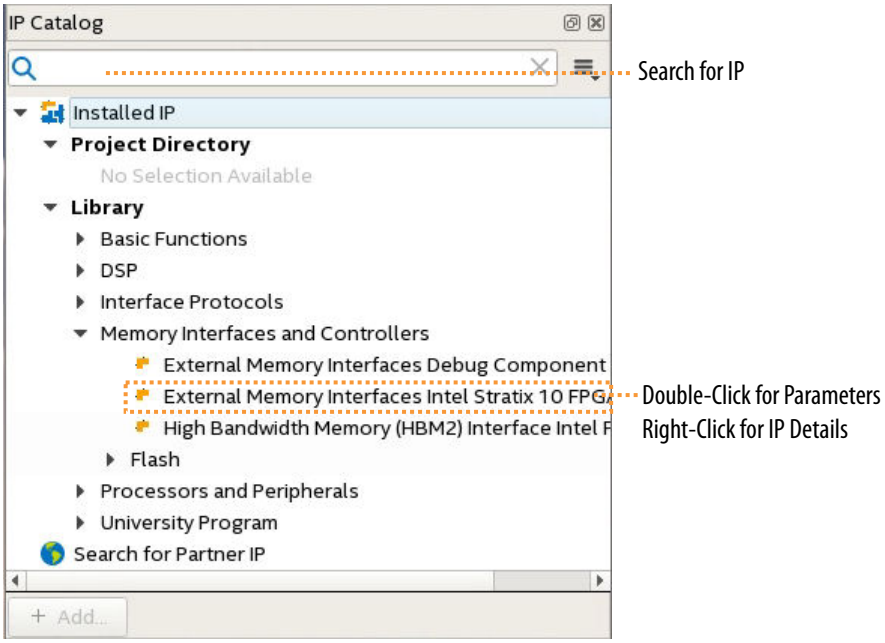
Determine whether you want to migrate your design to another device density to allow flexibility when your design nears completion. You may want to target a smaller (and less expensive) device and then move to a larger device if necessary to meet your design requirements. Other designers may prototype their design in a larger device to reduce optimization time and achieve timing closure more quickly, and then migrate to a smaller device after prototyping. If you want the flexibility to migrate your design, you must specify these migration options in the Intel Quartus Prime software at the beginning of your design cycle.

Selecting a migration device impacts pin placement because some pins may serve different functions in different device densities or package sizes. If you make pin assignments in the Intel Quartus Prime software, the Pin Migration View in the Pin Planner highlights pins that change function between your migration devices.

3.4. Plan for Intellectual Property Cores

Intel and third-party intellectual property (IP) partners offer a large selection of standardized IP cores optimized for Intel FPGA devices. The IP you select often affects system design and performance, especially if the FPGA interfaces with other devices in the system. Plan which I/O interfaces or other blocks in the system that you want to implement using IP cores. Whenever possible, plan to incorporate these functions into your design using Intel FPGA IP cores, many of which are available for production use in the Intel Quartus Prime software without additional license.

Figure 22. IP Catalog



For IP cores that require additional license for production use, the Intel FPGA IP Evaluation Mode, allows you to program the FPGA to verify the IP in the hardware before you purchase the IP license. Refer to Introduction to Intel FPGA IP Cores on page 51 for general information on using Intel FPGA IP cores.

Related Information

- Introduction to Intel FPGA IP Cores on page 51
- Intel FPGA IP Portfolio Web Page
  For descriptions and documentation for all available Intel FPGA and partner IP cores.

3.5. Plan for Standard Interfaces

To reduce design iterations and costly design changes, plan for use of standard interfaces in system design. Using standard interfaces ensures compatibility between design blocks from different design teams or vendors. Standard interfaces simplify the interface logic to each design block, and enable individual team members to test their individual design blocks against the specification for the interface protocol to ease system integration.

You can use the Intel Quartus Prime Platform Designer system integration tool to use standard interfaces and speed-up system-level integration. Platform Designer components use Avalon® standard interfaces for physical connections, allowing you to connect any logical device (either on-chip or off-chip) that has an Avalon interface. Platform Designer allows you to define system components in a GUI, and then automatically generates the required interconnect logic, along with clock-crossing and width adapters.

The Avalon standard includes two interface types:

- Avalon Memory-Mapped (Avalon-MM)—allow a component to use an address-mapped read or write protocol that connects master components to slave components.
- Avalon Streaming (Avalon-ST)—enables point-to-point connections between streaming components that send and receive data using a high-speed, unidirectional system interconnect between source and sink ports.

Related Information

Creating a System with Platform Designer

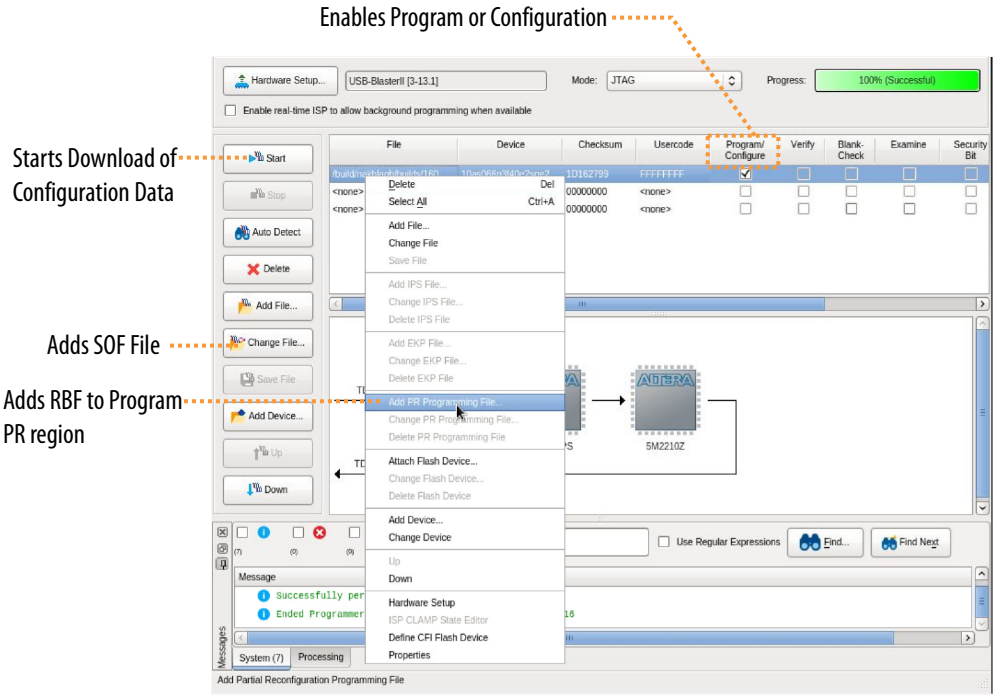
3.6. Plan for Device Programming

You must plan for the devices and hardware that you require for programming or configuration of the device. Comprehensive system planning includes determining what companion devices, if any, your system requires. Your programming or configuration method also impacts the board layout planning. For example, some programming options require a JTAG interface connection, requiring a JTAG chain on the board.

You can define a configuration scheme on the **Configuration** tab of the **Device and Pin Options** dialog box. The Intel Quartus Prime software uses the settings for the configuration scheme, configuration device, and configuration device voltage to enable

the appropriate dual purpose pins as regular I/O pins after you complete configuration. The Intel Quartus Prime software performs voltage compatibility checks of those pins during compilation of your design.

Figure 23. Intel Quartus Prime Programmer



The technical documentation for each device family describes the available configuration options.

3.7. Plan for Device Power Consumption

You can use the Intel Quartus Prime power estimation and analysis tools to estimate power consumption and guide PCB board and system design. You must accurately estimate device power consumption to develop an appropriate power budget and to design the power supplies, voltage regulators, heat sink, and cooling system. You can use the Early Power Estimator (EPE) spreadsheet to estimate power consumption before running a compilation or creating any source code. Then, you can use the Intel Quartus Prime Power Analyzer to perform a more accurate analysis after your design is complete.

Note: Because power consumption is heavily dependent on actual design and environmental conditions, make sure to verify the actual power consumption during device operation.

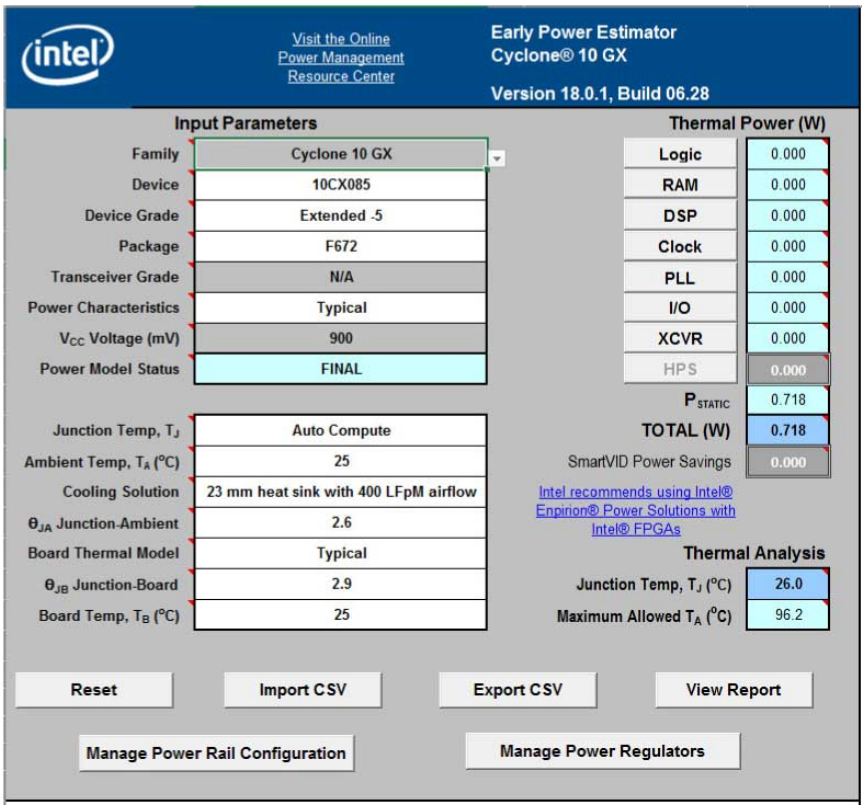
Power estimation and analysis helps you ensure that your design satisfies thermal and power supply requirements:

- Thermal—ensure that the cooling solution is sufficient to dissipate the heat generated by the device. The computed junction temperature must fall within normal device specifications.
- Power supply—ensure that the power supplies provide adequate current to support device operation.

Early Power Estimator (EPE) Spreadsheet

The Early Power Estimator (EPE) spreadsheet allows you to estimate power utilization for your design. Estimating power consumption early in the design cycle allows planning of power budgets and avoids unexpected results when designing the PCB.

Figure 24. Early Power Estimator (EPE) Spreadsheet



You can manually enter data into the EPE spreadsheet, or use the Intel Quartus Prime software to generate device resource information for your design.

To manually enter data into the EPE spreadsheet, enter the device resources, operating frequency, toggle rates, and other parameters for your design. If you do not have an existing design, estimate the number of device resources used in your design, and then enter the data into the EPE spreadsheet manually.

If you have an existing design or a partially completed design, you can use the Intel Quartus Prime software to generate the Early Power Estimator File (`.txt`, `.csv`) to assist you in completing the EPE spreadsheet.

The EPE spreadsheet includes the Import Data macro that parses the information in the EPE File and transfers the information into the spreadsheet. If you do not want to use the macro, you can manually transfer the data into the EPE spreadsheet. For example, after importing the EPE File information into the EPE spreadsheet, you can add device resource information. If the existing Intel Quartus Prime project represents only a portion of your full design, manually enter the additional device resources you use in the final design.

Intel Quartus Prime Power Analyzer

After you complete your design, you can use the Intel Quartus Prime Power Analyzer to perform a complete post-fit power analysis to check the power consumption more accurately. The Power Analyzer provides an accurate estimation of power, ensuring that thermal and supply limitations are met.

Related Information

Early Power Estimator and Power Analyzer Web Page

3.8. Plan for Interface I/O Pins

In many design environments, FPGA designers want to plan the top-level FPGA I/O pins early to help board designers begin the PCB design and layout. The I/O capabilities and board layout guidelines of the FPGA device influence pin locations and other types of assignments. If the board design team specifies an FPGA pin-out, the pin locations must be verified in the FPGA placement and routing software to avoid board design changes.

You can create a preliminary pin-out for an Intel FPGA with the Intel 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.

The Intel Quartus Prime I/O Assignment Analysis checks that the pin locations and assignments are supported in the target FPGA architecture. You can then use I/O Assignment Analysis to validate I/O-related assignments that you create or modify throughout the design process. When you compile your design in the Intel Quartus Prime software, I/O Assignment Analysis runs automatically in the Fitter to validate that the assignments meet all the device requirements and generates error messages.

Early in the design process, before creating the source code, the system architect has information about the standard I/O interfaces (such as memory and bus interfaces), the IP cores in your design, and any other I/O-related assignments defined by system requirements. You can use this information with the **Early Pin Planning** feature in the Pin Planner to specify details about the design I/O interfaces. You can then create a top-level design file that includes all I/O information.

The Pin Planner interfaces with the IP core parameter editor, which allows you to create or import custom IP cores that use I/O interfaces. You can configure how to connect the functions and cores to each other by specifying matching node names for selected ports. You can create other I/O-related assignments for these interfaces or other design I/O pins in the Pin Planner, as described in this section. The Pin Planner creates virtual pin assignments for internal nodes, so internal nodes are not assigned to device pins during compilation.

After analysis and synthesis of the newly generated top-level wrapper file, use the generated netlist to perform I/O Analysis with the **Start I/O Assignment Analysis** command.

You can use the I/O analysis results to change pin assignments or IP parameters even before you create your design, and repeat the checking process until the I/O interface meets your design requirements and passes the pin checks in the Intel Quartus Prime software. When you complete initial pin planning, you can create a revision based on the Intel Quartus Prime-generated netlist. You can then use the generated netlist to develop the top-level design file for your design, or disregard the generated netlist and use the generated Intel Quartus Prime Settings File (`.qsf`) with your design.

During this early pin planning, after you have generated a top-level design file, or when you have developed your design source code, you can assign pin locations and assignments with the Pin Planner.

With the Pin Planner, you can identify I/O banks, voltage reference (VREF) groups, and differential pin pairings to help you through the I/O planning process. If you selected a migration device, the **Pin Migration View** highlights the pins that have changed functions in the migration device when compared to the currently selected device. Selecting the pins in the Device Migration view cross-probes to the rest of the Pin Planner, so that you can use device migration information when planning your pin assignments. You can also configure board trace models of selected pins for use in "board-aware" signal integrity reports generated with the **Enable Advanced I/O Timing** option . This option ensures that you get accurate I/O timing analysis. You can use a Microsoft Excel spreadsheet to start the I/O planning process if you normally use a spreadsheet in your design flow, and you can export a Comma-Separated Value File (`.csv`) containing your I/O assignments for spreadsheet use when you assign all pins.

When you complete your pin planning, you can pass pin location information to PCB designers. The Pin Planner is tightly integrated with certain PCB design EDA tools, and can read pin location changes from these tools to check suggested changes. Your pin assignments must match between the Intel Quartus Prime software and your schematic and board layout tools to ensure the FPGA works correctly on the board, especially if you must make changes to the pin-out. The system architect uses the Intel Quartus Prime software to pass pin information to team members designing individual logic blocks, allowing them to achieve better timing closure when they compile their design.

Start FPGA planning before you complete the HDL for your design to improve the confidence in early board layouts, reduce the chance of error, and improve the overall time to market of the design. When you complete your design, use the Fitter reports for the final sign-off of pin assignments. After compilation, the Intel Quartus Prime software generates the Pin-Out File (`.pin`), and you can use this file to verify that each pin is correctly connected in board schematics.

Related Information

Intel Quartus Prime Standard Edition User Guide: Design Optimization
For more information about I/O assignment and analysis.

3.8.1. Simultaneous Switching Noise Analysis

Simultaneous switching noise (SSN) is a noise voltage inducted onto a victim I/O pin of a device due to the switching behavior of other aggressor I/O pins in the device.

Intel provides tools for SSN analysis and estimation, including SSN characterization reports, an Early SSN Estimator (ESE) spreadsheet tool, and the SSN Analyzer in the Intel Quartus Prime software. SSN often leads to the degradation of signal integrity by causing signal distortion, thereby reducing the noise margin of a system. You must

address SSN with estimation early in your system design, to minimize later board design changes. When your design is complete, verify your board design by performing a complete SSN analysis of your FPGA in the Intel Quartus Prime software.

3.9. Plan for other EDA Tools

Your complete FPGA design flow may include third-party EDA tools in addition to the Intel Quartus Prime software. Determine which tools you want to use with the Intel Quartus Prime software to ensure that they are supported and set up properly, and that you are aware of their capabilities.

3.9.1. Third-Party Synthesis Tools

You can use supported standard third-party EDA synthesis tools to synthesize your Verilog HDL or VHDL design, and then compile the resulting output netlist file in the Intel Quartus Prime software. The Intel Quartus Prime Standard Edition software includes integrated synthesis that supports Verilog HDL, VHDL, Altera Hardware Description Language (AHDL), and schematic design entry.

Different synthesis tools may give different results for each design. To determine the best tool for your application, you can experiment by synthesizing typical designs for your application and coding style. Perform placement and routing in the Intel Quartus Prime software to get accurate timing analysis and logic utilization results.

The synthesis tool you choose may allow you to create an Intel Quartus Prime project and pass constraints, such as the EDA tool setting, device selection, and timing requirements that you specified in your synthesis project. You can save time when setting up your Intel Quartus Prime project for placement and routing.

Tool vendors frequently add new features, fix tool issues, and enhance performance for Intel devices, you must use the most recent version of third-party synthesis tools.

3.9.2. Third-Party Simulation Tools

Intel provides the Mentor Graphics ModelSim* - Intel FPGA Edition simulator with the Intel Quartus Prime software. You can also purchase the ModelSim - Intel FPGA Edition or a full license of the ModelSim software to support large designs and achieve faster simulation performance. The Intel Quartus Prime software generates both functional and timing netlist files for ModelSim and other supported third-party simulators.

Use the simulator version that your Intel Quartus Prime software version supports for best results. You must also use the model libraries provided with your Intel Quartus Prime software version. Libraries can change between versions, which might cause a mismatch with your simulation netlist.

3.10. Plan 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.

The Intel Quartus Prime in-system debugging tools offer different advantages and trade-offs, depending on the characteristics of your design. 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) or Signal Probe tools, which require 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.

  Note: You can add the Signal Tap Logic Analyzer as a separate design partition for incremental compilation to minimize recompilation times.

Table 7. 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

Related Information

Intel Quartus Prime Standard Edition User Guide: Debug Tools

3.11. Plan HDL Coding Styles

When you develop complex FPGA designs, design practices and coding styles have an enormous impact on the timing performance, logic utilization, and system reliability of your device.

3.11.1. Design Recommendations

Use synchronous design practices to consistently meet your design goals. Problems with asynchronous design techniques include reliance on propagation delays in a device, incomplete timing analysis, and possible glitches.

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.

Clock signals have a large effect on the timing accuracy, performance, and reliability of your design. Problems with clock signals can cause functional and timing problems in your design. 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.

The Design Assistant in the Intel Quartus Prime software is a design-rule checking tool that enables you to verify design issues. The Design Assistant checks your design for adherence to Intel-recommended design guidelines. You can also use third-party lint tools to check your coding style. The Design Assistant does not support Max 10 and Intel Arria 10 devices.

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.

3.11.2. Recommended HDL Coding Styles

HDL coding styles can have a significant effect on the quality of results for programmable logic designs.

If you design memory and DSP functions, you must understand the target architecture of your device so you can use the dedicated logic block sizes and configurations. Follow the coding guidelines for inferring Intel FPGA IP and targeting dedicated device hardware, such as memory and DSP blocks.

Related Information

Intel Quartus Prime Standard Edition User Guide: Design Recommendations

3.11.3. Managing Metastability

Metastability problems can occur in digital design when a signal is transferred between circuitry in unrelated or asynchronous clock domains, because the designer cannot guarantee that the signal meets the setup and hold time requirements during the signal transfer.

Designers commonly use a synchronization chain to minimize the occurrence of metastable events. 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.

You can use the Intel 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. 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. Determine an acceptable target MTBF given the context of your entire system and the fact that MTBF calculations are statistical estimates.

The Intel Quartus Prime software can help you determine whether you have enough synchronization registers in your design to produce a high enough MTBF at your clock and data frequencies.

Related Information

Managing Metastability, Intel Quartus Prime Standard Edition User Guide: Design Recommendations

3.12. Plan for Hierarchical and Team-Based Designs

The Intel Quartus Prime Compiler supports hierarchical design methodologies to reduce design compilation times and preserve performance. In a flat compilation flow, the design hierarchy is flattened without design partitions. In block-based (hierarchical) flows, you can subdivide your design by creating design partitions.

Hierarchical flows allow you to isolate, optimize, and preserve compilation results for specific design blocks, but require more design planning to ensure effective results.

3.12.1. Flat Compilation without Design Partitions

In a flat compilation flow without any design partitions, the Intel 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 re-compile 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 Intel 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. You can run **Rapid Recompile** to preserve portions of previous placement and routing in subsequent compilations. **Rapid Recompile** can reduce your compilation time in a flat or partitioned design when you make small changes to your design.

3.12.2. Incremental Compilation with Design Partitions

In an incremental compilation flow, the system architect splits a large design into partitions. When hierarchical design partitions are well chosen and placed in the device floorplan, you can speed up your design compilation time while maintaining the quality of results.

Incremental compilation preserves the compilation results and performance of unchanged partitions in the design, greatly reducing design iteration time by focusing new compilations on changed design partitions only. Incremental compilation then merges new compilation results with the previous compilation results from unchanged design partitions. Additionally, you can target optimization techniques to specific design partitions, while leaving other partitions unchanged. You can also use empty partitions to indicate that parts of your design are incomplete or missing, while you compile the rest of your design.

Third-party IP designers can also export logic blocks to be integrated into the top-level design. Team members can work on partitions independently, which can simplify the design process and reduce compilation time. With exported partitions, the system architect must provide guidance to designers or IP providers to ensure that each partition uses the appropriate device resources. Because the designs may be developed independently, each designer has no information about the overall design or how their partition connects with other partitions. This lack of information can lead to problems during system integration. The top-level project information, including pin locations, physical constraints, and timing requirements, must be communicated to the designers of lower-level partitions before they start their design.

The system architect plans design partitions at the top level and allows third-party designs to access the top-level project framework. By designing in a copy of the top-level project (or by checking out the project files in a source control environment), the designers of the lower-level block have full information about the entire project, which helps to ensure optimal results.

When you plan your design code and hierarchy, ensure that each design entity is created in a separate file so that the entities remain independent when you make source code changes in the file. If you use a third-party synthesis tool, create separate Verilog Quartus Mapping or EDIF netlists for each design partition in your synthesis tool. You may have to create separate projects in your synthesis tool, so that the tool synthesizes each partition separately and generates separate output netlist files. The netlists are then considered the source files for incremental compilation.

3.12.3. Planning Design Partitions and Floorplan Location Assignments

Partitioning a design for an FPGA requires planning to ensure optimal results when you integrate the partitions. Following Intel's recommendations for creating design partitions should improve the overall quality of results.

For example, registering partition I/O boundaries keeps critical timing paths inside one partition that can be optimized independently. When you specify the design partitions, you can use the Incremental Compilation Advisor to ensure that partitions meet Intel's recommendations.

If you have timing-critical partitions that are changing through the design flow, or partitions exported from another Intel Quartus Prime project, you can create design floorplan assignments to constrain the placement of the affected partitions. Good

partition and floorplan design helps partitions meet top-level design requirements when integrated with the rest of your design, reducing time you spend integrating and verifying the timing of the top-level design.

Related Information

Analyzing and Optimizing the Design Floorplan

3.13. Design Planning Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• 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 "Early Pin Planning and I/O Analysis" to "Plan for Interface I/O Pins." • 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." • Revised some wording in "Planning for Hierarchical and Team-Based Design" • Retitled *Design Planning with the Intel Quartus Prime Software* to *Design Planning*

Date	Version	Changes
2017.11.06	17.1.0	• Changed instances of OpenCore Plus to Intel FPGA IP Evaluation Mode. • Changed instances of Qsys to Platform Designer (Standard) (Standard)
2016.05.03	16.0.0	Added information about Development Kit selection.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2015.05.04	15.0.0	Remove support for Early Timing Estimate feature.
2014.06.30	14.0.0	Updated document format.
November 2013	13.1.0	Removed HardCopy device information.
November, 2012	12.1.0	Update for changes to early pin planning feature

continued...

Date	Version	Changes
June 2012	12.0.0	Editorial update.
November 2011	11.0.1	Template update.
May 2011	11.0.0	• Added link to System Design with Qsys in "Creating Design Specifications" on page 1–2 • Updated "Simultaneous Switching Noise Analysis" on page 1–8 • Updated "Planning for On-Chip Debugging Tools" on page 1–10 • Removed information from "Planning Design Partitions and Floorplan Location Assignments" on page 1–15
December 2010	10.1.0	• Changed to new document template • Updated "System Design and Standard Interfaces" on page 1–3 to include information about the Qsys system integration tool • Added link to the Product Selector in "Device Selection" on page 1–3 • Converted information into new table (Table 1–1) in "Planning for On-Chip Debugging Options" on page 1–10 • Simplified description of incremental compilation usages in "Incremental Compilation with Design Partitions" on page 1–14 • Added information about the Rapid Recompile option in "Flat Compilation Flow with No Design Partitions" on page 1–14 • Removed details and linked to Intel Quartus Prime Help in "Fast Synthesis and Early Timing Estimation" on page 1–16
July 2010	10.0.0	• Added new section "System Design" on page 1–3 • Removed details about debugging tools from "Planning for On-Chip Debugging Options" on page 1–10 and referred to other handbook chapters for more information • Updated information on recommended design flows in "Incremental Compilation with Design Partitions" on page 1–14 and removed "Single-Project Versus Multiple-Project Incremental Flows" heading • Merged the "Planning Design Partitions" section with the "Creating a Design Floorplan" section. Changed heading title to "Planning Design Partitions and Floorplan Location Assignments" on page 1–15 • Removed "Creating a Design Floorplan" section • Removed "Referenced Documents" section • Minor updates throughout chapter
November 2009	9.1.0	• Added details to "Creating Design Specifications" on page 1–2 • Added details to "Intellectual Property Selection" on page 1–2 • Updated information on "Device Selection" on page 1–3 • Added reference to "Device Migration Planning" on page 1–4 • Removed information from "Planning for Device Programming or Configuration" on page 1–4 • Added details to "Early Power Estimation" on page 1–5 • Updated information on "Early Pin Planning and I/O Analysis" on page 1–6 • Updated information on "Creating a Top-Level Design File for I/O Analysis" on page 1–8 • Added new "Simultaneous Switching Noise Analysis" section • Updated information on "Synthesis Tools" on page 1–9 • Updated information on "Simulation Tools" on page 1–9 • Updated information on "Planning for On-Chip Debugging Options" on page 1–10

continued...

Date	Version	Changes
		• Added new "Managing Metastability" section • Changed heading title "Top-Down Versus Bottom-Up Incremental Flows" to "Single-Project Versus Multiple-Project Incremental Flows" • Updated information on "Creating a Design Floorplan" on page 1–18 • Removed information from "Fast Synthesis and Early Timing Estimation" on page 1–18
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	• Organization changes • Added "Creating Design Specifications" section • Added reference to new details in the In-System Design Debugging section of volume 3 • Added more details to the "Design Practices and HDL Coding Styles" section • Added references to the new Best Practices for Incremental Compilation and Floorplan Assignments chapter • Added reference to the Intel Quartus Prime Language Templates

4. Introduction to Intel FPGA IP Cores

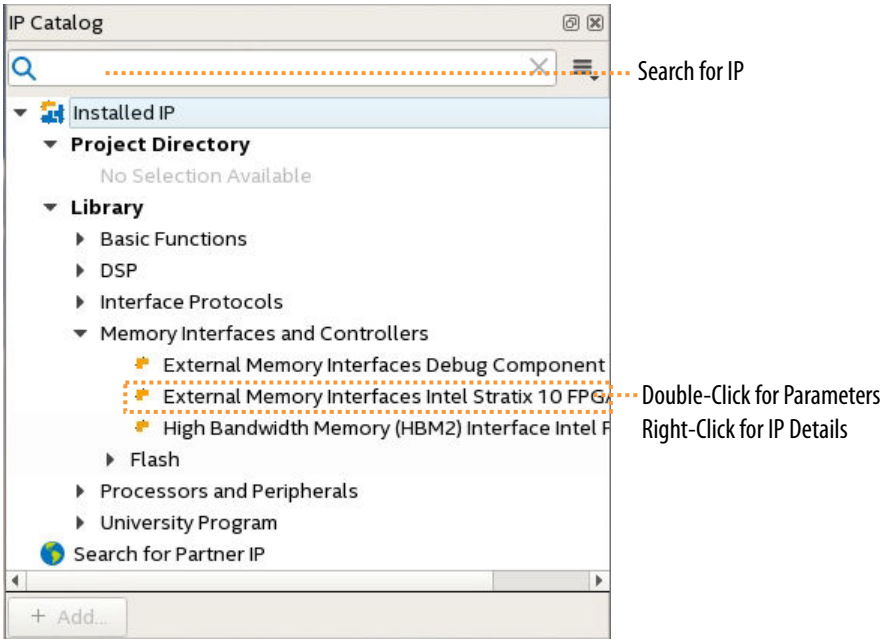
Intel and strategic IP partners offer a broad portfolio of configurable IP cores optimized for Intel FPGA devices.

The Intel 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 Intel 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
- DSP functions
- Interface protocols
- Low power functions
- Memory interfaces and controllers
- Processors and peripherals

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

Figure 25. IP Catalog



Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

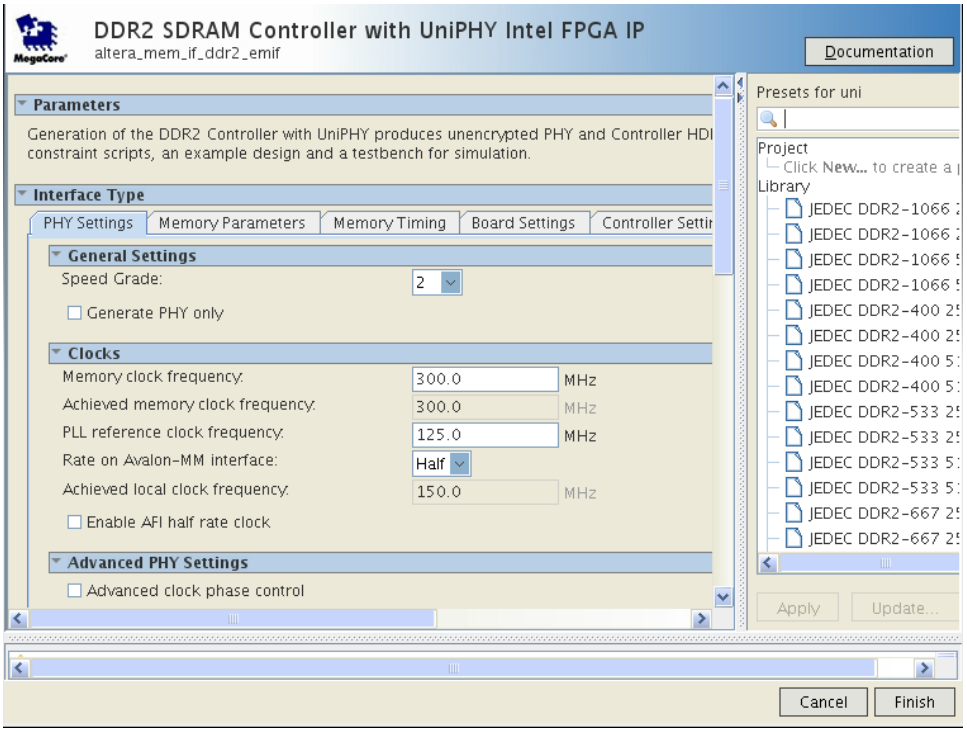
4.1. IP Catalog and Parameter Editor

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 generates a top-level Quartus IP file (`.qip`) for an IP variation in Intel Quartus Prime Standard Edition projects. These files represent the IP variation in the project, and store parameterization information.

Figure 26. IP Parameter Editor (Intel Quartus Prime Standard Edition)



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).

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 Intel Quartus Prime IP Catalog. Refer to *Creating a System with Platform Designer* or *Creating a System with Platform Designer (Standard)* for information on use of IP in Platform Designer (Standard) and Platform Designer, respectively.

Related Information

Creating a System with Platform Designer (Standard)

4.2. Installing and Licensing Intel FPGA IP Cores

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

Figure 27. IP Core Installation Path

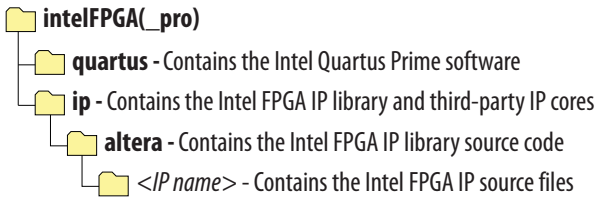


Table 8. IP Core Installation Locations

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

Note: The Intel 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.

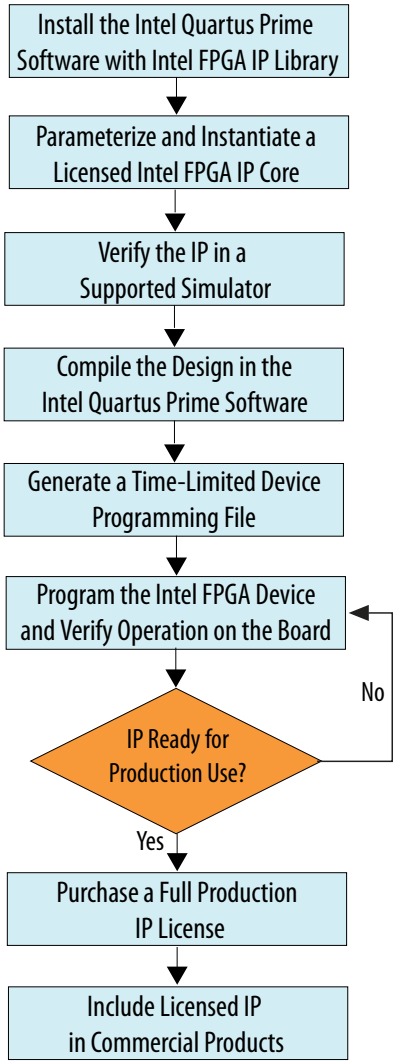
Intel FPGA IP Evaluation Mode supports the following operation modes:

- **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 Intel Quartus Prime Programmer for the duration of the hardware evaluation period. The Programmer only requires a minimum installation of the Intel Quartus Prime software, and requires no Intel 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 Intel 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 28. 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 Self-Service Licensing Center.

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

Related Information

- Intel Quartus Prime Licensing Site
- Introduction to Intel FPGA Software Installation and Licensing

4.2.1.1. Intel FPGA IP Versioning

IP versions are the same as the Intel Quartus Prime Design Suite software versions up to v19.1. From Intel Quartus Prime Design Suite software version 19.2 or later, IP cores have a new IP versioning scheme.

The IP versioning scheme (X.Y.Z) number changes from one software version to another. A change in:

- X indicates a major revision of the IP. If you update your Intel 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.

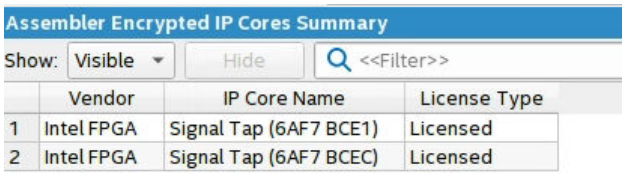
4.2.1.2. Checking the IP License Status

You can check the license status of all IP in an Intel 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.
3. Click the **Encrypted IP Cores Summary** report.

Figure 29. 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:

```
<project>/output_files/<project_name>.asm.rpt
```

```
Example of the assembler report:

+-----------------------------------------------------------------------+
; 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 Intel Quartus Prime software manages IP cores in a project:

Table 9. Location of IP Core General Settings in the Intel Quartus Prime Software

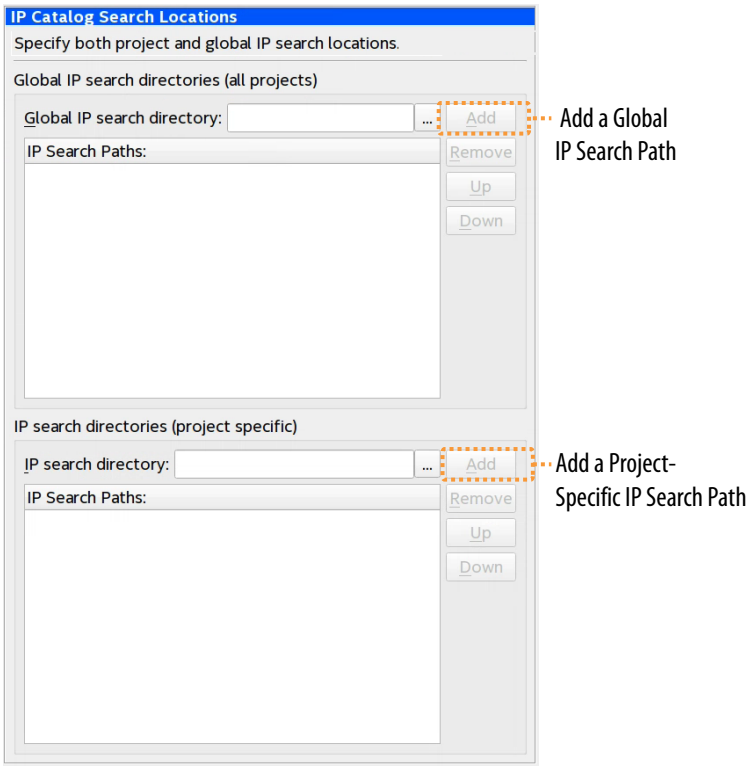
Setting	Description	Location
Maximum Platform Designer memory usage size	Increase if you experience slow processing for large systems, or for out of memory errors.	**Tools ➤ Options ➤ IP Settings** Or **Tasks pane ➤ Settings ➤ IP Settings**
IP generation HDL preference	The parameter editor generates 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.	
Additional project and global IP search locations. The Intel Quartus Prime software searches for IP cores in the project directory, in the Intel 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 Your Own 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 Intel 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.

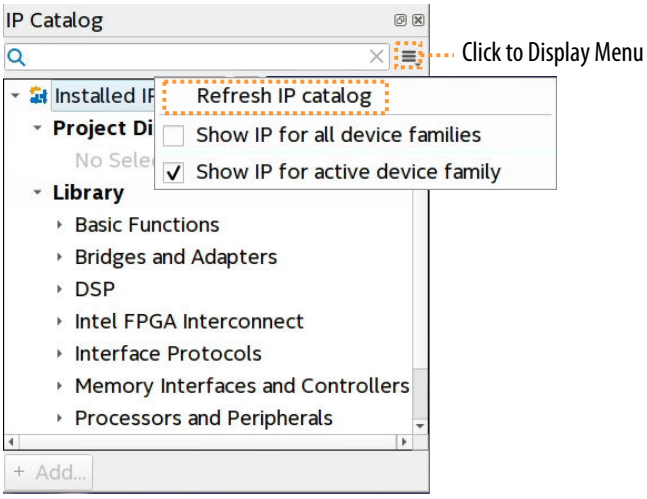
Follow these steps to add custom or third-party IP to the IP Catalog:

Figure 30. Specifying IP Search Locations



1. In the Intel 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 Intel Quartus Prime Platform Designer (Standard), or click **File ➤ Refresh System**in Platform Designer (Standard).

Figure 31. 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 Intel Quartus Prime software tools to create and edit these files.

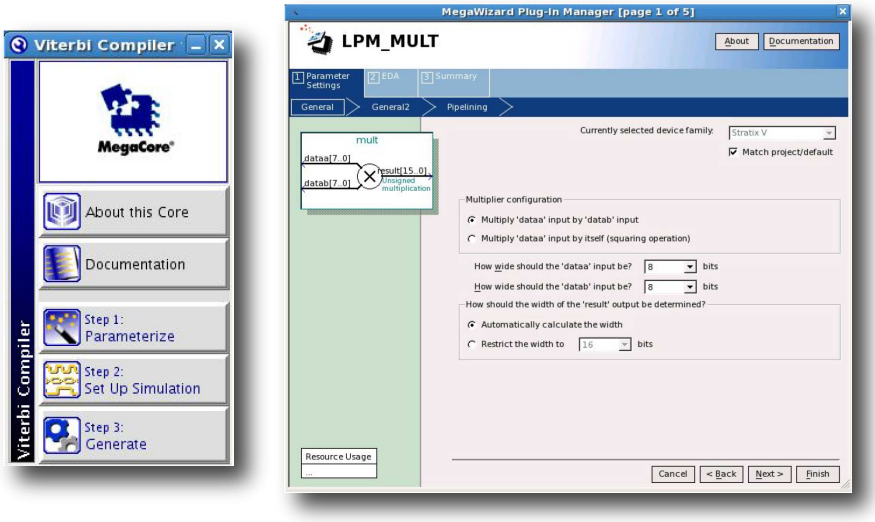
  Note: When generating IP cores, do not generate files into a directory that has a space in the directory name or path. Spaces are not legal characters for IP core paths or names.
- When you generate an IP core using the IP Catalog, the Intel Quartus Prime software generates a `.qsys` (for Platform Designer (Standard)-generated IP cores) or a `.ip` file (for Intel Quartus Prime Pro Edition) or a `.qip` file. The Intel Quartus Prime Pro Edition software automatically adds the generated `.ip` to your project. In the Intel 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 Intel Quartus Prime software generates for RAM or FIFO blocks targeting older device families (even though the Intel Quartus Prime software does not issue an error). The RAM blocks that Intel 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 Intel Quartus Prime `ip-setup-simulation` and `ip-make-simscript` utilities to generate simulation scripts for each IP core or Platform Designer (Standard) system in your design. These utilities produce a single simulation script that does not require manual update for upgrades to Intel Quartus Prime software or IP versions, as Simulating Intel FPGA IP Cores on page 69 describes.

4.6. Generating IP Cores (Intel Quartus Prime Standard Edition)

This topic describes parameterizing and generating an IP variation using a legacy parameter editor in the Intel Quartus Prime Standard Edition software.

Figure 32. Legacy Parameter Editors



Note: The legacy parameter editor generates a different output file structure than the Intel Quartus Prime Pro Edition software.

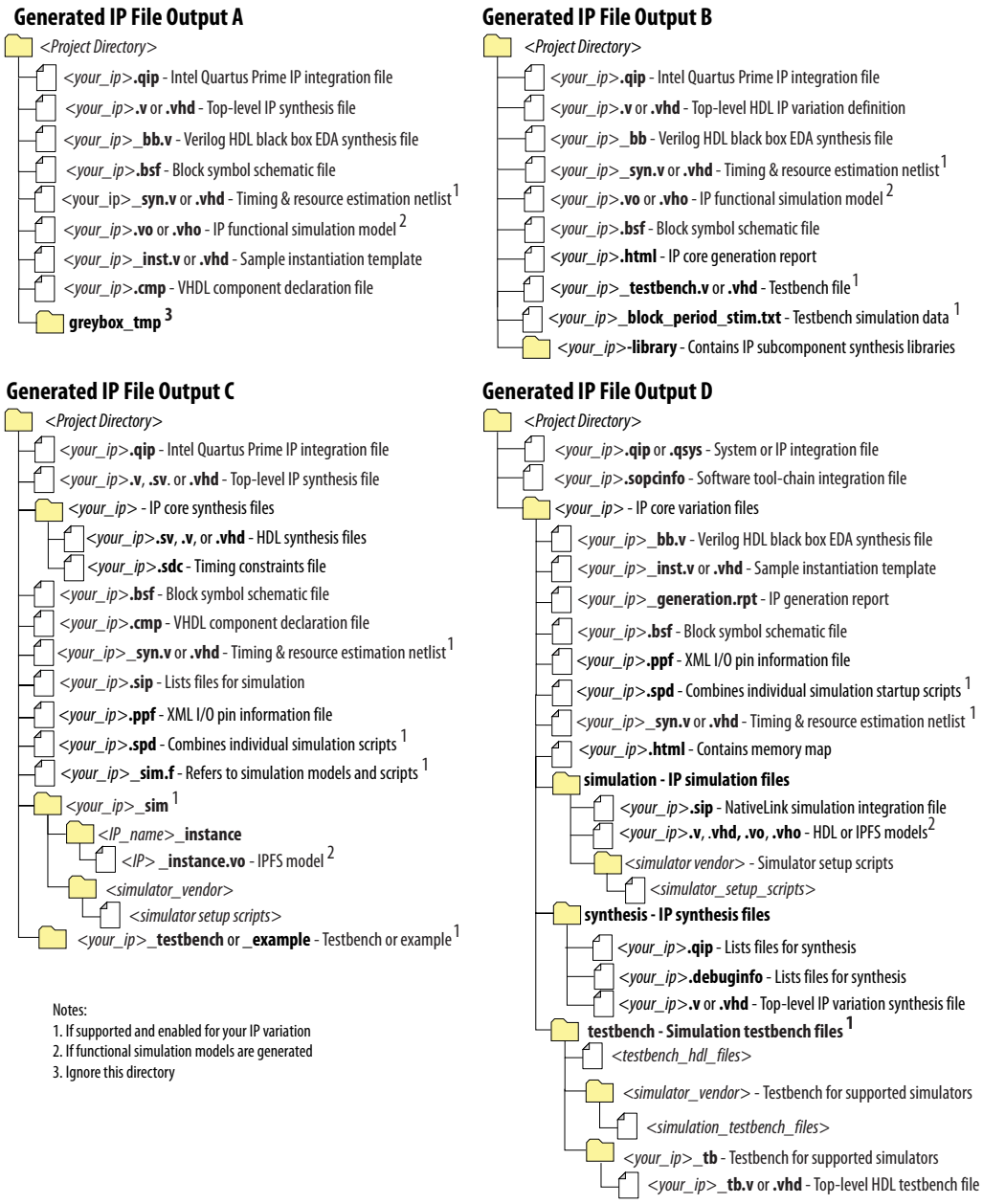
1. In the IP Catalog (**Tools ➤ IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name and output HDL file type for your IP variation. This name identifies the IP core variation files in your project. Click **OK**. Do not include spaces in IP variation names or paths.
3. Specify the parameters and options for your IP variation in the parameter editor. Refer to your IP core user guide for information about specific IP core parameters.
4. Click **Finish** or **Generate** (depending on the parameter editor version). The parameter editor generates the files for your IP variation according to your specifications. Click **Exit** if prompted when generation is complete. The parameter editor adds the top-level `.qip` file to the current project automatically.

   Note: For devices released prior to Intel Arria 10 devices, the generated `.qip` and `.sip` files must be added to your project to represent IP and Platform Designer systems. To manually add an IP variation generated with legacy parameter editor to a project, click **Project ➤ Add/Remove Files in Project** and add the IP variation `.qip` file.

4.6.1. IP Core Generation Output (Intel Quartus Prime Standard Edition)

The Intel Quartus Prime Standard Edition software generates one of the following output file structures for individual IP cores that use one of the legacy parameter editors.

Figure 33. IP Core Generated Files (Legacy Parameter Editors)



4.7. 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.

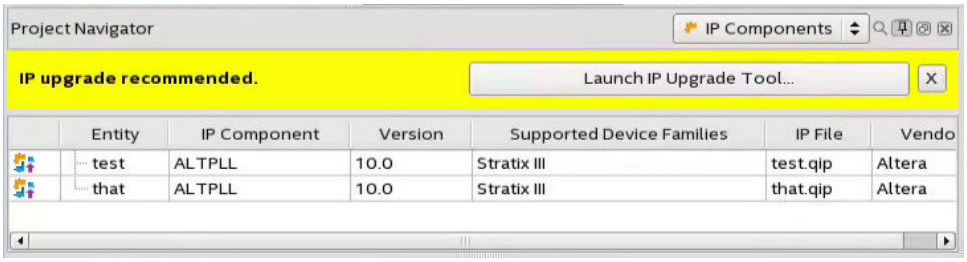
Table 10. Modifying an IP Variation

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 ➤ Utility Windows ➤ 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.

4.8. Upgrading IP Cores

Any Intel FPGA IP variations that you generate from a previous version or different edition of the Intel 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 34. 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 Intel 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 Intel Quartus Prime file, such as the Intel Quartus Prime Settings File (`.qsf`), Synopsys* Design Constraints File (`.sdc`), or Signal Tap File (`.stp`), if these files contain instance names. The Intel 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 11. 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.

continued...

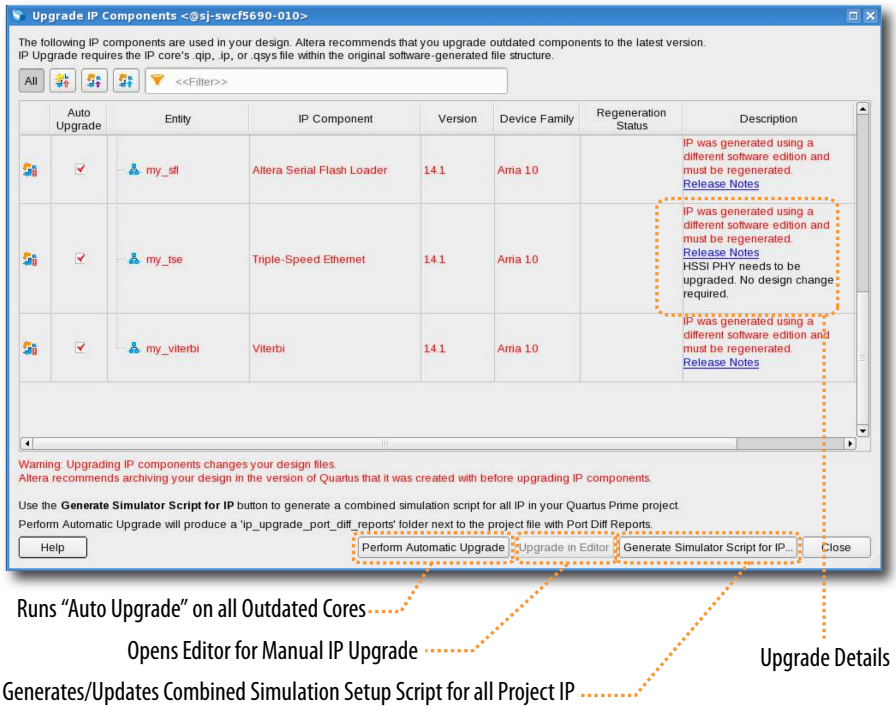
IP Core Status	Description
IP Component Outdated	Indicates that your IP variation uses an outdated version of the IP core.
IP Upgrade Optional	Indicates that upgrade is optional for this IP variation in the current version of the Intel Quartus Prime software. You can upgrade this IP variation to take advantage of the latest development of this IP core. Alternatively, you can retain previous IP core characteristics by declining to upgrade. Refer to the Description for details about IP core version differences. If you do not upgrade the IP, the IP variation synthesis and simulation files are unchanged and you cannot modify parameters until upgrading.
IP Upgrade Required	Indicates that you must upgrade the IP variation before compiling in the current version of the Intel Quartus Prime software. Refer to the Description for details about IP core version differences.
IP Upgrade Unsupported	Indicates that upgrade of the IP variation is not supported in the current version of the Intel Quartus Prime software due to incompatibility with the current version of the Intel Quartus Prime software. The Intel Quartus Prime software prompts you to replace the unsupported IP core with a supported equivalent IP core from the IP Catalog. Refer to the Description for details about IP core version differences and links to Release Notes.
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 Intel 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 Intel 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 Intel Quartus Prime software does not support compilation of your IP variation. This can occur if another edition of the Intel Quartus Prime software generated this IP. Replace this IP component with a compatible component in the current edition.

Follow these steps to upgrade IP cores:

1. In the latest version of the Intel Quartus Prime software, open the Intel 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 . 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 35. Upgrading IP Cores



Note: Intel FPGA IP cores older than Intel Quartus Prime software version 12.0 do not support upgrade. Intel verifies that the current version of the Intel 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

4.8.1. 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
```

4.8.2. 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

4.8.3. Troubleshooting IP or Platform Designer System Upgrade

The **Upgrade IP Components** dialog box reports the version and status of each Avalon® 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:

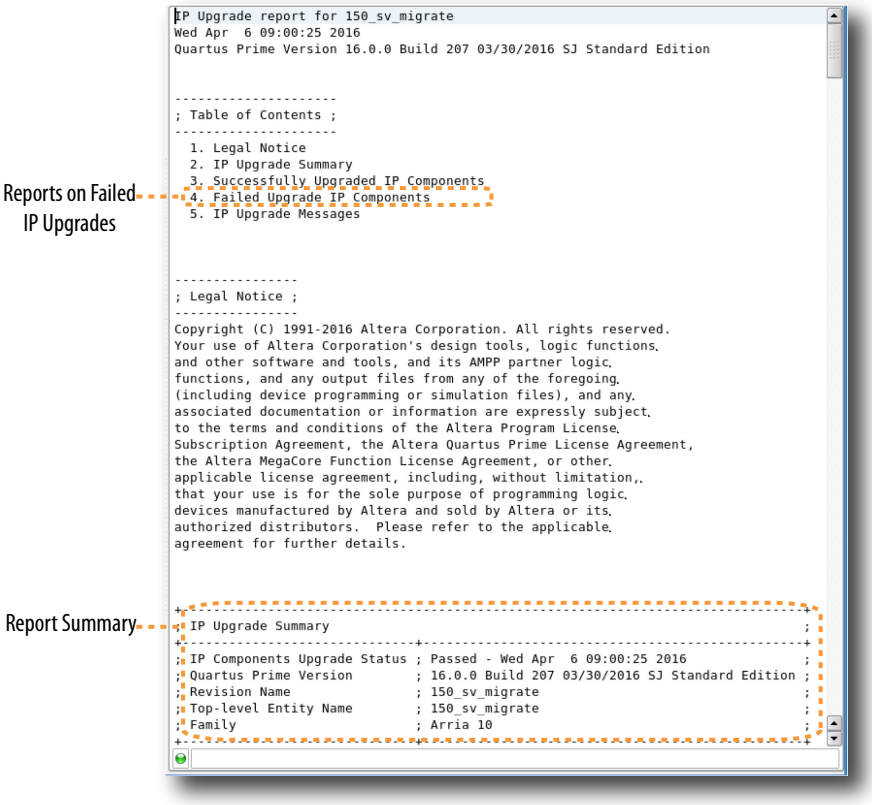
Table 12. IP Upgrade Error Information

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_report` 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 fails 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 36. IP Upgrade Report



4.9. Simulating Intel FPGA IP Cores

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

4.9.1. Generating IP Simulation Files

The Intel 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 control the generation of IP simulation files:

- To specify your supported simulator and options for IP simulation file generation, click **Assignment ➤ Settings ➤ EDA Tool Settings ➤ Simulation**.
- To parameterize a new IP variation, enable generation of simulation files, and generate the IP core synthesis and simulation files, click **Tools ➤ IP Catalog**.
- To edit parameters and regenerate synthesis or simulation files for an existing IP core variation, click **View ➤ Project Navigator ➤ IP Components**.
- To edit parameters and regenerate synthesis or simulation files for an existing IP core variation, click **View ➤ Utility Windows ➤ Project Navigator ➤ IP Components**.

Table 13. Intel FPGA IP Simulation Files

File Type	Description	File Name
Simulator setup scripts	Vendor-specific scripts to compile, elaborate, and simulate Intel FPGA IP models and simulation model library files.	<*my_dir*>/aldec/riviera_setup.tcl <*my_dir*>/cadence/ncsim__setup.sh <*my_dir*>/mentor/msim_setup.tcl <*my_dir*>/synopsys/vcs/vcs_setup.sh <*my_dir*>/synopsys/vcsmx/vcsmx_setup.sh

continued...

File Type	Description	File Name
	Note: For Intel Arria 10 designs, you can use the Intel Quartus Prime software to automatically create a combined simulator setup script. Refer to *Scripting IP Simulation* in the *Introduction to Intel FPGA IP Cores* for more information.	
Simulation IP File (Intel Quartus Prime Standard Edition)	Contains IP core simulation library mapping information. To use NativeLink, add the `.qip` and `.sip` files generated for IP to your project.	`<design name>.sip`
IP functional simulation models (Intel Quartus Prime Standard Edition)	IP functional simulation models are cycle-accurate VHDL or Verilog HDL models a that the Intel Quartus Prime software generates for some Intel FPGA IP cores. IP functional simulation models support fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.	`<my_ip>.vho` `<my_ip>.vo`
IEEE encrypted models (Intel Quartus Prime Standard Edition)	Intel provides Arria V, Cyclone V, Stratix V, and newer simulation model libraries and IP simulation models in Verilog HDL and IEEE-encrypted Verilog HDL. Your simulator's co-simulation capabilities support VHDL simulation of these models. IEEE encrypted Verilog HDL models are significantly faster than IP functional simulation models. The Intel Quartus Prime Pro Edition software does not support these models.	`<my_ip>.v`

Note: Intel FPGA IP cores support a variety of cycle-accurate simulation models, including simulation-specific IP functional simulation models and encrypted RTL models, and plain text RTL models. The models support fast functional simulation of your IP core instance using industry-standard VHDL or Verilog HDL simulators. For some IP cores, generation only produces the plain text RTL model, and you can simulate that model. Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

4.9.2. Using NativeLink Simulation (Intel Quartus Prime Standard Edition)

The NativeLink feature integrates your EDA simulator with the Intel Quartus Prime Standard Edition software by automating the following:

- Generation of simulator-specific files and simulation scripts.
- Compilation of simulation libraries.
- Launches your simulator automatically following Intel Quartus Prime Analysis & Elaboration, Analysis & Synthesis, or after a full compilation.

Note: The Intel Quartus Prime Pro Edition does not support NativeLink simulation. If you use NativeLink for Intel Arria 10 devices in the Intel Quartus Prime Standard Edition, you must add the `.qsys` file generated for the IP or Platform Designer (Standard) system to your Intel Quartus Prime project. If you use NativeLink for any other supported device family, you must add the `.qip` and `.sip` files to your project.

4.9.2.1. Setting Up NativeLink Simulation (Intel Quartus Prime Standard Edition)

Before running NativeLink simulation, specify settings for your simulator in the Intel Quartus Prime software.

To specify NativeLink settings in the Intel Quartus Prime Standard Edition software, follow these steps:

1. Open an Intel Quartus Prime Standard Edition project.
2. Click **Tools > Options** and specify the location of your simulator executable file.

Table 14. Execution Paths for EDA Simulators

Simulator	Path
Mentor Graphics ModelSim-AE	*<drive letter>*:*<simulator install path>*\win32aloem (Windows) /*<simulator install path>*/bin (Linux)
Mentor Graphics ModelSim Mentor Graphics QuestaSim	*<drive letter>*:*<simulator install path>*\win32 (Windows) *<simulator install path>*/bin (Linux)
Synopsys VCS/VCS MX	*<simulator install path>*/bin (Linux)
Cadence Incisive Enterprise	*<simulator install path>*/tools/bin (Linux)
Aldec Active-HDL Aldec Riviera-PRO	*<drive letter>*:*<simulator install path>*\bin (Windows) *<simulator install path>*/bin (Linux)

3. Click **Assignments ➤ Settings** and specify options on the **Simulation** page and the **More NativeLink Settings** dialog box. Specify default options for simulation library compilation, netlist and tool command script generation, and for launching RTL or gate-level simulation automatically following compilation.
4. If your design includes a testbench, turn on **Compile test bench**. Click **Test Benches** to specify options for each testbench. Alternatively, turn on **Use script to compile testbench** and specify the script file.
5. To use a script to setup a simulation, turn on **Use script to setup simulation**.

4.9.2.2. Generating IP Functional Simulation Models (Intel Quartus Prime Standard Edition)

Intel provides IP functional simulation models for some Intel FPGA IP supporting 40nm FPGA devices.

To generate IP functional simulation models:

1. Turn on the **Generate Simulation Model** option when parameterizing the IP core.
2. When you simulate your design, compile only the `.vo` or `.vho` for these IP cores in your simulator. Do not compile the corresponding HDL file. The encrypted HDL file supports synthesis by only the Intel Quartus Prime software.

   Note:
   - Intel FPGA IP cores that do not require IP functional simulation models for simulation, do not provide the **Generate Simulation Model** option in the IP core parameter editor.
   - Many recently released Intel FPGA IP cores support RTL simulation using IEEE Verilog HDL encryption. IEEE encrypted models are significantly faster than IP functional simulation models. Simulate the models in both Verilog HDL and VHDL designs.

Related Information

AN 343: Intel FPGA IP Evaluation Mode of AMPP IP

4.10. 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 Intel 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 Intel Quartus Prime project.

4.11. 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.

4.11.1. 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
```

4.11.2. 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;
```

4.12. Introduction to Intel FPGA IP Cores Revision History

This chapter has the following revision history.

Document Version	Intel Quartus Prime Version	Changes
2019.05.13	18.1.0	• Added archives topic. • Updated the keyname and added --help information to "Support for the IEEE 1735 Encryption Standard."
2018.10.24	18.1.0	• Updated information about obtaining IEEE 1735 Encryption key.
2018.09.24	18.1.0	• Added statement that the Intel 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*.
2018.05.07	18.0.0	• 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*
2017.11.06	17.1.0	• Revised product branding for Intel standards. • Revised topics on Intel FPGA IP Evaluation Mode (formerly OpenCore).
2017.05.08	17.0.0	• Added note that IP core encryption is supported only in Intel Quartus Prime Pro Edition. • Revised product branding for Intel standards.
2016.10.31	16.1.0	• Removed references to `.qsys` file creation during Intel Quartus Prime Pro Edition stand-alone IP generation. • Added references to `.ip` file creation during Intel Quartus Prime Pro Edition stand-alone IP generation. • Updated IP Core Generation Output files list and diagram. • Indicated distinctions between Intel Quartus Prime Pro Edition and Intel Quartus Prime Standard Edition features. • Added Support for IP Core Encryption topic.

5. Migrating to Intel Quartus Prime Pro Edition

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

Note: The migration steps for Quartus Prime Lite Edition, Intel 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 Intel Quartus Prime Pro Edition requires the following changes to other Quartus software product projects:

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

This document describes each migration step in detail.

5.1. Keep Pro Edition Project Files Separate

The Intel 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 Intel Quartus Prime Pro Edition project files. In general, use Intel Quartus Prime Pro Edition project files and directories only for Intel Quartus Prime Pro Edition projects, and use other Quartus software product files only with those software tools.

Intel Quartus Prime Pro Edition projects do not support compilation in other Quartus software products, and vice versa. The Intel 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.

5.2. Upgrade Project Assignments and Constraints

Intel 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 Intel 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

The following sections detail each type project assignment upgrade that migration requires.

Related Information

- Modify Entity Name Assignments on page 75
- Resolve Timing Constraint Entity Names on page 75
- Verify Generated Node Name Assignments on page 76
- Replace Logic Lock (Standard) Regions on page 76
- Modify Signal Tap Logic Analyzer Files on page 78
- Remove Unsupported Feature Assignments on page 79

5.2.1. Modify Entity Name Assignments

Intel 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 Intel 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 Intel Quartus Prime Pro Edition software may eliminate all support for entity-based assignments.

5.2.2. Resolve Timing Constraint Entity Names

The Intel 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
```

5.2.3. Verify Generated Node Name Assignments

Intel Quartus Prime synthesis generates and automatically names internal design nodes during processing. The Intel 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 Intel 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 Intel 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.

5.2.4. Replace Logic Lock (Standard) Regions

Intel 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 Intel Quartus Prime Pro Edition project. The following table compares other Quartus software product region constraint support with the Intel Quartus Prime Pro Edition software.

Table 15. Region Constraints Per Edition

Constraint Type	Logic Lock (Standard) Region Support Other Quartus Software Products	Logic Lock Region Support Intel 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.

5.2.4.1. Logic Lock Region Assignment Examples

These examples show the syntax of Logic Lock region assignments in the `.qsf` file. Optionally, 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 right 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 Intel 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"
```

5.2.5. Modify Signal Tap Logic Analyzer Files

Intel 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 Intel 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 Intel Quartus Prime Pro Edition:
   a. Click **View ➤ Utility Windows ➤ 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. Intel 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 ➤ Utility Windows ➤ 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 Intel 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 Intel Quartus Prime Pro Edition. Your scripts require no changes to `quartus_stp` for migration.
4. Modify `.sdc` constraints for JTAG. Intel 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.

5.2.6. Remove References to .qip Files

In Intel Quartus Prime Standard Edition projects, Platform Designer (Standard) (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 Intel 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 Intel Quartus Prime Pro Edition, remove all references to `.qip` files from the `.qsf` file.

5.2.7. Remove Unsupported Feature Assignments

The Intel 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 Intel Quartus Prime Pro Edition software.

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

5.3. Upgrade IP Cores and Platform Designer (Standard) Systems

Upgrade all IP cores and Platform Designer (Standard) systems in your project for migration to the Intel Quartus Prime Pro Edition software. The Intel 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 (Standard) systems for synthesis files. The Intel Quartus Prime Pro Edition does not support this scheme. To upgrade all IP cores and Platform Designer (Standard) systems in your project, click **Project ➤ Upgrade IP Components**.(1)

Table 16. IP Core and Platform Designer (Standard) System Differences

Other Quartus Software Products	Intel Quartus Prime Pro Edition
IP and Platform Designer (Standard) system generation use a proprietary Verilog HDL configuration scheme within the top level of IP cores and Platform Designer (Standard) 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. Intel 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. Intel 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 Intel 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.

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

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

5.4. Upgrade Non-Compliant Design RTL

The Intel 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 Intel Quartus Prime Pro Edition synthesis.

Related Information

- Verify Verilog Compilation Unit on page 81
- Update Entity Auto-Discovery on page 82
- Ensure Distinct VHDL Namespace for Each Library on page 83
- Remove Unsupported Parameter Passing on page 83
- Remove Unsized Constant from WYSIWYG Instantiation on page 83
- Remove Non-Standard Pragmas on page 84
- Declare Objects Before Initial Values on page 84
- Confine SystemVerilog Features to SystemVerilog Files on page 84
- Avoid Assignment Mixing in Always Blocks on page 85
- Avoid Unconnected, Non-Existent Ports on page 85
- Avoid Illegal Parameter Ranges on page 85
- Update Verilog HDL and VHDL Type Mapping on page 86

5.4.1. Verify Verilog Compilation Unit

Intel 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 17. Verilog Compilation Unit Differences

Other Quartus Software Products	Intel 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.	Intel 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`

5.4.1.1. Verilog HDL Configuration Instantiation

Intel 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 18. Verilog HDL Configuration Instantiation

Other Quartus Software Products	Intel 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`	

5.4.2. Update Entity Auto-Discovery

All editions of the Intel 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, Intel 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 19. Entity Auto-Discovery Differences

Other Quartus Software Products	Intel 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. Intel 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`

5.4.3. Ensure Distinct VHDL Namespace for Each Library

Intel 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 20. VHDL Namespace Differences

Other Quartus Software Products	Intel 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;`	

5.4.4. Remove Unsupported Parameter Passing

Intel 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 21. SystemVerilog Feature Differences

Other Quartus Software Products	Intel 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`	

5.4.5. Remove Unsized Constant from WYSIWYG Instantiation

Intel 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.

Intel 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.

5.4.6. Remove Non-Standard Pragmas

Intel 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 Intel Quartus Prime Pro Edition migration. Use the following guidelines to implement the pragma functionality in Intel 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 Intel 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 Intel Quartus Prime Pro Edition migration, specify the compilation library in the `.qsf`.

5.4.7. Declare Objects Before Initial Values

Intel 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 22. Object Declaration Differences

Other Quartus Software Products	Intel 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;`	

5.4.8. Confine SystemVerilog Features to SystemVerilog Files

Intel 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 Intel 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 23. SystemVerilog Feature Differences

Other Quartus Software Products	Intel 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. Intel 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];`	

5.4.9. Avoid Assignment Mixing in Always Blocks

Intel 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 Intel Quartus Prime Pro Edition migration.

Table 24. ALWAYS Block Assignment Differences

Other Quartus Software Products	Intel 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.

5.4.10. Avoid Unconnected, Non-Existent Ports

Intel 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 Intel Quartus Prime Pro Edition migration.

Table 25. Unconnected, Non-Existent Port Differences

Other Quartus Software Products	Intel 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.

5.4.11. Avoid Illegal Parameter Ranges

Intel 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.

5.4.12. Update Verilog HDL and VHDL Type Mapping

Intel 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. Intel 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.

5.5. Migrating to Intel Quartus Prime Pro Edition Revision History

This chapter has the following revision history.

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.

A. Intel Quartus Prime Pro Edition User Guide: Getting Started Documentation Archive

If the table does not list a software version, the user guide for the previous software version applies.

Intel Quartus Prime Version	User Guide
18.1	Intel Quartus Prime Pro Edition User Guide: Getting Started

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

B. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Platform Designer

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Creating a System with Platform Designer.....10
1.1. Platform Designer Interface Support.....10
1.2. Platform Designer System Design Flow.....12
1.3. Starting or Opening a Project in Platform Designer.....12
1.4. Viewing a Platform Designer System.....12
1.4.1. Viewing the System Hierarchy.....13
1.4.2. Filtering the System View.....14
1.4.3. Viewing Clock and Reset Domains.....15
1.4.4. Viewing Avalon Memory-Mapped Domains in a System.....17
1.4.5. Viewing the System Schematic.....19
1.4.6. Viewing System Assignments and Connections.....19
1.4.7. Customizing the Platform Designer Layout.....20
1.5. Adding IP Components to a System.....21
1.5.1. Modifying IP Parameters.....23
1.5.2. Applying Preset Parameters for Specific Applications.....25
1.5.3. Adding Third-Party IP Components.....26
1.5.4. Creating or Opening an IP Core Variant.....28
1.6. Connecting System Components.....29
1.6.1. Platform Designer 64-Bit Addressing Support.....31
1.6.2. Connecting Masters and Slaves.....32
1.6.3. Changing a Conduit to a Reset.....33
1.6.4. Wire-Level Connectivity.....33
1.6.5. Previewing the System Interconnect.....38
1.7. Specifying Interconnect Requirements.....39
1.7.1. Interconnect Requirements.....40
1.8. Defining Instance Parameters.....41
1.8.1. Creating an Instance Parameter Script in Platform Designer.....42
1.8.2. Platform Designer Instance Parameter Script Tcl Commands.....43
1.9. Implementing Performance Monitoring.....49
1.10. Configuring Platform Designer System Security.....50
1.10.1. System Security Options.....52
1.10.2. Specifying a Default Slave.....52
1.10.3. Accessing Undefined Memory Regions.....53
1.11. Upgrading Outdated IP Components.....54
1.11.1. Troubleshooting IP or Platform Designer System Upgrade.....55
1.12. Synchronizing System Component Information.....56
1.13. Generating a Platform Designer System.....57
1.13.1. Generation Dialog Box Options.....58
1.13.2. Specifying the Generation ID.....58
1.13.3. Files Generated for IP Cores and Platform Designer Systems.....59
1.13.4. Generating System Testbench Files.....60
1.13.5. Generating Example Designs for IP Components.....63
1.13.6. Generating the HPS IP Component System View Description File.....64
1.13.7. Generating Header Files for Master Components.....64
1.14. Simulating a Platform Designer System.....65
1.14.1. Adding Assertion Monitors for Simulation.....66
1.14.2. Simulating Software Running on a Nios II Processor.....66

1.15. Integrating a Platform Designer System with the Intel Quartus Prime Software..... 67
1.15.1. Integrate a Platform Designer System and the Intel Quartus Prime Software With the .qsys File..... 68
1.15.2. Integrate a Platform Designer System and the Intel Quartus Prime Software With the .qip File..... 68
1.16. Managing Hierarchical Platform Designer Systems..... 69
1.16.1. Adding a Subsystem to a Platform Designer System.....69
1.16.2. Viewing and Traversing Subsystem Contents.....70
1.16.3. Editing a Subsystem..... 71
1.16.4. Changing a Component's Hierarchy Level..... 71
1.16.5. Saving a Subsystem..... 72
1.16.6. Exporting a System as an IP Component..... 72
1.16.7. Hierarchical System Using Instance Parameters Example..... 73
1.17. Creating a System with Platform Designer Revision History.....78

2. Optimizing Platform Designer System Performance.....80
2.1. Designing with Avalon and AXI Interfaces..... 80
2.1.1. Designing Streaming Components.....81
2.1.2. Designing Memory-Mapped Components..... 81
2.2. Using Hierarchy in Systems.....82
2.3. Using Concurrency in Memory-Mapped Systems.....85
2.3.1. Implementing Concurrency With Multiple Masters.....86
2.3.2. Implementing Concurrency With Multiple Slaves.....87
2.3.3. Implementing Concurrency with DMA Engines.....89
2.4. Inserting Pipeline Stages to Increase System Frequency..... 90
2.5. Using Bridges.....90
2.5.1. Using Bridges to Increase System Frequency.....91
2.5.2. Using Bridges to Minimize Design Logic.....94
2.5.3. Using Bridges to Minimize Adapter Logic..... 96
2.5.4. Considering the Effects of Using Bridges.....97
2.6. Increasing Transfer Throughput.....103
2.6.1. Using Pipelined Transfers.....104
2.6.2. Arbitration Shares and Bursts..... 105
2.7. Reducing Logic Utilization.....109
2.7.1. Minimizing Interconnect Logic to Reduce Logic Unitization.....109
2.7.2. Minimizing Arbitration Logic by Consolidating Multiple Interfaces..... 110
2.7.3. Reducing Logic Utilization With Multiple Clock Domains.....112
2.7.4. Duration of Transfers Crossing Clock Domains 114
2.8. Reducing Power Consumption..... 115
2.8.1. Reducing Power Consumption With Multiple Clock Domains..... 115
2.8.2. Reducing Power Consumption by Minimizing Toggle Rates..... 118
2.8.3. Reducing Power Consumption by Disabling Logic..... 119
2.9. Reset Polarity and Synchronization in Platform Designer.....120
2.10. Optimizing Platform Designer System Performance Design Examples.....123
2.10.1. Avalon Pipelined Read Master Example..... 123
2.10.2. Multiplexer Examples..... 125
2.11. Optimizing Platform Designer System Performance Revision History..... 127

3. Platform Designer Interconnect.....128
3.1. Memory-Mapped Interfaces..... 128
3.1.1. Platform Designer Packet Format.....130
3.1.2. Interconnect Domains.....133

3.1.3. Master Network Interfaces....135
3.1.4. Slave Network Interfaces....138
3.1.5. Arbitration....140
3.1.6. Memory-Mapped Arbiter....144
3.1.7. Datapath Multiplexing Logic....146
3.1.8. Width Adaptation....146
3.1.9. Burst Adapter....148
3.1.10. Waitrequest Allowance Adapter....150
3.1.11. Read and Write Responses....151
3.1.12. Platform Designer Address Decoding....152
3.2. Avalon Streaming Interfaces....152
3.2.1. Avalon-ST Adapters....154
3.3. Interrupt Interfaces....162
3.3.1. Individual Requests IRQ Scheme....162
3.3.2. Assigning IRQs in Platform Designer....163
3.4. Clock Interfaces....165
3.4.1. (High Speed Serial Interface) HSSI Clock Interfaces....166
3.5. Reset Interfaces....171
3.5.1. Single Global Reset Signal Implemented by Platform Designer....172
3.5.2. Reset Controller....172
3.5.3. Reset Bridge....172
3.5.4. Reset Sequencer....173
3.6. Conduits....184
3.7. Interconnect Pipelining....184
3.7.1. Manually Control Pipelining in the Platform Designer Interconnect....187
3.8. Error Correction Coding (ECC) in Platform Designer Interconnect....188
3.9. AMBA 3 AXI Protocol Specification Support (version 1.0)....188
3.9.1. Channels....188
3.9.2. Cache Support....189
3.9.3. Security Support....190
3.9.4. Atomic Accesses....190
3.9.5. Response Signaling....190
3.9.6. Ordering Model....190
3.9.7. Data Buses....191
3.9.8. Unaligned Address Commands....191
3.9.9. Avalon and AXI Transaction Support....191
3.10. AMBA 3 APB Protocol Specification Support (version 1.0)....192
3.10.1. Bridges....192
3.10.2. Burst Adaptation....192
3.10.3. Width Adaptation....192
3.10.4. Error Response....193
3.11. AMBA 4 AXI Memory-Mapped Interface Support (version 2.0)....193
3.11.1. Burst Support....193
3.11.2. QoS....193
3.11.3. Regions....193
3.11.4. Write Response Dependency....193
3.11.5. AWCACHE and ARCACHE....194
3.11.6. Width Adaptation and Data Packing in Platform Designer....194
3.11.7. Ordering Model....194
3.11.8. Read and Write Allocate....194
3.11.9. Locked Transactions....194

3.11.10. Memory Types....195
3.11.11. Mismatched Attributes....195
3.11.12. Signals....195
3.12. AMBA 4 AXI Streaming Interface Support (version 1.0)....195
3.12.1. Connection Points....195
3.12.2. Adaptation....196
3.13. AMBA 4 AXI-Lite Protocol Specification Support (version 2.0)....196
3.13.1. AMBA 4 AXI-Lite Signals....196
3.13.2. AMBA 4 AXI-Lite Bus Width....197
3.13.3. AMBA 4 AXI-Lite Outstanding Transactions....197
3.13.4. AMBA 4 AXI-Lite IDs....197
3.13.5. Connections Between AMBA 3 AXI,AMBA 4 AXI and AMBA 4 AXI-Lite....197
3.13.6. AMBA 4 AXI-Lite Response Merging....198
3.14. Port Roles (Interface Signal Types)....198
3.14.1. AXI Master Interface Signal Types....198
3.14.2. AXI Slave Interface Signal Types....199
3.14.3. AMBA 4 AXI Master Interface Signal Types....200
3.14.4. AMBA 4 AXI Slave Interface Signal Types....202
3.14.5. AMBA 4 AXI-Stream Master and Slave Interface Signal Types....203
3.14.6. ACE-Lite Interface Signal Roles....204
3.14.7. APB Interface Signal Types....204
3.14.8. Avalon Memory-Mapped Interface Signal Roles....204
3.14.9. Avalon Streaming Interface Signal Roles....208
3.14.10. Avalon Clock Source Signal Roles....209
3.14.11. Avalon Clock Sink Signal Roles....209
3.14.12. Avalon Conduit Signal Roles....209
3.14.13. Avalon Tristate Conduit Signal Roles....209
3.14.14. Avalon Tri-State Slave Interface Signal Types....211
3.14.15. Avalon Interrupt Sender Signal Roles....212
3.14.16. Avalon Interrupt Receiver Signal Roles....212
3.15. Platform Designer Interconnect Revision History....212

4. Platform Designer System Design Components....214

4.1. Bridges....214
4.1.1. Clock Bridge....215
4.1.2. Avalon-MM Clock Crossing Bridge....216
4.1.3. Avalon-MM Pipeline Bridge....218
4.1.4. Avalon-MM Unaligned Burst Expansion Bridge....219
4.1.5. Bridges Between Avalon and AXI Interfaces....222
4.1.6. AXI Bridge....223
4.1.7. AXI Timeout Bridge....228
4.1.8. Address Span Extender....232
4.2. Error Response Slave....237
4.2.1. Error Response Slave Parameters....238
4.2.2. Error Response Slave CSR Registers....239
4.2.3. Designating a Default Slave....242
4.3. Tri-State Components....243
4.3.1. Generic Tri-State Controller....245
4.3.2. Tri-State Conduit Pin Sharer....245
4.3.3. Tri-State Conduit Bridge....246
4.4. Test Pattern Generator and Checker Cores....246

4.4.1. Test Pattern Generator....247
4.4.2. Test Pattern Checker....249
4.4.3. Software Programming Model for the Test Pattern Generator and Checker Cores....250
4.4.4. Test Pattern Generator API....254
4.4.5. Test Pattern Checker API....259
4.5. Avalon-ST Splitter Core....266
4.5.1. Splitter Core Backpressure....266
4.5.2. Splitter Core Interfaces....267
4.5.3. Splitter Core Parameters....267
4.6. Avalon-ST Delay Core....268
4.6.1. Delay Core Reset Signal....268
4.6.2. Delay Core Interfaces....268
4.6.3. Delay Core Parameters....269
4.7. Avalon-ST Round Robin Scheduler....270
4.7.1. Almost-Full Status Interface (Round Robin Scheduler)....270
4.7.2. Request Interface (Round Robin Scheduler)....270
4.7.3. Round Robin Scheduler Operation....270
4.7.4. Round Robin Scheduler Parameters....271
4.8. Avalon Packets to Transactions Converter....272
4.8.1. Packets to Transactions Converter Interfaces....272
4.8.2. Packets to Transactions Converter Operation....272
4.9. Avalon-ST Streaming Pipeline Stage....274
4.10. Streaming Channel Multiplexer and Demultiplexer Cores....275
4.10.1. Software Programming Model For the Multiplexer and Demultiplexer Components....276
4.10.2. Avalon-ST Multiplexer....276
4.10.3. Avalon-ST Demultiplexer....278
4.11. Single-Clock and Dual-Clock FIFO Cores....279
4.11.1. Interfaces Implemented in FIFO Cores....280
4.11.2. FIFO Operating Modes....281
4.11.3. Fill Level of the FIFO Buffer....282
4.11.4. Almost-Full and Almost-Empty Thresholds to Prevent Overflow and Underflow....282
4.11.5. Single-Clock and Dual-Clock FIFO Core Parameters....282
4.11.6. Avalon-ST Single-Clock FIFO Registers....283
4.12. Platform Designer System Design Components Revision History....284

5. Creating Platform Designer Components....286

5.1. Platform Designer Components....286
5.1.1. Platform Designer Interface Support....286
5.1.2. Component Structure....288
5.1.3. Component File Organization....288
5.1.4. Component Versions....289
5.2. Design Phases of an IP Component....290
5.3. Create IP Components in the Platform Designer Component Editor....291
5.3.1. Save an IP Component and Create the _hw.tcl File....292
5.3.2. Edit an IP Component with the Platform Designer Component Editor....293
5.4. Specify IP Component Type Information....293
5.5. Create an HDL File in the Platform Designer Component Editor....295
5.6. Create an HDL File Using a Template in the Platform Designer Component Editor....295

5.7. Specify Synthesis and Simulation Files in the Platform Designer Component Editor..... 297
5.7.1. Specify HDL Files for Synthesis in the Platform Designer Component Editor....298
5.7.2. Analyze Synthesis Files in the Platform Designer Component Editor.....299
5.7.3. Name HDL Signals for Automatic Interface and Type Recognition in the Platform Designer Component Editor.....300
5.7.4. Specify Files for Simulation in the Component Editor.....301
5.7.5. Include an Internal Register Map Description in the .svd for Slave Interfaces Connected to an HPS Component..... 302
5.8. Add Signals and Interfaces in the Platform Designer Component Editor..... 303
5.9. Specify Parameters in the Platform Designer Component Editor.....304
5.9.1. Valid Ranges for Parameters in the _hw.tcl File.....306
5.9.2. Types of Platform Designer Parameters..... 307
5.9.3. Declare Parameters with Custom _hw.tcl Commands.....308
5.9.4. Validate Parameter Values with a Validation Callback.....310
5.10. Declaring SystemVerilog Interfaces in _hw.tcl.....310
5.11. User Alterable HDL Parameters in _hw.tcl..... 312
5.12. Scripting Wire-Level Expressions..... 314
5.13. Control Interfaces Dynamically with an Elaboration Callback.....314
5.14. Control File Generation Dynamically with Parameters and a Fileset Callback..... 315
5.15. Create a Composed Component or Subsystem..... 316
5.16. Create an IP Component with Platform Designer a System View Different from the Generated Synthesis Output Files.....318
5.17. Add Component Instances to a Static or Generated Component.....319
5.17.1. Static Components..... 320
5.17.2. Generated Components.....321
5.17.3. Design Guidelines for Adding Component Instances..... 324
5.18. Creating Platform Designer Components Revision History.....324

6. Platform Designer Command-Line Utilities.....326

6.1. Run the Platform Designer Editor with qsys-edit..... 326
6.2. Scripting IP Core Generation.....328
6.2.1. qsys-generate Command-Line Options..... 329
6.3. Display Available IP Components with ip-catalog..... 330
6.4. Create an .ipx File with ip-make-ipx.....331
6.5. Generate Simulation Scripts..... 332
6.6. Generate a Platform Designer System with qsys-script..... 333
6.7. Platform Designer Scripting Command Reference..... 335
6.7.1. System.....336
6.7.2. Subsystems.....349
6.7.3. Instances.....358
6.7.4. Connections..... 391
6.7.5. Top-level Exports..... 403
6.7.6. Validation.....416
6.7.7. Miscellaneous..... 422
6.7.8. Wire-Level Connection Commands.....435
6.8. Platform Designer Scripting Property Reference.....439
6.8.1. Connection Properties.....440
6.8.2. Design Environment Type Properties..... 441
6.8.3. Direction Properties.....442
6.8.4. Element Properties..... 443
6.8.5. Instance Properties..... 444

6.8.6. Interface Properties....445
6.8.7. Message Levels Properties....446
6.8.8. Module Properties....447
6.8.9. Parameter Properties....448
6.8.10. Parameter Status Properties....450
6.8.11. Parameter Type Properties....451
6.8.12. Port Properties....452
6.8.13. Project Properties....453
6.8.14. System Info Type Properties....454
6.8.15. Units Properties....456
6.8.16. Validation Properties....457
6.8.17. Interface Direction....458
6.8.18. File Set Kind....459
6.8.19. Access Type....460
6.8.20. Instantiation HDL File Properties....461
6.8.21. Instantiation Interface Duplicate Type....462
6.8.22. Instantiation Interface Properties....463
6.8.23. Instantiation Properties....464
6.8.25. VHDL Type....466
6.9. Platform Designer Command-Line Interface Revision History....466

7. Component Interface Tcl Reference....467

7.1. Platform Designer _hw.tcl Command Reference....467
7.1.1. Interfaces and Ports....468
7.1.2. Parameters....486
7.1.3. Display Items....497
7.1.4. Module Definition....504
7.1.5. Composition....516
7.1.6. Fileset Generation....536
7.1.7. Miscellaneous....547
7.1.8. SystemVerilog Interface Commands....553
7.1.9. Wire-Level Expression Commands....559
7.2. Platform Designer _hw.tcl Property Reference....563
7.2.1. Script Language Properties....564
7.2.2. Interface Properties....565
7.2.3. SystemVerilog Interface Properties....565
7.2.4. Instance Properties....567
7.2.5. Parameter Properties....568
7.2.6. Parameter Type Properties....570
7.2.7. Parameter Status Properties....571
7.2.8. Port Properties....572
7.2.9. Direction Properties....574
7.2.10. Display Item Properties....575
7.2.11. Display Item Kind Properties....576
7.2.12. Display Hint Properties....577
7.2.13. Module Properties....578
7.2.14. Fileset Properties....580
7.2.15. Fileset Kind Properties....581
7.2.16. Callback Properties....582
7.2.17. File Attribute Properties....583
7.2.18. File Kind Properties....584

7.2.19. File Source Properties..... 585
7.2.20. Simulator Properties..... 586
7.2.21. Port VHDL Type Properties..... 587
7.2.22. System Info Type Properties..... 588
7.2.23. Design Environment Type Properties..... 590
7.2.24. Units Properties.....591
7.2.25. Operating System Properties.....592
7.2.26. Quartus.ini Type Properties..... 593
7.3. Component Interface Tcl Reference Revision History..... 594

A. Intel Quartus Prime Standard Edition User Guides.....595

1. Creating a System with Platform Designer

The Intel® 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.

Platform Designer allows you to specify interface requirements and integrate IP components within a graphical representation of the system. The Intel 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 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.
- 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.

Related Information

- Platform Designer Command-Line Utilities on page 326
- Introduction to Intel FPGA IP Cores
- Platform Designer System Design Flow on page 12

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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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 1. IP Component Interface Types

Interface Type	Description
Memory-Mapped	Connects memory-referencing master devices with slave memory devices. Master devices can be processors and DMAs, while slave memory devices can be RAMs, ROMs, and control registers. Data transfers between master and slave may be uni-directional (read only or write only), or bi-directional (read and write).
Streaming	Connects Avalon Streaming (Avalon-ST) 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-ST 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

- Avalon Interface Specifications
- AMBA Protocol Specifications

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 Intel 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 326 describes.

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

The circled numbers in the diagram correspond with the following topics in this chapter:

1. Starting or Opening a Project in Platform Designer on page 12
2. Adding IP Components to a System on page 21
3. Connecting System Components on page 29
4. Specifying Interconnect Requirements on page 39
5. Synchronizing System Component Information on page 56
6. Generating a Platform Designer System on page 57
7. Simulating a Platform Designer System on page 65
8. Integrating a Platform Designer System with the Intel Quartus Prime Software on page 67

1.3. Starting or Opening a Project in Platform Designer

1. To start a new Platform Designer project, save the default system that appears when you open Platform Designer (**File ➤ Save**), or click **File ➤ New System**, and then save your new project.
   Platform Designer saves the new project in the Intel Quartus Prime project directory. To alternatively save your Platform Designer project in a different directory, click **File ➤ Save As**.
2. To open a recent Platform Designer project, click **File ➤ Open** to browse for the project, or locate a recent project with the **File ➤ Recent Projects** command.
3. To revert the project currently open in Platform Designer to the saved version, click the first item in the **Recent Projects** list.

Note: You can edit the directory path information in the `recent_projects.ini` file to reflect a new location for items that appear in the Recent Projects list.

1.4. 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 whenever you open a system. You can also access other panels 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.

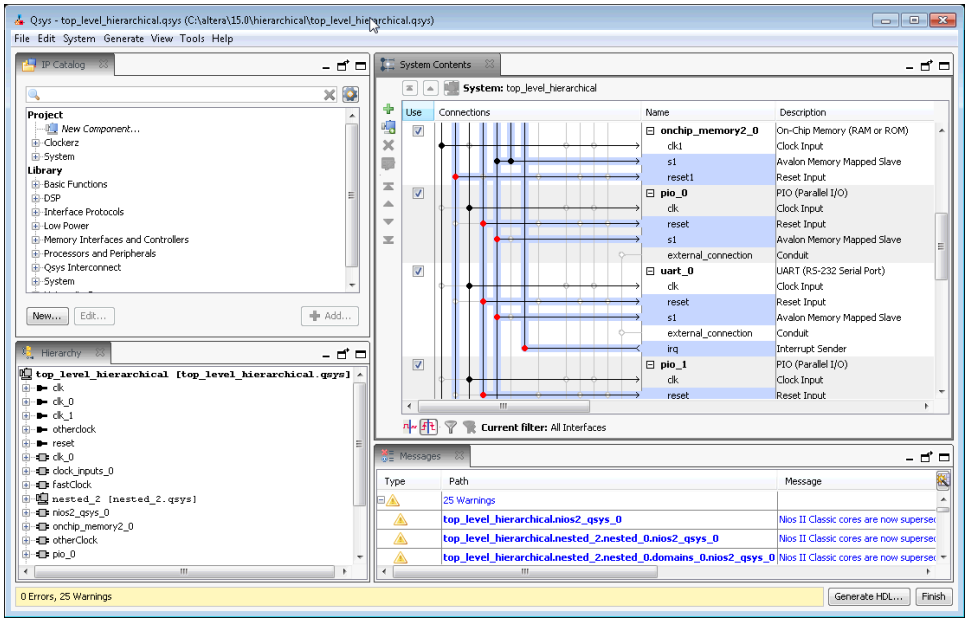
Click the View menu to interact with the elements of your system in various tabs.

The 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.

The IP Catalog and **Hierarchy** tabs display to the left of the main frame by default. The **System View**, **Address Map**, **Interconnect Requirements**, and **Device Family** tabs display in the main frame.

The **Messages** tab displays in the lower portion of Platform Designer. Double-clicking a message in the **Messages** tab changes focus to the associated element in the relevant tab to facilitate debugging. When the **Messages** tab is closed or not open in your workspace, error and warning message counts continue to display in the status bar of the Platform Designer window.

Figure 1. View a Platform Designer System



1.4.1. 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.

Expanding the System Hierarchy

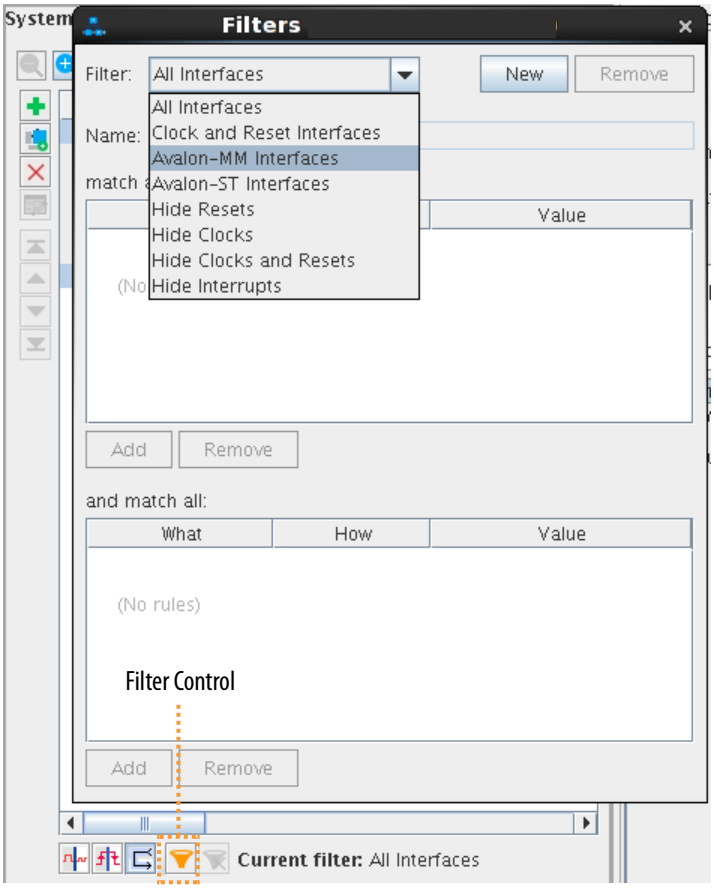
Click the **+** icon to expand any interface in the **Hierarchy** tab to view sub-components, associated elements, and signals for the interface. The **Hierarchy** tab displays a unique icon for each element type in the system. In the example below, the `ram_master` selection appears selected in both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

1.4.2. Filtering the System View

You can easily filter the display of your system in the **System View** by interface type, instance name, or other custom properties that you define. Filtering the view allows you to simplify the display and focus only on the items you want.

For example, you can click the **Filter** button to display only instances that include memory-mapped interfaces, or display only instances that connect to a particular Nios® II processor. Conversely, you can temporarily hide clock and reset interfaces to further simplify the display.

Figure 2. Filter Icon and Filters Dialog Box



Related Information

Filters Dialog Box

1.4.3. 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.

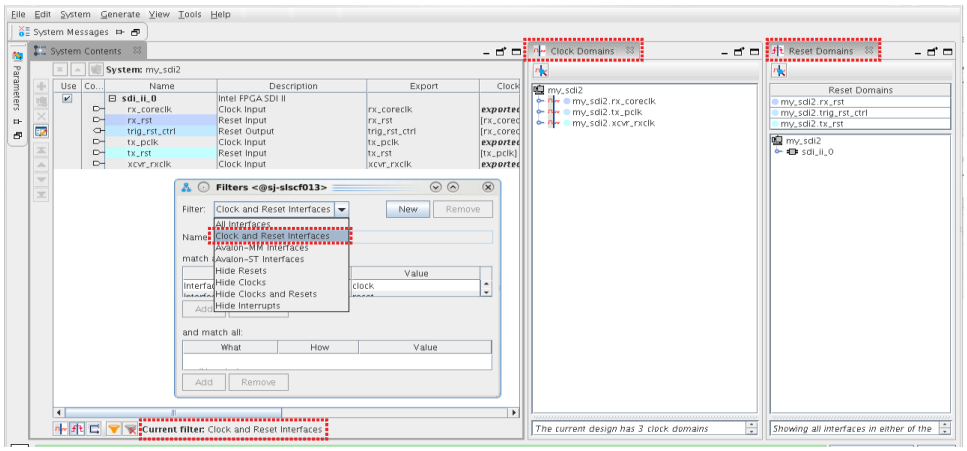
Click **View ➤ Clock Domains** or click **View ➤ Reset Domains** to display these tabs.

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 3. Platform Designer Clock and Reset Domains



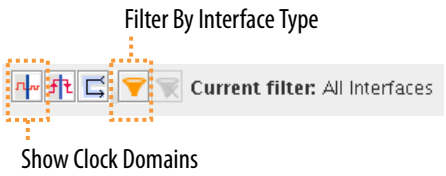
1.4.3.1. Viewing Clock Domains in a System

On the **Clock Domains** tab, you can filter the **System View** tab to display a single clock domain, or multiple clock domains. You can further filter your view with the **Filter** control. 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**:

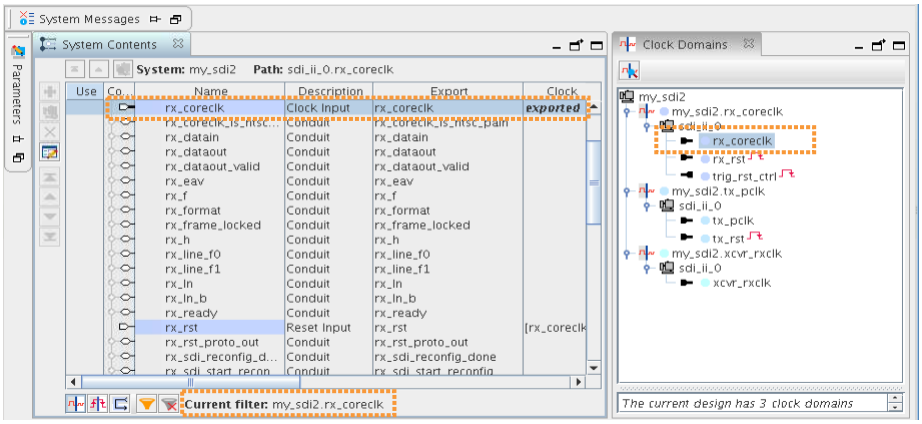
1. Click **View ➤ Clock Domains**.
2. Select any clock or reset domain in the list to view associated interfaces. The corresponding selection appears in the **System View** tab.
3. To highlight clock domains in the **System View** tab, click **Show clock domains in the system table** or at the bottom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

Figure 4. 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.

Figure 5. Clock Domains



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.

   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.

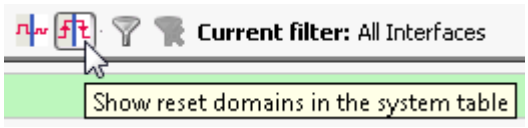
1.4.3.2. Viewing Reset Domains in a System

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 6. 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.

   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.

   Note: 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.

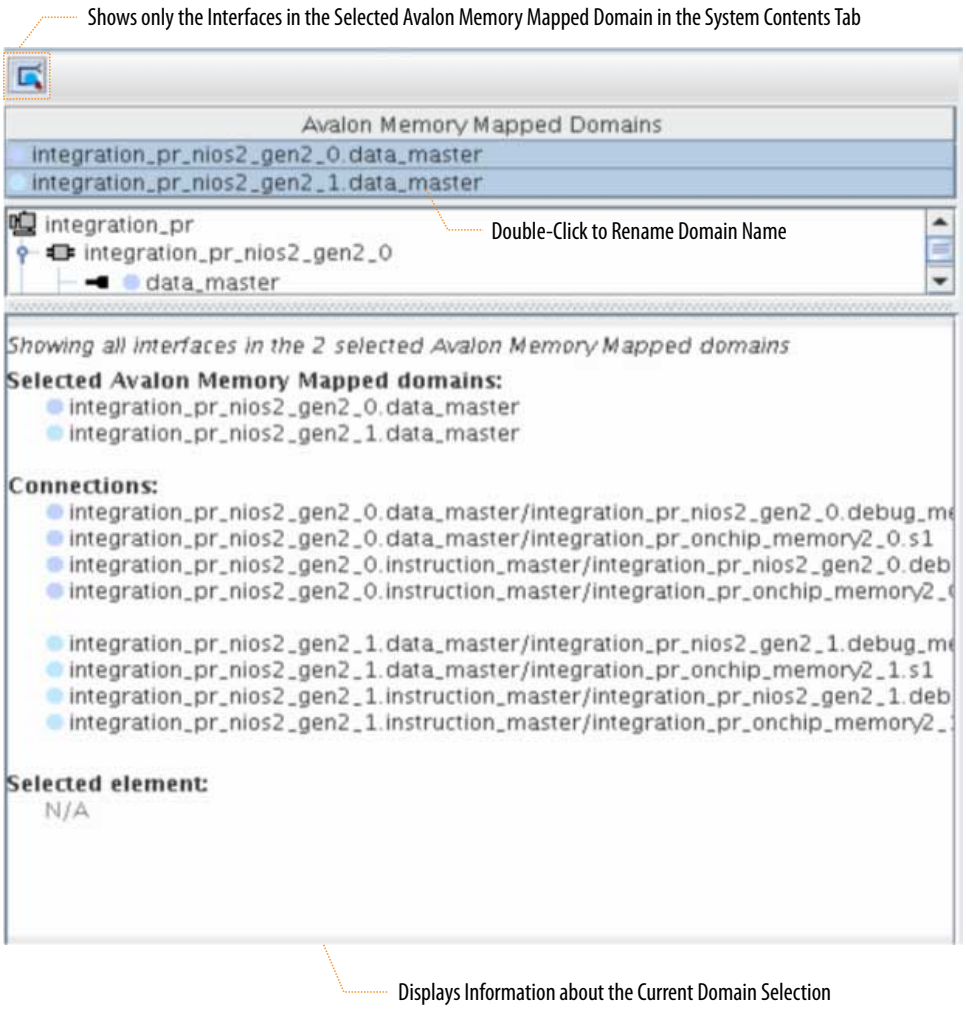
   - Multiple resets fan into a common sink.
   - Reset inputs are associated with different clock domains.
   - Reset inputs have different synchronicity.

1.4.4. Viewing Avalon Memory-Mapped Domains in a System

The **Avalon Memory Mapped Domains** tab displays a list of all the Avalon domains in the system. When you select a domain in the **Avalon Memory Mapped Domains** tab, the corresponding selection highlights in the **System View** tab.

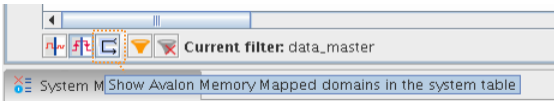
Click **View ➤ Avalon Memory Mapped Domains** to display this tab.

Figure 7. Avalon Memory Mapped Domains 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.
- 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 8. Avalon Memory Mapped Domains Control Tool

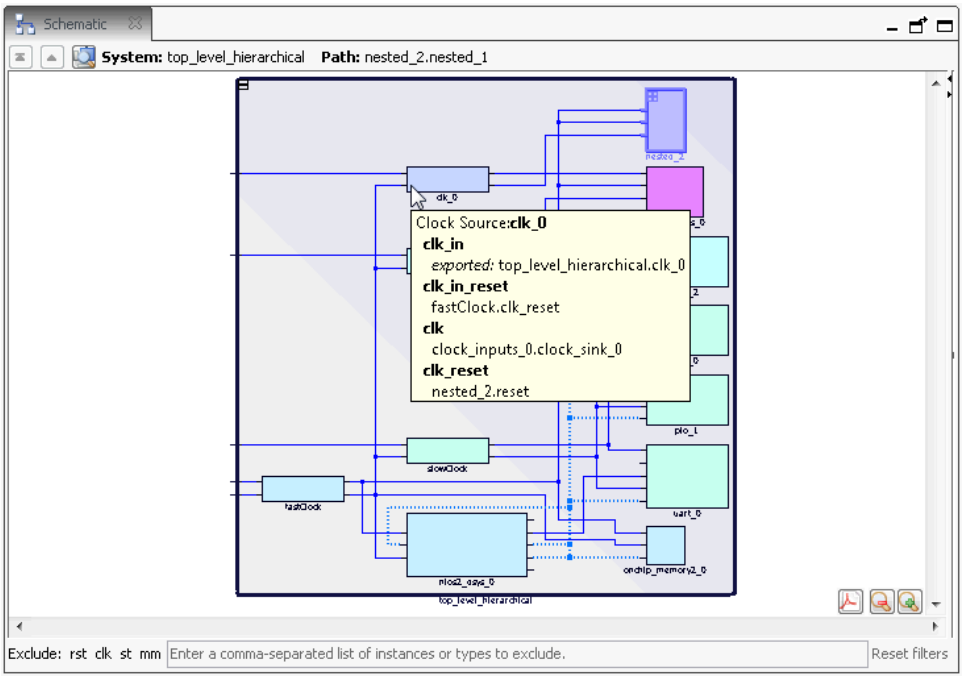


1.4.5. Viewing the System Schematic

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.

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 9. Platform Designer Schematic Tab



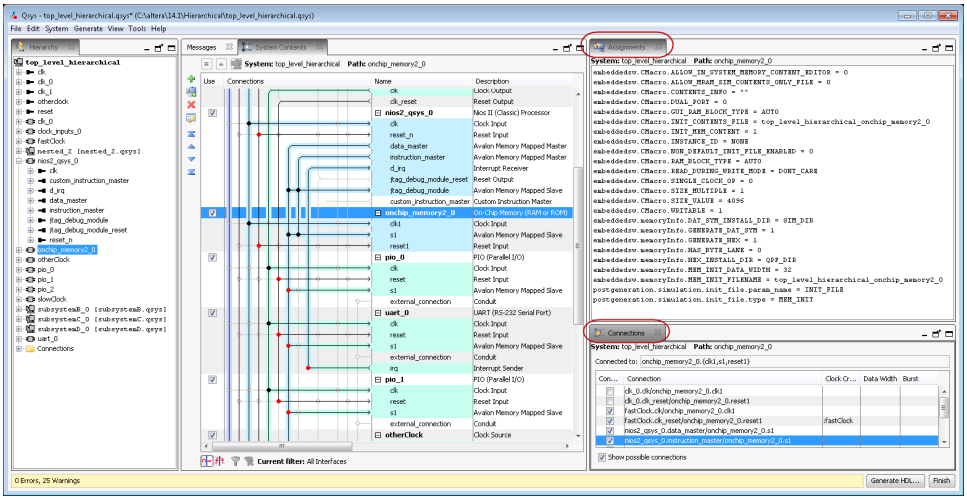
Related Information

Editing a Subsystem on page 71

1.4.6. Viewing System Assignments and Connections

On the **Assignments** tab (**View ➤ Assignments**), you can view assignments for a module or element that you select in the **System View** tab. The **Connections** tab displays a lists of connections in your Platform Designer system. On the **Connections** tab (**View ➤ Connections**), you can choose to connect or un-connect a module in your system, and then view the results in the **System View** tab.

Figure 10. Assignments and Connections tabs in Platform Designer



1.4.7. 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.

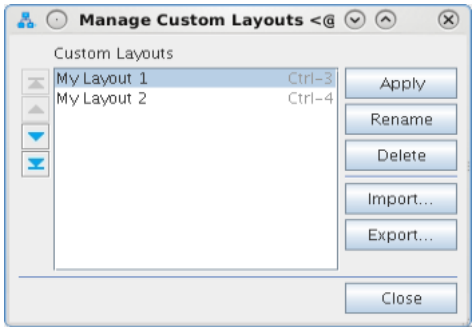
Follow these steps to customize and save the Platform Designer layout:

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.
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.

Figure 11. Manage Custom Layouts



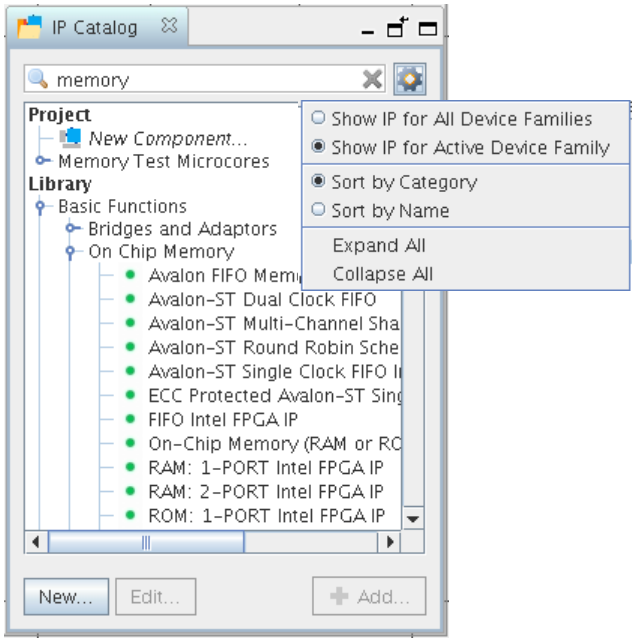
1.5. Adding IP Components to a System

You can quickly add Intel FPGA IP components to a system from the IP Catalog in Platform Designer. The IP Catalog launches a parameter editor that allows you to specify options and add the component to your system. 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.

Follow these steps to locate, parameterize, and instantiate an IP component in a Platform Designer system:

1. To locate a component by name, type some or all of the component's name in the IP Catalog search box. For example, type `memory` to locate memory-mapped IP components. You can also find components by category.

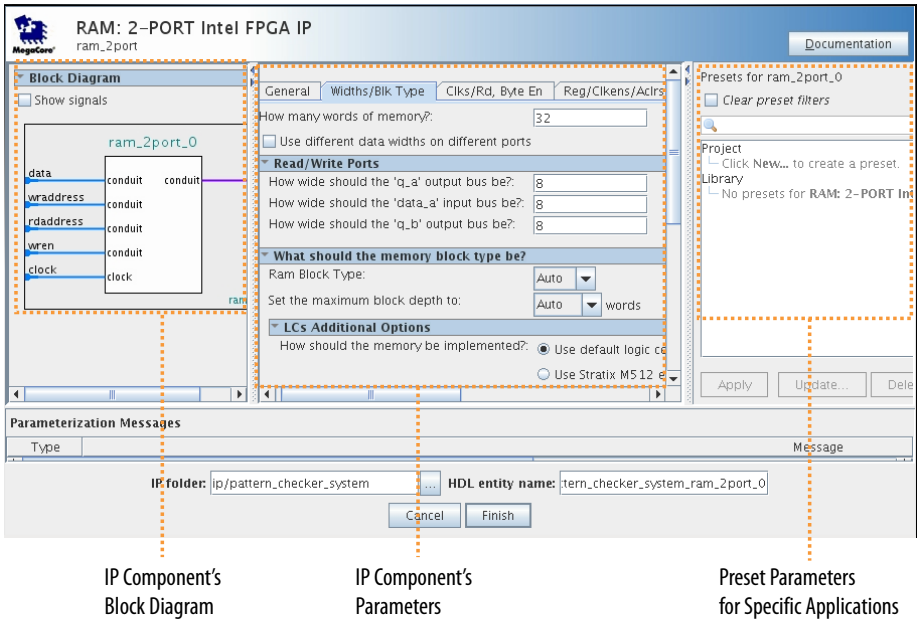
Figure 12. Platform Designer IP Catalog



2. Double-click any component to launch the component's parameter editor and specify options for the component.

   For some IP components, you can select and **Apply** a pre-defined set of parameters values for specific applications from the **Presets** list.

Figure 13. Parameter Editor



3. To complete customization of the IP component, click **Finish**. The IP component appears in the **System View** tab.

1.5.1. Modifying IP Parameters

The **Parameters** tab allows you to view and edit the current parameter settings for IP components in your system.

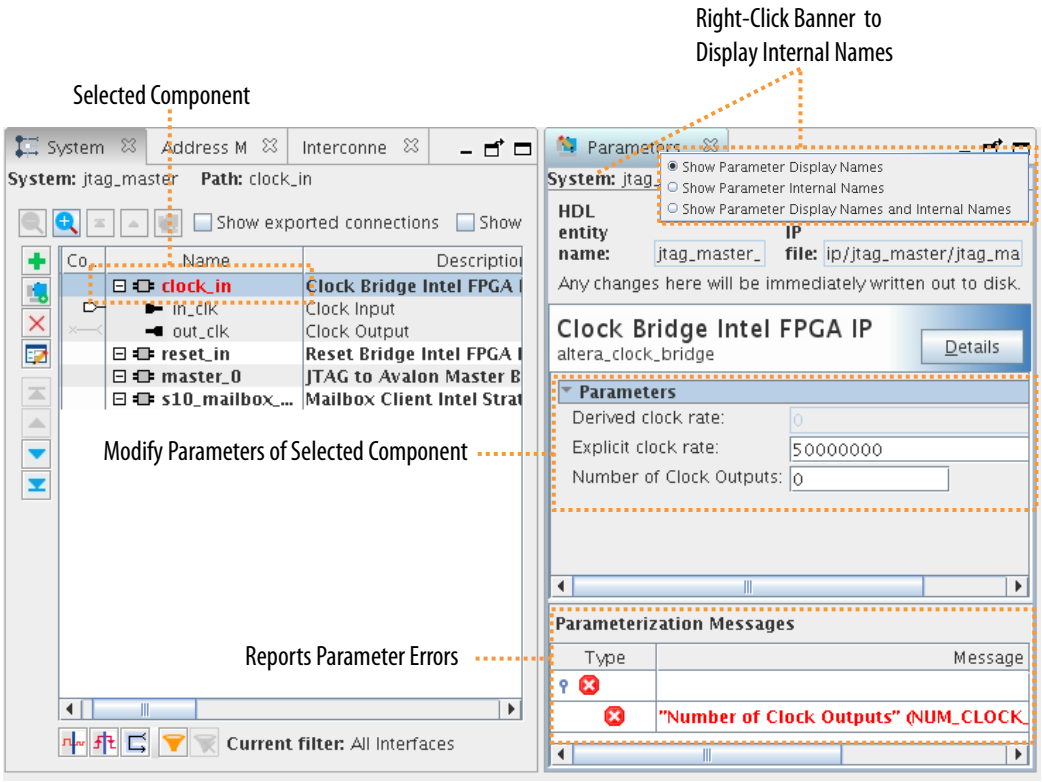
To display a components parameters on the **Parameters** tab:

1. click **View ➤ Parameters**.
2. Select the component in the **System View** or **Hierarchy** tabs..

The **Parameters** tab provides the following functionality:

- **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 allows you to enable display of internal names. Below the hierarchical path, the parameter editor shows 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.
- **Read/Write Waveforms**—displays the interface timing and the corresponding read and write waveforms.
- **Details**—displays links to detailed information about the component.
- **Parameterization Messages**—displays parameter warning and error messages about the IP component.

Figure 14. 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.

1.5.1.1. 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.

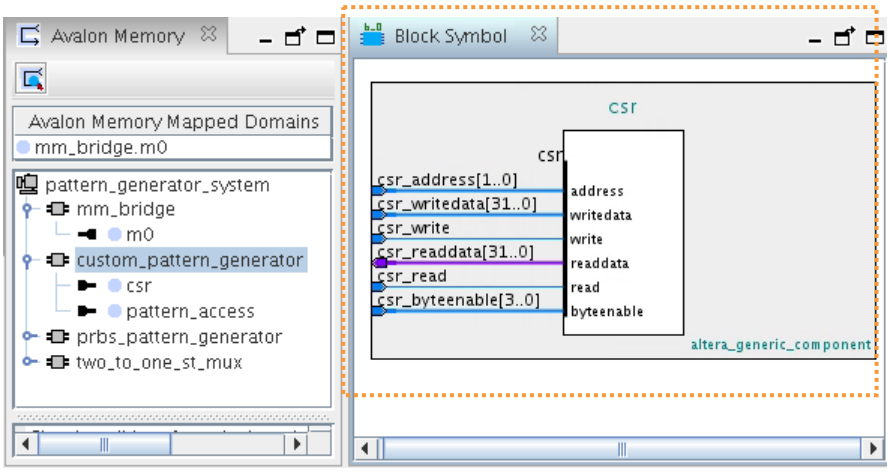
To view a component's details:

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.

1.5.1.2. 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.

Figure 15. Block Symbol Tab

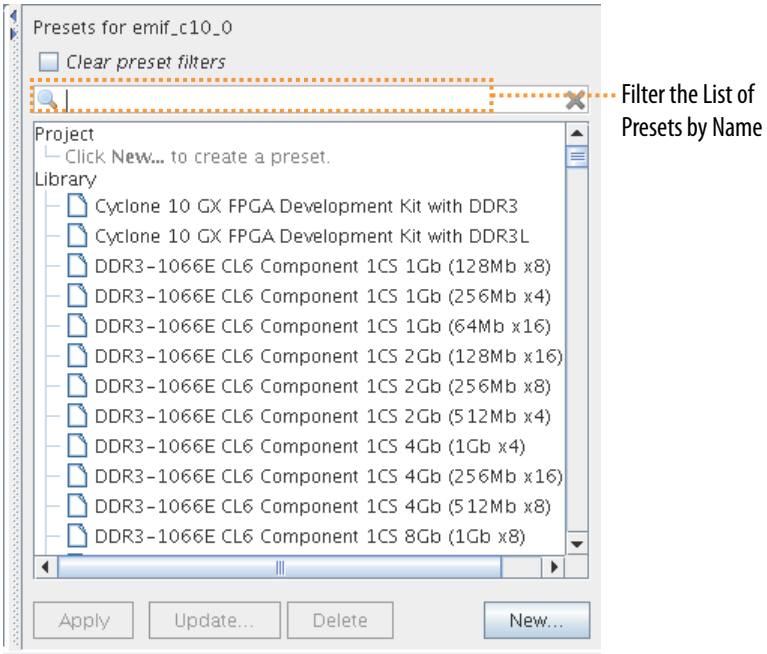


The **Block Symbol** tab appears by default in the parameter editor when you add a component to your system. When the **Block Symbol** tab is open in your workspace, it reflects changes that you make in other tabs.

1.5.2. Applying Preset Parameters for Specific Applications

The **Preset** tab displays the names of any available preset settings for an IP component. The preset preserves a collection of parameter setting that may be appropriate for a specific protocol or application. Not all IP components include preset parameters. Double-click the preset parameter name to apply the preset parameter values to a component you are defining.

Figure 16. Selecting Preset Parameters



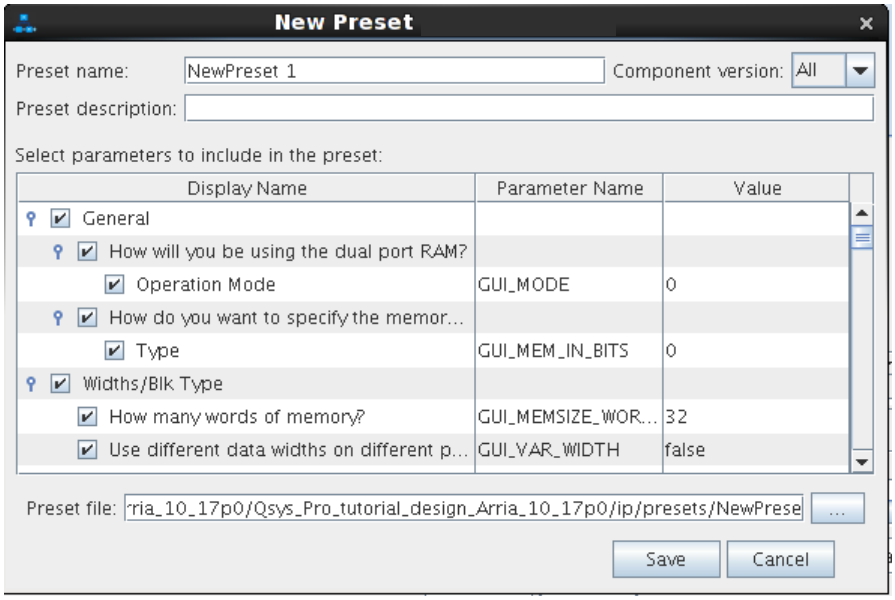
1.5.2.1. Creating IP Custom Preset Parameters Settings

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

Follow these steps to save custom preset parameter settings:

1. In IP Catalog, double-click any component to launch the parameter editor.
2. To search for a specific preset for the 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. Under **Select parameters to include in the preset**, enable or disable the parameters you want to include in the preset.
5. Specify the path for the **Preset file** that preserves the collection of parameter settings.

Figure 17. Create New Preset



If the file location that you specify is not already in the IP search path, Platform Designer adds the location of the new preset file to the IP search path.

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

1.5.3. 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-MM 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 27 describes.

Related Information

Find Intel FPGA and Partner IP

1.5.3.1. IP Search Path Recursive Search

The Intel 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. You can use ** characters to halt a recursive search at any directory that contains a `_hw.tcl` or `.ipx` file.

In the following list of search locations, ** indicates a recursive descent.

Table 2. IP Search Locations

Location	Description
`PROJECT_DIR/*`	Finds IP components and index files in the Intel Quartus Prime project directory.
`PROJECT_DIR/ip/**/*`	Finds IP components and index files in any subdirectory of the `/ip` subdirectory of the Intel Quartus Prime project directory.

1.5.3.1.1. IP Search Path Precedence

If the Intel 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`.

1.5.3.1.2. IP Component Description Files

The Intel 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.

1.5.3.2. 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 specify a search path in the `user_components.ipx` file in either in Platform Designer (**Tools ➤ Options**) or the Intel Quartus Prime software (**Tools ➤ Options ➤ IP Catalog Search Locations**). This method of discovering IP components allows you to add a locations dependent of the default search path. The `user_components.ipx` file directs Platform Designer to the location of an IP component or directory to search.

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 331

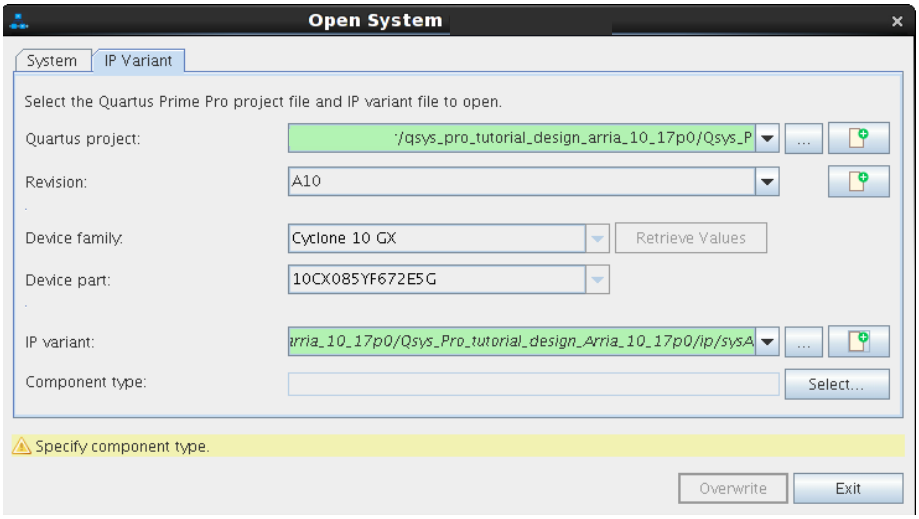
1.5.4. Creating or Opening an IP Core Variant

In addition to creating a system, Platform Designer allows you to define a stand-alone IP core variant that you can add to your Intel 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.

Figure 18. Platform Designer IP Variant Tab



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 Intel 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 Intel 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 58 for generation options.

1.6. Connecting System Components

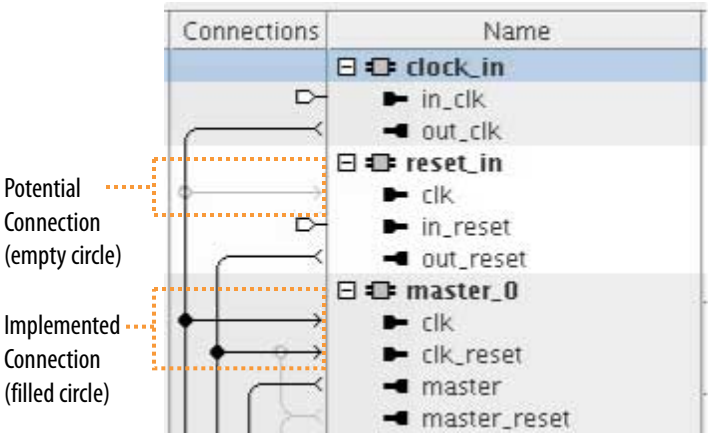
You must appropriately connect the components in your Platform Designer system. The **System View** and **Connections** tabs allow you to connect and configure IP components quickly. Platform Designer supports connections between interfaces of compatible types and opposite directions.

For example, you can connect a memory-mapped master interface to a slave 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.

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.

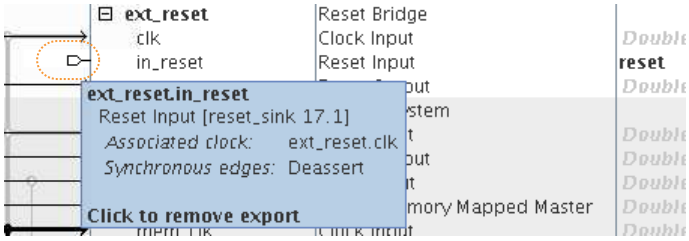
Figure 19. Potential and Implemented Connections in System View



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 20. 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.

1.6.1. 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 masters, with each master having its own address space. You can share slaves between masters, and masters can map slaves to different addresses. For example, one master can interact with slave `0` at base address `0000_0000_0000`, and another master can see the same slave at base address `c000_000_000`.

Intel Quartus Prime debugging tools provide access to the state of an addressable system via the Avalon-MM interconnect. These tools are also 64-bit compatible, and process within a 64-bit address space, including a JTAG to Avalon master bridge.

Platform Designer supports auto base address assignment for Avalon-MM components. In the **Address Map** tab, click **Auto Assign Base Address**.

Related Information

- Address Map Tab Help
- Address Span Extender on page 232
- auto_assign_base_addresses on page 423

1.6.1.1. Support for Avalon-MM Non-Power of Two Data Widths

Platform Designer requires that you connect all multi point Avalon-MM connections to interfaces with data widths that are equal to powers of two.

Platform Designer issues a validation error if an Avalon-MM master or slave interface on a multi point connection is parameterized with a non-power of two data width.

Note: Avalon-MM point-to-point connections between an Avalon-MM master and an Avalon-MM slave are an exception, you can set their data widths to a non-power of two.

1.6.2. Connecting Masters and Slaves

Specify connections between master and slave components in the **Address Map** tab. This tab allows you to specify the address range that each memory-mapped master uses to connect to a slave in a Platform Designer system.

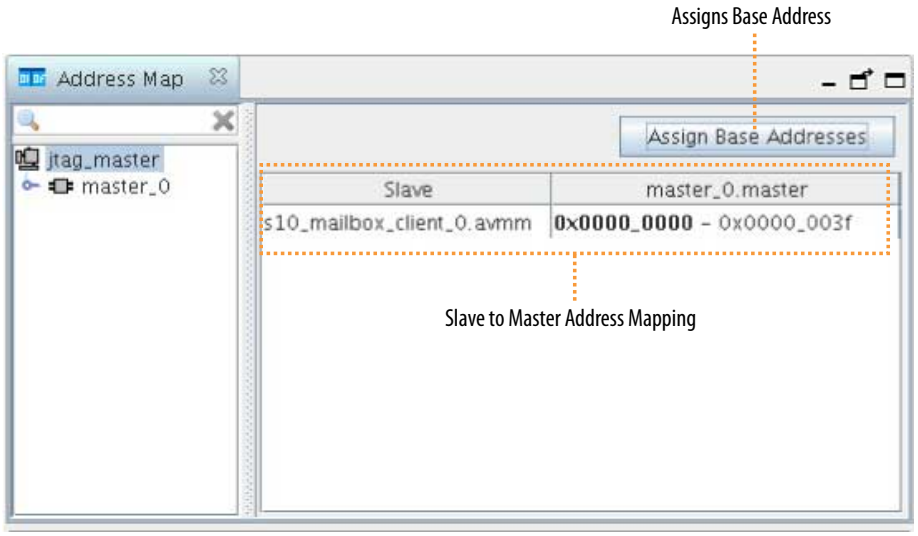
The **Address Map** tab shows the slaves on the left, the masters across the top, and the address span of the connection in each cell. If there is no connection between a master and a slave, the table cell is empty. In this case, use the **Address Map** tab to view the individual memory addresses for each connected master.

Platform Designer enables you to design a system where two masters access the same slave 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 master and slave IP components:

1. In Platform Designer, click the **Address Map** tab.
2. Locate the table cell that represents the connection between the master and slave component pair.
3. Either type in a base address, or update the current base address in the cell. The base address of a slave component must be a multiple of the address span of the component. This restriction is a requirement of the Platform Designer interconnect, which provides an efficient address decoding logic, which in turn allows Platform Designer to achieve the best possible f_{MAX}.

Figure 21. Address Map Tab for Connection Masters and Slaves



Related Information

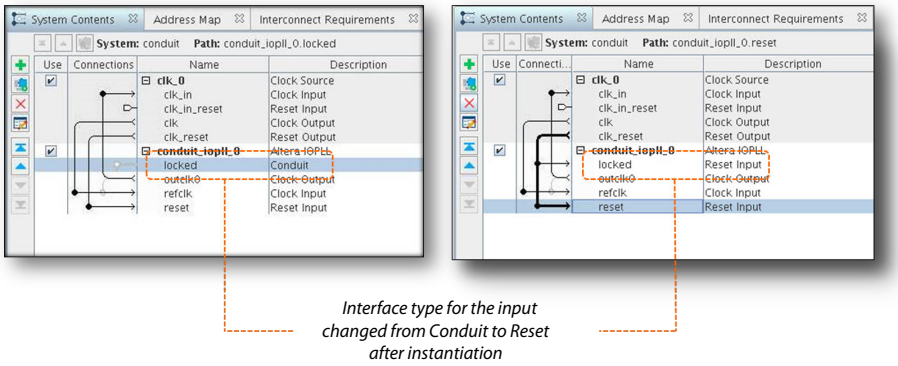
- Address Map Tab Help
- Platform Designer 64-Bit Addressing Support on page 31
- auto_assign_base_addresses on page 423

1.6.3. Changing a Conduit to a Reset

1. In the IP Catalog search box, locate **IOPLL Intel FPGA IP** and double-click to add the component to your system.
2. In the **System View** tab, select the **PLL** component.
3. Click **View ➤ Component Instantiation** and open the **Component Instantiation** tab for the selected component.
4. In the **Signals & Interfaces** tab, select the **locked** conduit interface.
5. Change the **Type** from **Conduit** to **Reset Input**, and the **Synchronous edges** from **Deassert** to **None**.
6. Select the `locked [1]` signal below the **locked** interface.
7. Change the **Signal Type** from **export** to **reset_n**. Change the **Direction** from **output** to **input**.
8. Click **Apply**.

The conduit interface changes to reset for the instantiated PLL component.

Figure 22. Changing Conduit to a Reset



1.6.4. 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.
- The wire-level expression 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 so that they are available for use in wire-level expressions at the same hierarchy level.
- 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 for your system and resolve any errors, you next generate the system to create the Verilog files. When you apply the wire-level connections in the Platform Designer GUI, or with the `qsys-script` utility, the wire-level expression is inserts in 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 Verilog wrapper file that generates for the specified IP component.

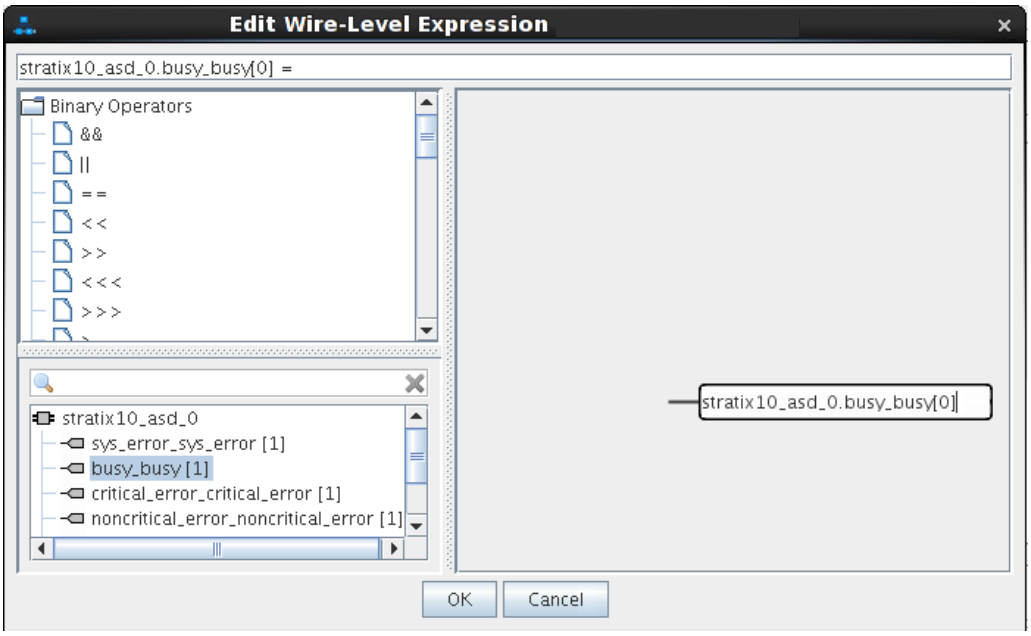
1.6.4.1. Editing Wire-Level Expressions

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:

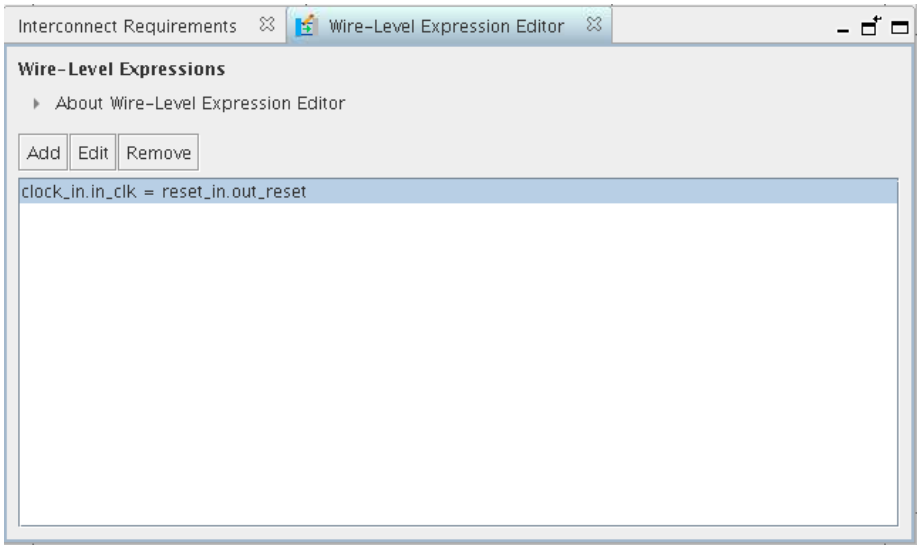
1. To specify a new wire-level expression, right-click an input port in the **Hierarchy** tab and click **Add Wire-Level Expression**. The **Edit Wire-Level Expression** dialog box appears.

Figure 23. Edit Wire-Level Expression Dialog Box



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 36 for a list of legal operators.
3. Click in 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.
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 on 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 24. Wire-Level Expression Editor



1.6.4.2. 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 3. 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 4. Operators (Bitwise)

Operator	Description
~	Negation
&	AND
|	OR
~&	NAND

continued...

Operator	Description
~|	NOR
^	XOR
~^	XNOR

Table 5. Operators (Logical)

Operator	Description
?	Conditional
!	Negation
&&	AND
||	OR

Table 6. 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 7. Operators (Mathematical)

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Table 8. Operators (Other)

Operator	Description
{integer {*x*}}	Replication of *x*
{*x*, *y*, ...}	Concatenation

1.6.4.3. 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**.

1.6.4.4. 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.

1.6.5. Previewing the System Interconnect

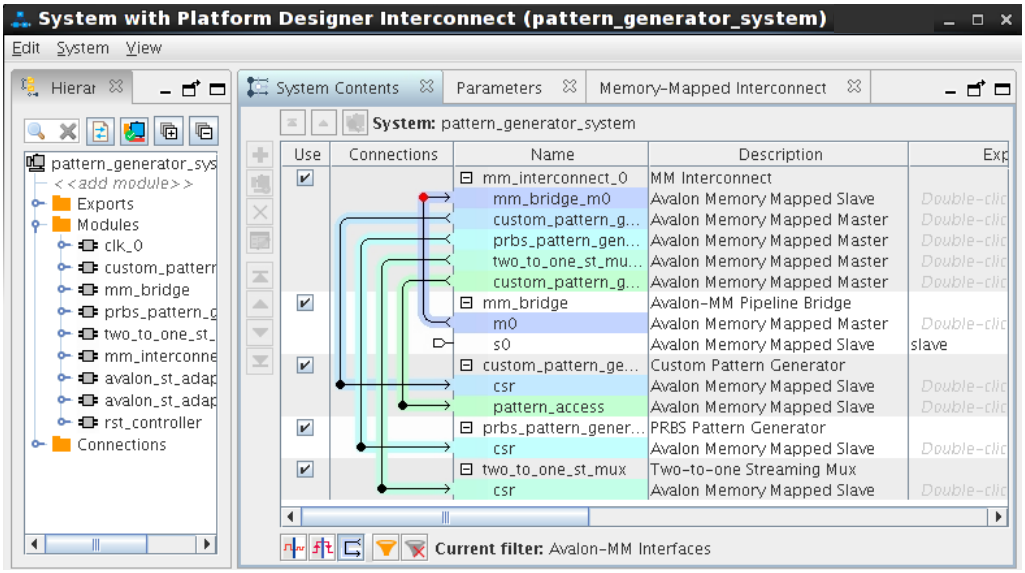
You can review a graphical representation of the Platform Designer interconnect before you generate the system. The System with Platform Designer Interconnect window shows how Platform Designer converts connections between interfaces to interconnect logic during system generation.

To open the System with Platform Designer Interconnect window, click **System ➤ Show System With Platform Designer Interconnect**.

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.
- **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.

Figure 25. System with Platform Designer Interconnect window



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. 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 master or slave 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.

Note: You must select more than one module to highlight a path.

1.7. Specifying Interconnect Requirements

The **Interconnect Requirements** tab allows you to apply system-wide (`$system`) or interface-specific interconnect requirements for IP components in your system.

Available options in the **Setting** column vary, depending on the **Identifier** column value. Click the drop-down menu to select the settings, and to assign the corresponding values to the settings.

Follow these steps to specify system or interface interconnect requirements.

1. To create a new **Identifier** to assign an interconnect requirement, click **Add**. A **new_target** row appears for edit.
2. Click the **new_target** cell and select **$system** to define a system-wide requirement, or select any interface name to specify interconnect requirements for the interface.
3. In the same row, click the **new_requirement** cell, select any of the available requirements, as Interconnect Requirements on page 40 describes.
4. In the same row, Click the **new_requirement_value** cell and specify the requirement value.

For more information about HPS, refer to the *Cyclone® V Device Handbook* in volume 3 of the *Hard Processor System Technical Reference Manual*.

Related Information

- Platform Designer Interconnect on page 128
- Reset Interfaces on page 171

1.7.1. Interconnect Requirements

Table 9. System-Wide Interconnect Requirements

Option	Description
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 pipeline stages, where 0 means that the interconnect has a combinational datapath. This setting is specific for each Platform Designer system or subsystem.
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 slave for undefined memory region accesses during system generation.
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.
Burst Adapter Implementation	Allows you to choose the converter type that Platform Designer applies to each burst.

continued...

Option	Description
	Generic converter (slower, lower area): Default. 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 masters or slaves 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 188.

Table 10. Specifying Interface Interconnect Requirements

You can apply the following interconnect requirements when you select a component interface as the **Identifier** in the **Interconnect Requirements** tab, in the **All Requirements** table.

Option	Value	Description
Security	• Non-secure • Secure • Secure ranges • TrustZone*-aware	After you establish connections between the masters and slaves, allows you to set the security options, as needed, for each master and slave in your system.
Secure address ranges	Accepts valid address range.	Allows you to type in any valid address range.

Related Information

- Interconnect Pipelining on page 184
- Error Correction Coding (ECC) in Platform Designer Interconnect on page 188

1.8. Defining Instance Parameters

You can implement instance parameters to test the functionality of your Platform Designer system when using another Platform Designer system as a sub-component of a larger system. A higher-level Platform Designer system can assign values to instance parameters, and then test those values in the lower-level system.

You can use the **Instance Parameters** tab to define how the specified values for the instance parameters affect the sub-components in the Platform Designer system. You define an **Instance Script** that creates queries for the instance parameters, and sets the values of the parameters for the lower-level system components.

When you click **Preview Instance**, Platform Designer creates a preview of the current Platform Designer system with the specified parameters and instance script and opens the parameter editor. This command allows you to see how an instance of a system appears when you use it in another system. The preview instance does not affect your saved system.

To use instance parameters, the IP components or subsystems in your Platform Designer system must have parameters that can be set when they are instantiated in a higher-level system.

If you create hierarchical Platform Designer systems, each Platform Designer system in the hierarchy can include instance parameters to pass parameter values through multiple levels of hierarchy.

1.8.1. Creating an Instance Parameter Script in Platform Designer

The first command in an instance parameter script must specify the Tcl command version. The version command ensures the Tcl commands behave identically in future versions of the tool. Use the following command to specify the version of the Tcl commands, where <*version*> is the Intel Quartus Prime software version number, such as 14.1:

```
package require -exact qsys <version>
```

To use Tcl commands that work with instance parameters in the instance script, you must specify the commands within a Tcl composition callback. In the instance script, you specify the name for the composition callback with the following command:

```
set_module_property COMPOSITION_CALLBACK <name of callback procedure>
```

Specify the appropriate Tcl commands inside the Tcl procedure with the following syntax:

```
proc <name of procedure defined in previous command> {}
{#Tcl commands to query and set parameters go here}
```

Example 3. Instance Parameter Script Example

In this example, an instance script uses the `pio_width` parameter to set the `width` parameter of a parallel I/O (PIO) component. The script combines the `get_parameter_value` and `set_instance_parameter_value` commands using brackets.

```
# Request a specific version of the scripting API
package require -exact qsys 13.1

# Set the name of the procedure to manipulate parameters:
set_module_property COMPOSITION_CALLBACK compose

proc compose {} {

# Get the pio_width parameter value from this Platform Designer system and
# pass the value to the width parameter of the pio_0 instance

set_instance_parameter_value pio_0 width \
[get_parameter_value pio_width]
}
```

Related Information

Component Interface Tcl Reference on page 467

1.8.2. Platform Designer Instance Parameter Script Tcl Commands

You can use standard Tcl commands to manipulate parameters in the script, such as the `set` command to create variables, or the `expr` command for mathematical manipulation of the parameter values. Instance scripts also use Tcl commands to query the parameters of a Platform Designer system, or to set the values of the parameters of the sub-IP-components instantiated in the system.

1.8.2.1. 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 44
- set_instance_parameter_value on page 388

1.8.2.2. 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 378
- get_instance_parameter_value on page 43
- set_instance_parameter_value on page 388

1.8.2.3. get_parameter_value

Description

Returns the current value of a parameter defined previously with the `add_parameter` command.

Usage

`get_parameter_value` <*parameter*>

Returns

The value of the parameter.

Arguments

parameter The name of the parameter whose value is being retrieved.

Example

```
get_parameter_value fifo_width
```

1.8.2.4. get_parameters

Description

Returns the names of all the parameters in the component.

Usage

```
get_parameters
```

Returns

A list of parameter names.

Arguments

No arguments.

Example

```
get_parameters
```

1.8.2.5. 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 Intel 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!"
```

1.8.2.6. 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 may not be set using this command.

Usage

`set_instance_parameter_value` <*instance*> <*parameter*> <*value*>

Returns

No return value.

Arguments

instance The name of the child instance.

parameter The name of the parameter.

value The new parameter value.

Example

```
set_instance_parameter_value uart_0 baudRate 9600
```

1.8.2.7. 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 341
- get_module_property on page 342
- Module Properties on page 447

1.9. Implementing Performance Monitoring

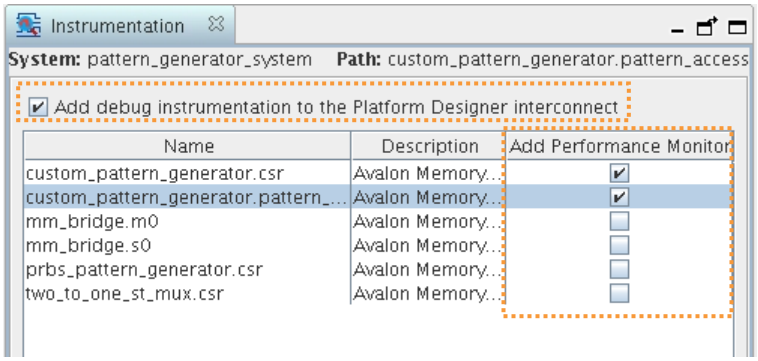
You can set up real-time performance monitoring for your Platform Designer system using throughput metrics such as read and write transfers.

Platform Designer supports performance monitoring for only Avalon-MM interfaces. In your Platform Designer system, you can monitor the performance of no less than three, and no greater than 15 Avalon-MM interface components at one time.

Follow these steps to implement performance monitoring:

1. Open a system in Platform Designer.
2. Click **View ➤ Instrumentation**.
3. To enable performance monitoring, turn on **Add debug instrumentation to the Platform Designer Interconnect** option. Enabling this option allows the system to interact with the Bus Analyzer Toolkit, accessible from the Intel Quartus Prime Tools menu.
4. For any interconnect, enable or disable the **Add Performance Monitor** option.

Figure 26. Enabling Performance Monitoring



Note: For more information about the Bus Analyzer Toolkit and the Platform Designer **Instrumentation** tab, refer to the Bus Analyzer Toolkit page.

1.10. Configuring Platform Designer System Security

You can specify Platform Designer system and interconnect security settings on the **Interconnect Requirements** 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, as Table 12 on page 53 describes.

The AXI AxPROT protection signal specifies a secure or non-secure transaction. When an AXI master sends a command, the AxPROT signal specifies whether the command is secure or non-secure. When an AXI slave receives a command, the AxPROT 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 masters and slaves can be TrustZone-aware. All other master and slave interfaces, such as Avalon-MM interfaces, are non-TrustZone-aware.

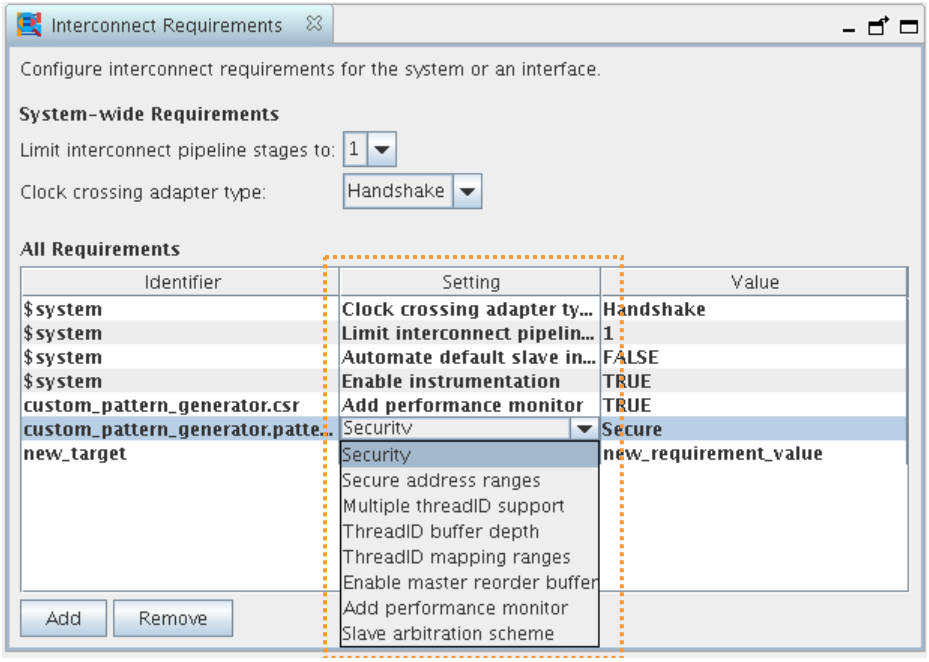
The Avalon specification does not include a protection signal. Consequently, when an Avalon master sends a command, there is no embedded security and Platform Designer recognizes the command as non-secure. Similarly, when an Avalon slave receives a command, the slave always accepts the command and responds.

Follow these steps to set compile-time security support for non-TrustZone-aware components:

1. To begin creating a secure system, add masters and slaves to your system, as Adding IP Components to a System on page 21 describes.
2. Make connections between the masters and slaves in your system, as Connecting Masters and Slaves on page 32 describes.
3. Click **View ➤ Interconnect Requirements**. The **Interconnect Requirements** tab allows you to specify system-wide and interconnect-specific requirements.
4. To specify security requirements for an interconnect, click the **Add** button.
5. In the **Identifier** column, select the interconnect in the **new_target** cell.

6. In the **Setting** column, select **Security**.
7. In the **Value** column, select the appropriate **Secure**, **Non-Secure**, **Secure Ranges**, or **TrustZone-aware** security for the interface. Refer to System Security Options on page 52 for details of each option.

Figure 27. Security Settings in Interconnect Requirements Tab



8. After setting compile-time security options for non-TrustZone-aware master and slave interfaces, you must identify those masters that require a default slave before generation, as Specifying a Default Slave on page 52.

Related Information

- Platform Designer Interconnect on page 128
- Platform Designer System Design Components on page 214

1.10.1. System Security Options

Table 11. Security Options

Option	Description
Secure	Master sends only secure transactions, and the slave receives only secure transactions. Platform Designer treats transactions from a secure master as secure. Platform Designer blocks non-secure transactions to a secure slave and routes to the default slave.
Non-Secure	The master sends only non-secure transactions, and the slave receives any transaction, secure or non-secure. Platform Designer treats transactions from a non-secure master as non-secure. Platform Designer allows all transactions, regardless of security status, to reach a non-secure slave.
Secure Ranges	Applies to only the slave interface. Allows you to specify secure memory regions for a slave. Platform Designer blocks non-secure transactions to secure regions and routes to the default slave. The specified address ranges within the slave'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 masters have signals that control the security status of their transactions. TrustZone-aware slaves 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 masters are TrustZone-aware. • You can set AXI, AMBA 3 AXI, and AMBA 3 AXI-Lite slaves as TrustZone-aware, secure, non-secure, or secure range ranges. • You can set non-AXI master interfaces as secure or non-secure. • You can set non-AXI slave interfaces as secure, non-secure, or secure address ranges.

1.10.2. Specifying a Default Slave

If a master issues "per-access" or "not allowed" transactions, your design must contain a default slave. Per-access refers to the ability of a TrustZone-aware master 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 master that initiates transactions to a secure slave resulting in unsuccessful transfers, within the same hierarchy.

A transaction that violates security is rerouted to the default slave and subsequently responds to the master with an error. The following rules apply to specifying a default slave:

- You can designate any slave as the default slave.
- You can share a default slave between multiple masters.
- Have one default slave for each interconnect domain.
- An interconnect domain is a group of connected memory-mapped masters and slaves that share the same interconnect. The `altera_error_response_slave` component includes the required TrustZone features.

To designate a slave interface as the default slave for non TrustZone-aware interfaces, follow these steps:

1. Specify interconnect security settings, as Configuring Platform Designer System Security on page 50 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 slave interface. A master can have only one default slave.

Table 12. Secure and Non-Secure Access Between Master, Slave, and Memory Components

Transaction Type	TrustZone-aware Master	Non-TrustZone-aware Master Secure	Non-TrustZone-aware Master Non-Secure
TrustZone-aware slave/memory	OK	OK	OK
Non-TrustZone-aware slave (secure)	Per-access	OK	Not allowed
Non-TrustZone-aware slave (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 on page 237
- Designating a Default Slave on page 242

1.10.3. Accessing Undefined Memory Regions

Access to an undefined memory region occurs when a transaction from a master targets a memory region unspecified in the slave memory map. To ensure predictable response behavior when this condition occurs, you must specify a default slave, as Specifying a Default Slave on page 52 describes.

You can designate any memory-mapped slave as a default slave. Have only one default slave for each interconnect domain in your system. Platform Designer then routes undefined memory region accesses to the default slave, which terminates the transaction with an error response.

Note: If you do not specify the default slave, Platform Designer automatically assigns the slave at the lowest address within the memory map for the master that issues the request as the default slave.

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 slaves, but are not mapped.
- Accesses by a master to a region that does not belong to any slaves that is mapped to the master.
- When a non-secured transaction is accessing a secured slave. This applies to only slaves that are secured at compilation time.
- When a read-only slave is accessed with a write command, or a write-only slave is accessed with a read command.

1.11. Upgrading Outdated IP Components

When you open a Platform Designer system that contains outdated IP components, Platform Designer automatically attempts to upgrade the IP components if it cannot locate the requested version.

Most Platform Designer IP components support automatic upgrade.

Platform Designer allows you to include a path to older IP components in the IP Search Path, and then use those components even if upgraded versions are available. However, older versions of IP components may not work in newer version of Platform Designer.

If a Platform Designer system includes IP components outside of the project directory or the directory of the `.qsys` file, you must add the location of these components to the Platform Designer IP Search Path (**Tools ➤ Options**).

To upgrade IP cores:

1. With the Platform Designer system open, click **System ➤ Upgrade IP Cores**. Only IP Components that are associated with the open Platform Designer system, and that do not support automatic upgrade appear in **Upgrade IP Cores** dialog box.
2. In the **Upgrade IP Cores** dialog box, select one or multiple IP components, and then click **Upgrade**.
   A green check mark appears for the IP components that Platform Designer successfully upgrades.
3. Generate the Platform Designer system.

Alternatively, you can upgrade IP components with the following command:

```
qsys-generate --upgrade-ip-cores <qsys_file>
```

The *<qsys_file>* variable accepts a path to the `.qsys` file. You do not need to run this command in the same directory as the `.qsys` file. Platform Designer reports the start and finish of the command-line upgrade, but does not name the particular IP components upgraded.

For device migration information, refer to *Introduction to Intel FPGA IP*.

Related Information

Introduction to Intel FPGA IP Cores

1.11.1. Troubleshooting IP or Platform Designer System Upgrade

The **Upgrade IP Components** dialog box reports the version and status of each 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:

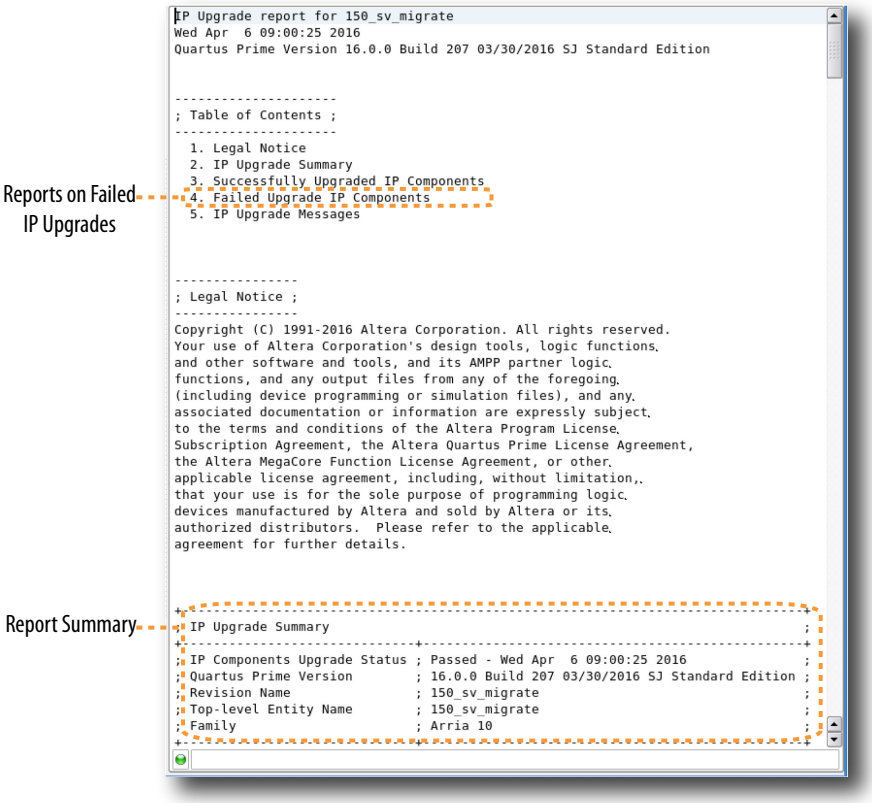
Table 13. IP Upgrade Error Information

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_report` 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 fails 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 28. IP Upgrade Report



1.12. Synchronizing System Component Information

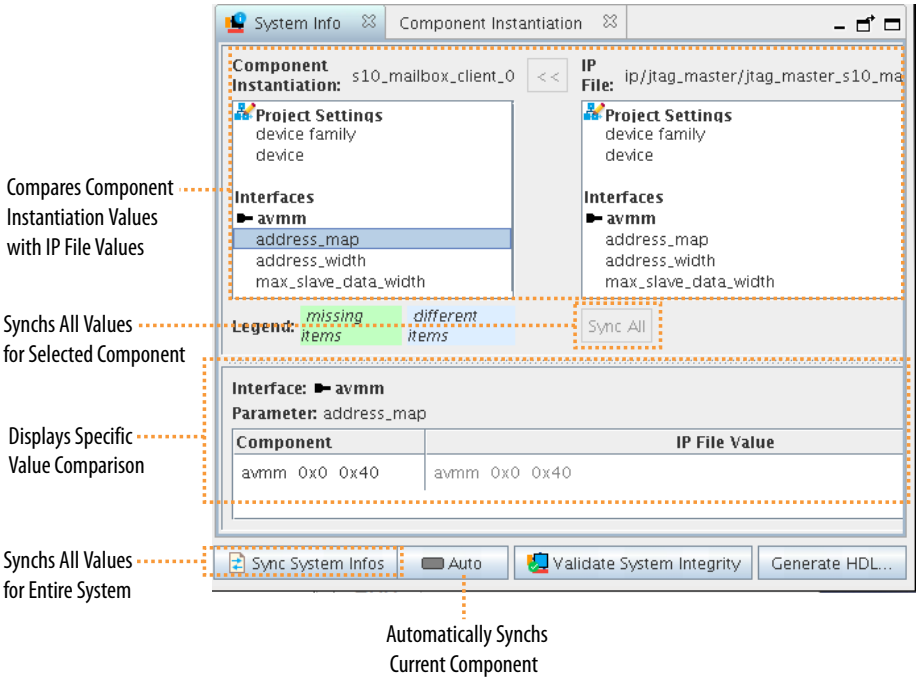
When a component instantiation values do match the component's corresponding `.ip` file, Platform Designer reports these mismatches as Component Instantiation Warnings in the **System Messages** tab.

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 and click **View ➤ System Info**. Alternatively, you can double-click the corresponding Component Instantiation Warning in the **System Messages** tab.

Figure 29. 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 All System Info** to synchronize all IP components in the current system at once.

1.13. Generating a Platform Designer System

Platform Designer system generation creates the interconnect between IP components, and generates files for Intel Quartus Prime synthesis and simulation in supported third-party tools.

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 58 describes.
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 58 describes.
5. Consider whether to specify options for **Parallel IP Generation**, as Disabling or Enabling Parallel IP Generation on page 0 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 uniquness 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.

1.13.1. Generation Dialog Box Options

Platform Designer system generation creates files for Intel 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.

You can specify the following system generation options in the **Generation** dialog box:

Table 14. 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`).
Generate IP Core Documentation	Generates the IP user guide documentation for the components in your system (when available).
Create simulation model	Allows you to generate Verilog HDL or VHDL simulation model and simulation script files. *Note:* ModelSim* - Intel FPGA Edition supports native, mixed-language (VHDL/Verilog/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 ModelSim - Intel FPGA Edition, or purchase a mixed language simulation license from Mentor.
Path	Specifies the output directory path.

1.13.2. Specifying the Generation ID

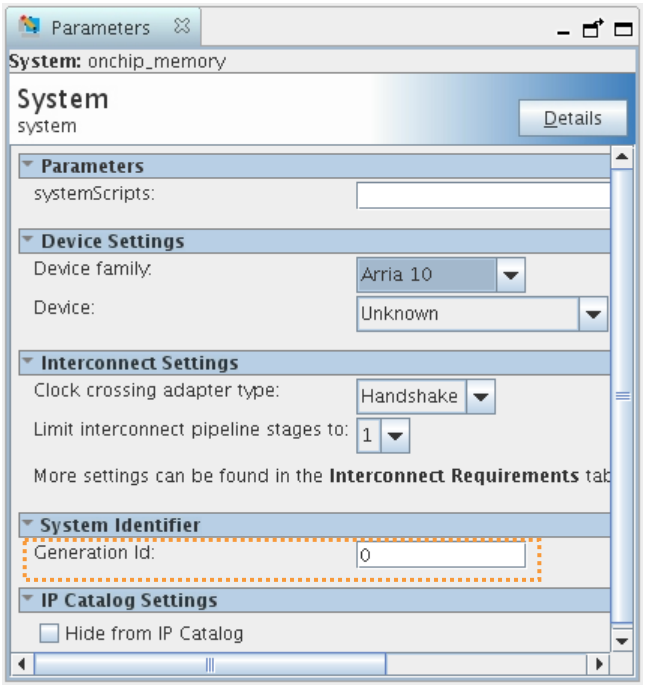
You can specify the **Generation ID** to uniquely identify that specific system generation. This parameter allows system tools, such as Nios II or HPS (Hard Processor System), to verify software-build compatibility with a specific Platform Designer system.

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**.

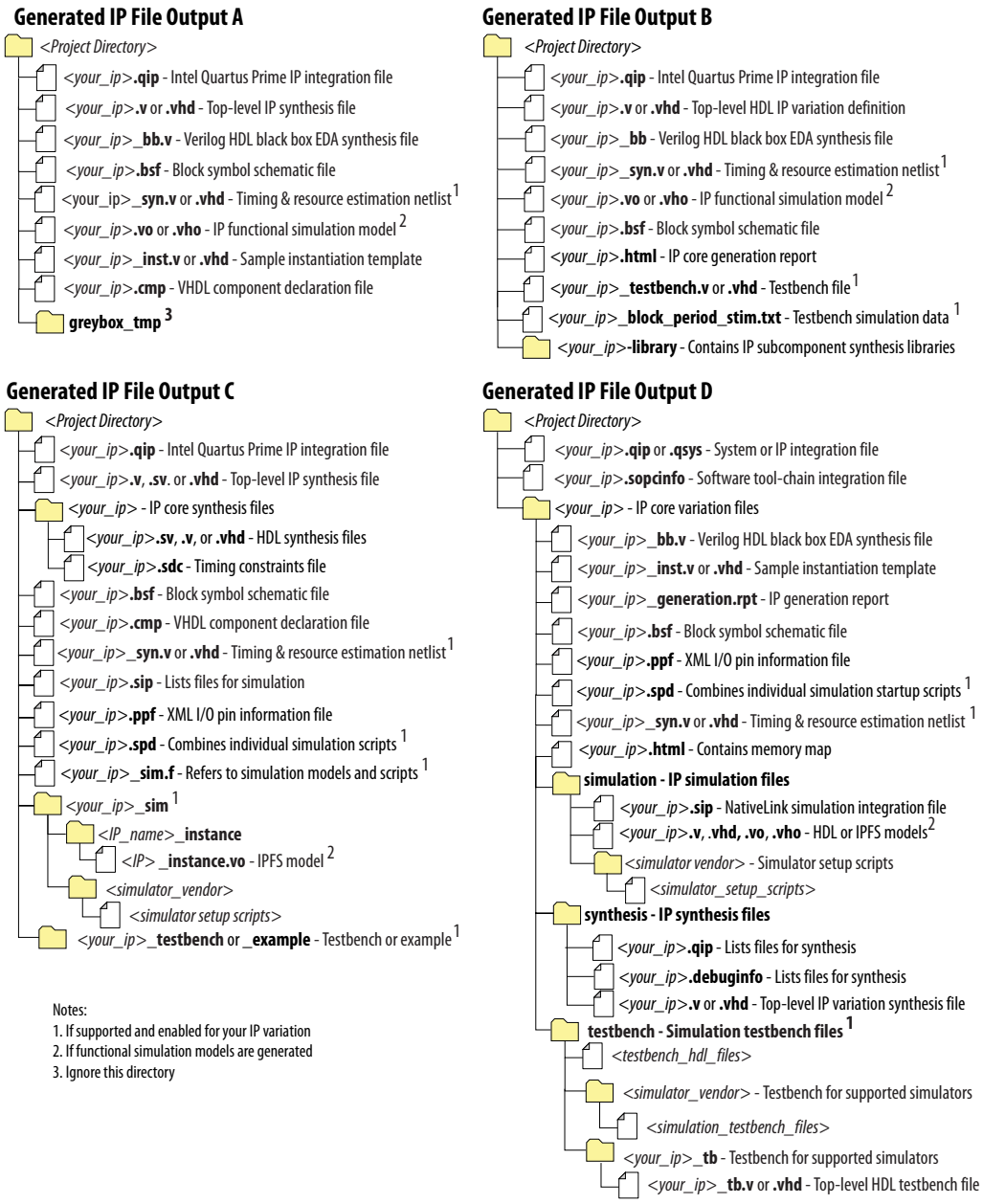
Figure 30. Generation ID in Parameters Tab



1.13.3. Files Generated for IP Cores and Platform Designer Systems

The Intel Quartus Prime Standard Edition software generates one of the following output file structures for individual IP cores that use one of the legacy parameter editors.

Figure 31. IP Core Generated Files (Legacy Parameter Editors)



1.13.4. 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:

Table 15. Testbench Generation Options

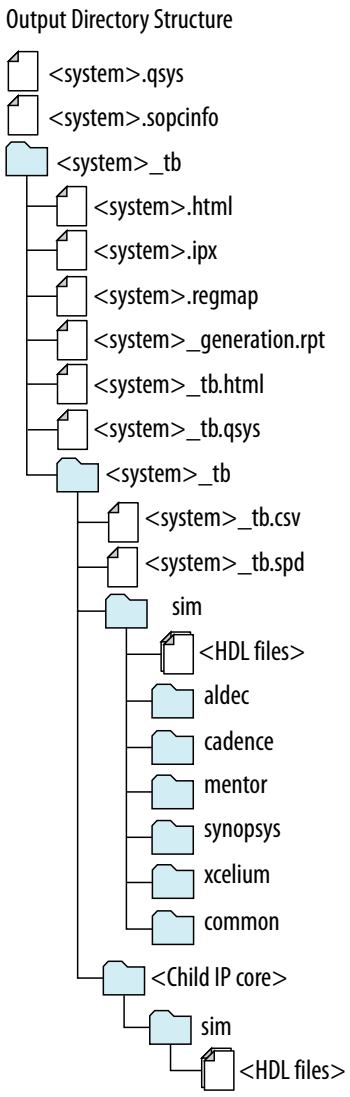
Option	Description
Create testbench Platform Designer system	Specifies a simple or standard testbench system: • **Standard, BFMs for standard Platform Designer Interconnect**—Creates a testbench Platform Designer system with BFM IP components attached to exported Avalon and AMBA 3 AXI or AMBA 3 AXI interfaces. 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 master/slave 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.

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.

1.13.4.1. Platform Designer Testbench Simulation Output Directories

Platform Designer generates the following testbench files.

Figure 32. Platform Designer Simulation Testbench Directory Structure



1.13.4.2. Platform Designer Testbench Files

Platform Designer generates the following testbench files.

Table 16. 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`.

continued...

File Name or Directory Name	Description
<*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 slave with respect to each master 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 slave interfaces visible to System Console masters 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 slaves, 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.
/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.

1.13.5. 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

1.13.6. 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 467
- CMSIS - Cortex Microcontroller Software

1.13.7. Generating Header Files for Master Components

You can use the `sopc-create-header-files` command from the Nios II command shell to create header files for any master 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 slave, relative to each master that accesses the slave. Different masters may have different address maps to access a particular slave 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 17. 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-masters`	Does not combine a module's masters 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 master.
`--module[=<moduleName>]`	Specifies the module name when creating a single header file.
`--master[=<masterName>]`	Specifies the master 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 master group in each module. A master group is a set of masters in a module in the same address space. In general, a module may have multiple master groups. Addresses and available devices are a function of the master group.

Alternatively, you can use the `--single` option to create one header file for one master group. If there is one CPU module in the Platform Designer system with one master group, the command generates a header file for that CPU's master group. If there are no CPU modules, but there is one module with one master group, the command generates the header file for that module's master group.

You can use the `--module` and `--master` options to override these defaults. If your module has multiple master groups, use the `--master` option to specify the name of a master in the desired master group.

Table 18. 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 does not have access to.

1.14. Simulating a Platform Designer System

You can simulate a Platform Designer system in a supported third-party simulator to verify and debug operation. Platform Designer generates the simulation models for your system, along with optional scripts to set up the simulation environment for specific, supported third-party simulators.

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 19. 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 4. 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
```

Note: The VHDL version of the Tristate Conduit BFM component is not supported in Synopsys VCS, NCSim, and Riviera-PRO in the Intel Quartus Prime software version 14.0. These simulators do not support the VHDL protected type, which is used to implement the BFM. For a workaround, use a simulator that supports the VHDL protected type.

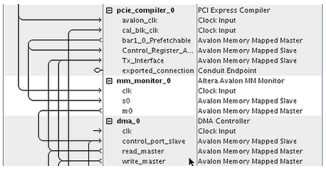
1.14.1. Adding Assertion Monitors for Simulation

You can add monitors to Avalon-MM, AXI, and Avalon-ST interfaces in your system to verify protocol and test coverage with a simulator that supports SystemVerilog assertions.

Note: ModelSim - Intel FPGA Edition does not support SystemVerilog assertions. If you want to use assertion monitors, you must use a supported third-party simulator. For more information, refer to *Introduction to Intel FPGA IP Cores*.

Figure 33. Inserting an Avalon-MM Monitor Between an Avalon-MM Master and Slave Interface

This example demonstrates the use of a monitor with an Avalon-MM monitor between the `pcie_compiler bar1_0_Prefetchable` Avalon-MM master interface, and the `dma_0 control_port_slave` Avalon-MM slave interface.



Similarly, you can insert an Avalon-ST monitor between Avalon-ST source and sink interfaces.

1.14.2. Simulating Software Running on a Nios II Processor

To simulate the software in a system driven by a Nios II processor, generate the simulation model for the Platform Designer testbench system with the following steps:

1. Click **Generate ➤ Generate Testbench System**.
2. In the **Generation** dialog box, select **Simple, BFMs for clocks and resets**.
3. For **Create testbench simulation model**, select **Verilog** or **VHDL**.
4. Click **Generate.**
5. Open the Nios II Software Build Tools for Eclipse.
6. Set up an application project and board support package (BSP) for the `<system>.sopcinfo` file.

7. To simulate, right-click the application project in Eclipse, and then click **Run as ➤ Nios II ModelSim**. This command prepares the ModelSim simulation environment, and compiles and loads the Nios II software simulation.
8. To run the simulation in ModelSim, type `run -all` in the ModelSim transcript window.
9. Set the ModelSim settings and select the Platform Designer Testbench Simulation Package Descriptor (`.spd`) file, `< system >_tb.spd`. The `.spd` file generates with the testbench simulation model for Nios II designs, and specifies the files you require for Nios II simulation.

Related Information

Nios II Gen2 Software Developer's Handbook

1.15. Integrating a Platform Designer System with the Intel Quartus Prime Software

To integrate a Platform Designer system with your Intel Quartus Prime project, you must add either the Platform Designer System File (`.qsys`) or the Intel Quartus Prime IP File (`.qip`), but never both to your Intel Quartus Prime project. Platform Designer creates the `.qsys` file when you save your Platform Designer system, and produces the `.qip` file when you generate your Platform Designer system. Both the `.qsys` and `.qip` files contain the information necessary for compiling your Platform Designer system within a Intel Quartus Prime project.

You can choose to include the `.qsys` file automatically in your Intel Quartus Prime project when you generate your Platform Designer system by turning on the **Automatically add Intel Quartus Prime IP files to all projects** option in the Intel Quartus Prime software (**Tools ➤ Options ➤ IP Settings**). If this option is turned off, the Intel Quartus Prime software asks you if you want to include the `.qsys` file in your Intel Quartus Prime project after you exit Platform Designer.

If you want file generation to occur as part of the Intel Quartus Prime software's compilation, you should include the `.qsys` file in your Intel Quartus Prime project. If you want to manually control file generation outside of the Intel Quartus Prime software, you should include the `.qip` file in your Intel Quartus Prime project.

Note: The Intel Quartus Prime software generates an error message during compilation if you add both the `.qsys` and `.qip` files to your Intel Quartus Prime project.

Does Intel Quartus Prime Overwrite Platform Designer-Generated Files During Compilation?

Platform Designer supports standard and legacy device generation. Standard device generation refers to generating files for the Intel Arria® 10 device, and later device families. Legacy device generation refers to generating files for device families prior to the release of the Intel Arria 10 device, including MAX 10 devices.

When you integrate your Platform Designer system with the Intel Quartus Prime software, if a `.qsys` file is included as a source file, Platform Designer generates standard device files under `<system>/` next to the location of the `.qsys` file. For legacy devices, if a `.qsys` file is included as a source file, Platform Designer generates HDL files in the Intel Quartus Prime project directory under `/db/ip`.

For standard devices, Platform Designer-generated files are only overwritten during Intel Quartus Prime compilation if the `.qip` file is removed or missing. For legacy devices, each time you compile your Intel Quartus Prime project with a `.qsys` file, the Platform Designer-generated files are overwritten. Therefore, you should not edit Platform Designer-generated HDL in the `/db/ip` directory; any edits made to these files are lost and never used as input to the Quartus HDL synthesis engine.

Related Information

- Generating a Platform Designer System on page 57
- IP Core Generation Output
- Introduction to Intel FPGA IP Cores
- Implementing and Parameterizing Memory IP

1.15.1. Integrate a Platform Designer System and the Intel Quartus Prime Software With the .qsys File

Use the following steps to integrate your Platform Designer system and your Intel Quartus Prime project using the `.qsys` file:

1. In Platform Designer, create and save a Platform Designer system.
2. To automatically include the `.qsys` file in the your Intel Quartus Prime project during compilation, in the Intel Quartus Prime software, select **Tools ➤ Options ➤ IP Settings**, and turn on **Automatically add Intel Quartus Prime IP files to all projects**.
3. When the **Automatically add Intel Quartus Prime IP files to all projects** option is not checked, when you exit Platform Designer, the Intel Quartus Prime software displays a dialog box asking whether you want to add the `.qsys` file to your Intel Quartus Prime project. Click **Yes** to add the `.qsys` file to your Intel Quartus Prime project.
4. In the Intel Quartus Prime software, select **Processing ➤ Start Compilation**.

1.15.2. Integrate a Platform Designer System and the Intel Quartus Prime Software With the .qip File

Use the following steps to integrate your Platform Designer system and your Intel Quartus Prime project using the `.qip` file:

1. In Platform Designer, create and save a Platform Designer system.
2. In Platform Designer, click **Generate HDL**.
3. In the Intel Quartus Prime software, select **Assignments ➤ Settings ➤ Files**.
4. On the **Files** page, use the controls to locate your `.qip` file, and then add it to your Intel Quartus Prime project.
5. In the Intel Quartus Prime software, select **Processing ➤ Start Compilation**.

1.16. 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 13

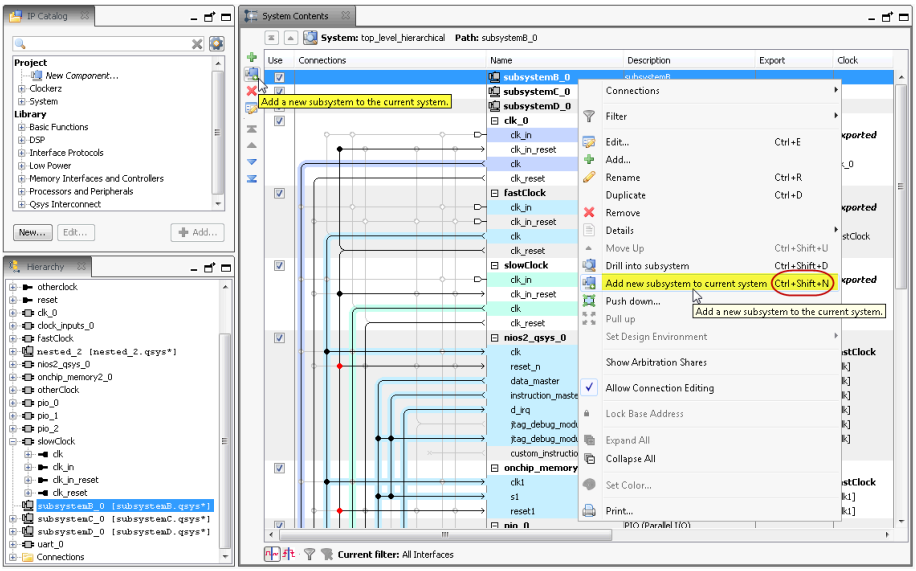
1.16.1. Adding a Subsystem to a Platform Designer System

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 34. Add a Subsystem to a Platform Designer Design



1.16.2. 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:
   - Double-click a subsystem in the **Hierarchy** tab. The subsystem opens in the **System View**.
   - Right-click a system in the **Hierarchy**, **System Contents**, or **Schematic** tabs, and then select **Drill into subsystem**.
   - 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:

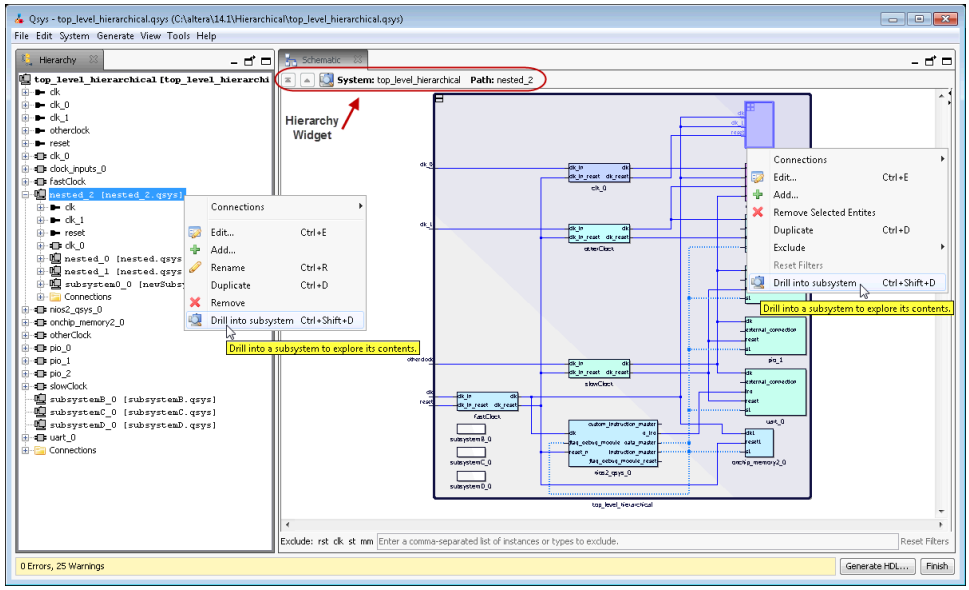
Table 20. System View and Schematic Tab Navigation Buttons

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: In the **System View** tab, you can press Ctrl+Shift+U to navigate up one level, and Ctrl+Shift+D to drill into a system.

Figure 35. Traversing Subsystem Contents

Figure 36. Traversing Subsystem Contents



1.16.3. 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.

1.16.4. 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.

1.16.5. 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 unnamed 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.

1.16.6. Exporting a System as an IP Component

You can export a Platform Designer system as a `_hw.tcl` component for use in other Platform Designer systems.

1. Open a Platform Designer system.
2. Click **File ➤ Export System as hw.tcl Component**.

The exported system displays as a new component under the **System** category in the IP Catalog.

1.16.7. Hierarchical System Using Instance Parameters Example

This example illustrates how you can use instance parameters to control the implementation of an on-chip memory component, `onchip_memory_0` when instantiated into a higher-level Platform Designer system.

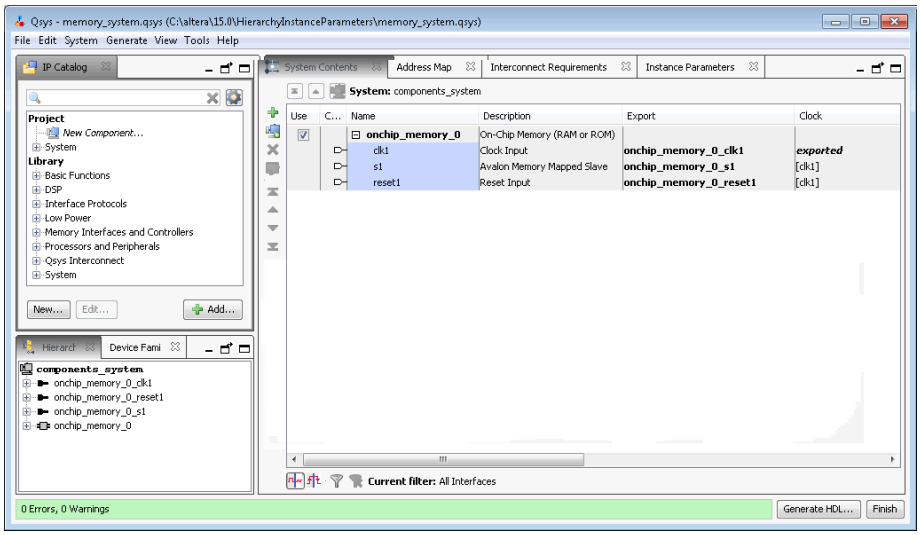
Follow the steps below to create a system that contains an on-chip memory IP component with instance parameters, and the instantiating higher-level Platform Designer system. With your completed system, you can vary the values of the instance parameters to review their effect within the On-Chip Memory component.

1.16.7.1. Create the Memory System

This procedure creates a Platform Designer system to use as subsystem as part of a hierarchical instance parameter example.

1. In Platform Designer, click **File ➤ New System**.
2. Right-click `clk_0`, and then click **Remove**.
3. In the IP Catalog search box, type `on-chip` to locate the On-Chip Memory (RAM or ROM) component.
4. Double-click to add the On-Chip Memory component to your system.
   The parameter editor opens. When you click **Finish**, Platform Designer adds the component to your system with default selections.
5. Rename the On-Chip Memory component to `onchip_memory_0`.
6. In the **System View** tab, for the `clk1` element (`onchip_memory_0`), double-click the **Export** column.
7. In the **System View** tab, for the `s1` element (`onchip_memory_0`), double-click the **Export** column.
8. In the **System View** tab, for the `reset1` element (`onchip_memory_0`), double-click the **Export** column.
9. Click **File ➤ Save** to save your Platform Designer system as `memory_system.qsys`.

Figure 37. On-Chip Memory Component System and Instance Parameters (memory_system.qsys)

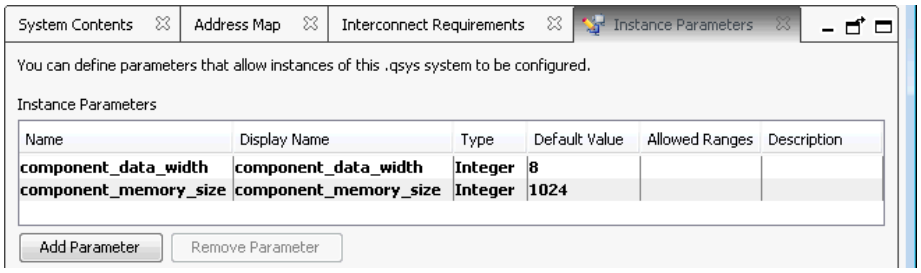


1.16.7.2. Add Platform Designer Instance Parameters

The **Instance Parameters** tab allows you to define parameters to control the implementation of a subsystem component. Each column in the **Instance Parameters** table defines a property of the parameter. This procedure creates instance parameters in a Platform Designer system to be used as a subsystem in a higher-level system.

1. In the `memory_system.qsys` system, click **View ➤ Instance Parameters**.
2. Click **Add Parameter**.
3. In the **Name** and **Display Name** columns, rename the `new_parameter_0` parameter to `component_data_width`.
4. For `component_data_width`, select **Integer** for **Type**, and 8 as the **Default Value**.
5. Click **Add Parameter**.
6. In the **Name** and **Display Name** columns, rename the `new_parameter_0` parameter to `component_memory_size`.
7. For `component_memory_size`, select **Integer** for **Type**, and **1024** as the **Default Value**.

Figure 38. Platform Designer Instance Parameters Tab

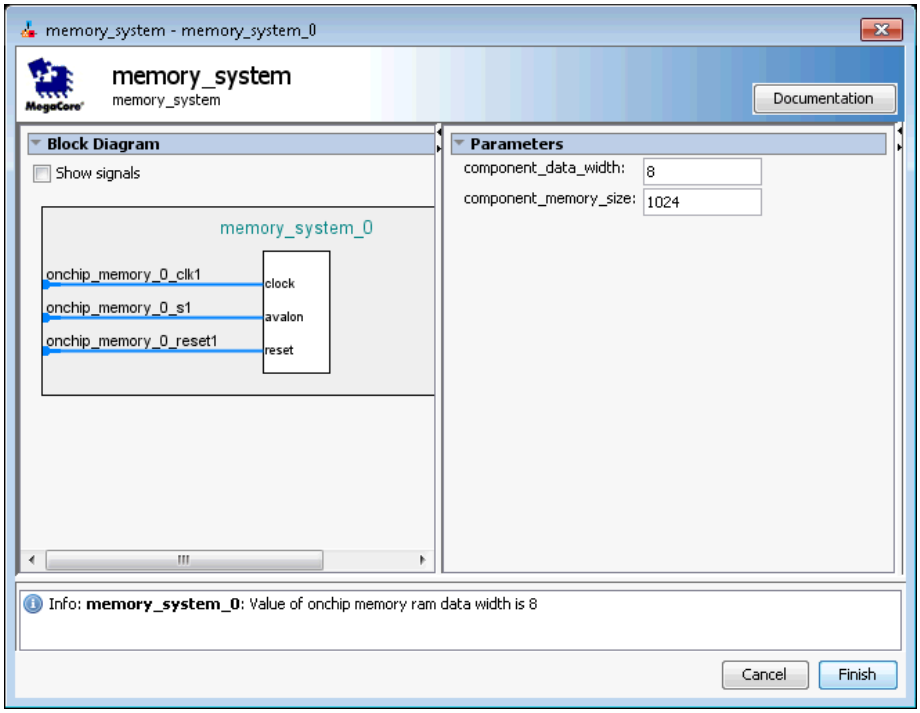


8. In the **Instance Script** section, type the commands that control how Platform Designer passes parameters to an instance from the higher-level system. For example, in the script below, the `onchip_memory_0` instance receives its `dataWidth` and `memorySize` parameter values from the instance parameters that you define.

```
# request a specific version of the scripting API
package require -exact qsys 15.0

# Set the name of the procedure to manipulate parameters
set_module_property COMPOSITION_CALLBACK compose

proc compose {} {
     # manipulate parameters in here
     set_instance_parameter_value onchip_memory_0 dataWidth
[get_parameter_value component_data_width]
     set_instance_parameter_value onchip_memory_0 memorySize
[get_parameter_value component_memory_size]

     set value [get_instance_parameter_value onchip_memory_0 dataWidth]
         send_message info "Value of onchip memory ram data width is $value "
}
```

9. Click **Preview Instance** to open the parameter editor GUI.
Preview Instance allows you to see how an instance of a system appears when you use it in another system.

Figure 39. Preview an Instance in the Parameter Editor



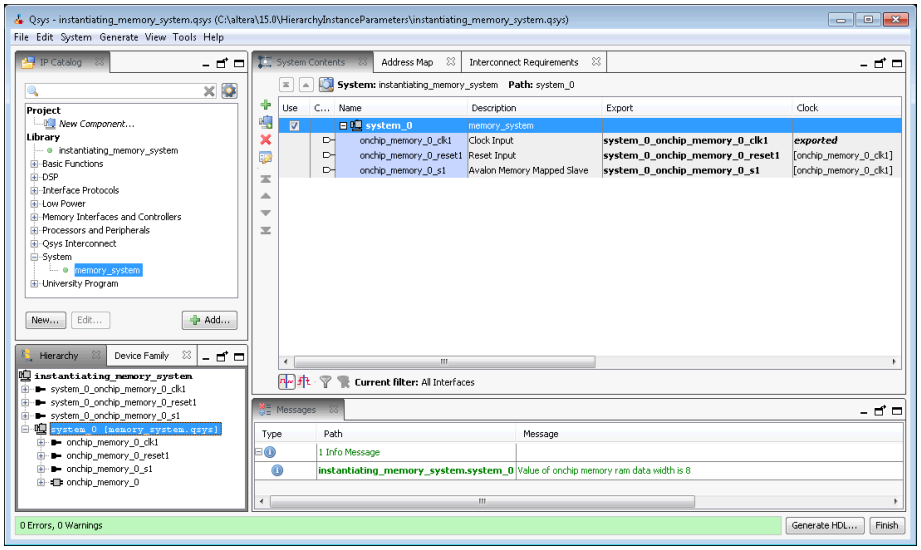
10. Click **File ➤ Save**.

1.16.7.3. Create a Platform Designer Instantiating Memory System

This procedure creates a Platform Designer system to use as a higher-level system as part of a hierarchical instance parameter example.

1. In Platform Designer, click **File ➤ New System**.
2. Right-click `clk_0`, and then click **Remove**.
3. In the IP Catalog, under **System**, double-click **memory_system**.
   The parameter editor opens. When you click **Finish**, Platform Designer adds the component to your system.
4. In the **Systems Contents** tab, for each element under **system_0**, double-click the **Export** column.
5. Click **File ➤ Save** to save your Platform Designer as `instantiating_memory_system.qsys`.

Figure 40. Instantiating Memory System (instantiating_memory_system.qsys)

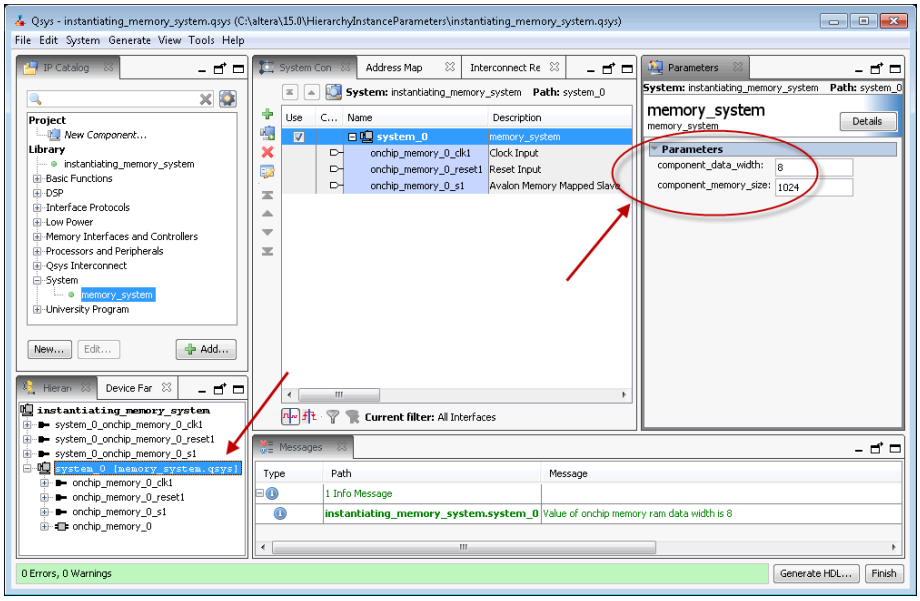


1.16.7.4. Apply Instance Parameters at a Higher-Level Platform Designer System and Pass the Parameters to the Instantiated Lower-Level System

This procedure shows you how to use Platform Designer instance parameters to control the implementation of an on-chip memory component as part of a hierarchical instance parameter example.

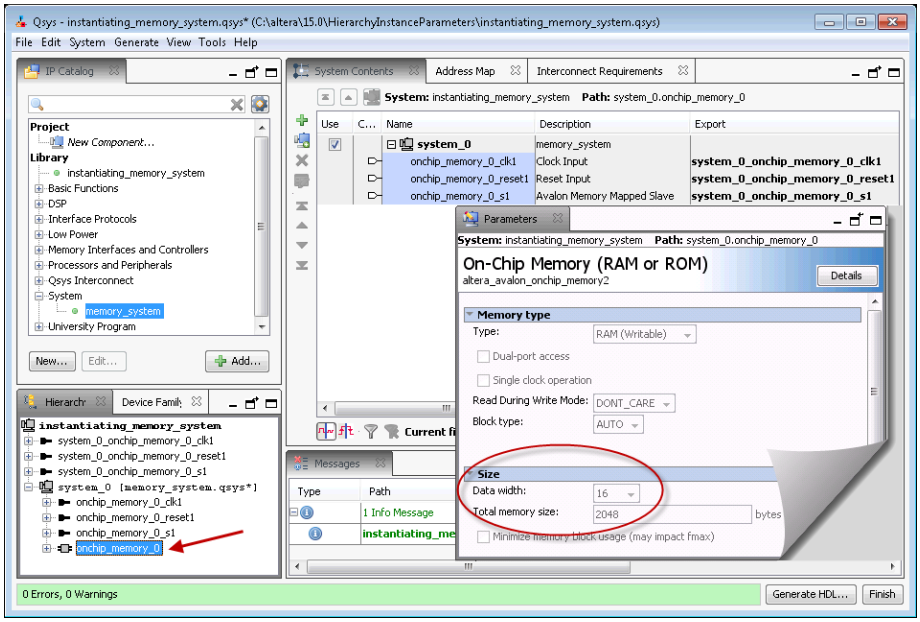
1. In the `instantiating_memory_system.qsys` system, in the **Hierarchy** tab, click and expand **system_0 (memory_system.qsys)**.
2. Click **View ➤ Parameters**.
   The instance parameters for the `memory_system.qsys` display in the parameter editor.

Figure 41. Displays memory_system.qsys Instance Parameters in the Parameter Editor



3. On the **Parameters** tab, change the value of **memory_data_width** to 16, and **memory_memory_size** to 2048.
4. In the **Hierarchy** tab, under **system_0 (memory_system.qsys)**, click `onchip_memory_0`.
   When you select `onchip_memory_0`, the new parameter values for **Data width** and **Total memory size** size are displayed.

Figure 42. Changing the Values of an Instance Parameters



1.17. Creating a System with Platform Designer Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.12.15	18.1.0	• Moved command-line utility information into new "Platform Designer Command-Line Interface" chapter. • Revised headings and re-organized content into user task-based sections.
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • Removed duplicated topic: *Manually Control Pipelining in the Platform Design Interconnect*. The topic is now in the *Platform Design Interconnect* chapter. • 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. • In topic *64-Bit Addressing Support*, added link to information about the auto base assignment feature.
2017.11.06	17.1.0	• Changed instances of *Qsys* to *Platform Designer (Standard)*
2016.05.03	16.0.0	• Qsys Command-Line Utilities updated with latest supported command-line options. • Added: *Generate Header Files*
2015.11.02	15.1.0	• Added: *Troubleshooting IP or Qsys System Upgrade*. • Added: *Generating Version-Agnostic IP and Qsys Simulation Scripts*. • Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• New figure: *Avalon-MM Write Master 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	14.1.0	• 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	14.0a10.0	• 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	14.0.0	• Added tab descriptions: Details, Connections. • Added *Managing IP Settings in the Quartus II Software*. • Added *Upgrading Outdated IP Components*. • Added *Support for Avalon-MM Non-Power of Two Data Widths*.

continued...

Document Version	Intel Quartus Prime Version	Changes
November 2013	13.1.0	• Added *Integrating with the .qsys File*. • Added *Using the Hierarchy Tab*. • Added *Managing Interconnect Requirements*. • Added *Viewing Qsys Interconnect*.
May 2013	13.0.0	• 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	12.1.0	• Added AMBA AXI4 support.
June 2012	12.0.0	• Added AMBA AX3I support. • Added Preset Editor updates. • Added command-line utilities, and scripts.
November 2011	11.1.0	• 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	11.0.0	• Added simulation support in Verilog HDL and VHDL. • Added testbench generation support. • Updated simulation and file generation sections.
December 2010	10.1.0	Initial release.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

2. 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.

Note: Intel now refers to Qsys as Platform Designer (Standard).

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 10
- Creating Platform Designer Components on page 286
- Platform Designer Interconnect on page 128
- Avalon Interface Specifications
- AMBA Protocol Specifications

2.1. Designing with Avalon and AXI Interfaces

Platform Designer Avalon and AXI interconnect for memory-mapped interfaces is flexible, partial crossbar logic that connects master and slave interfaces.

Avalon Streaming (Avalon-ST) 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.

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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Related Information

Creating Platform Designer Components on page 286

2.1.1. 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-ST 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.

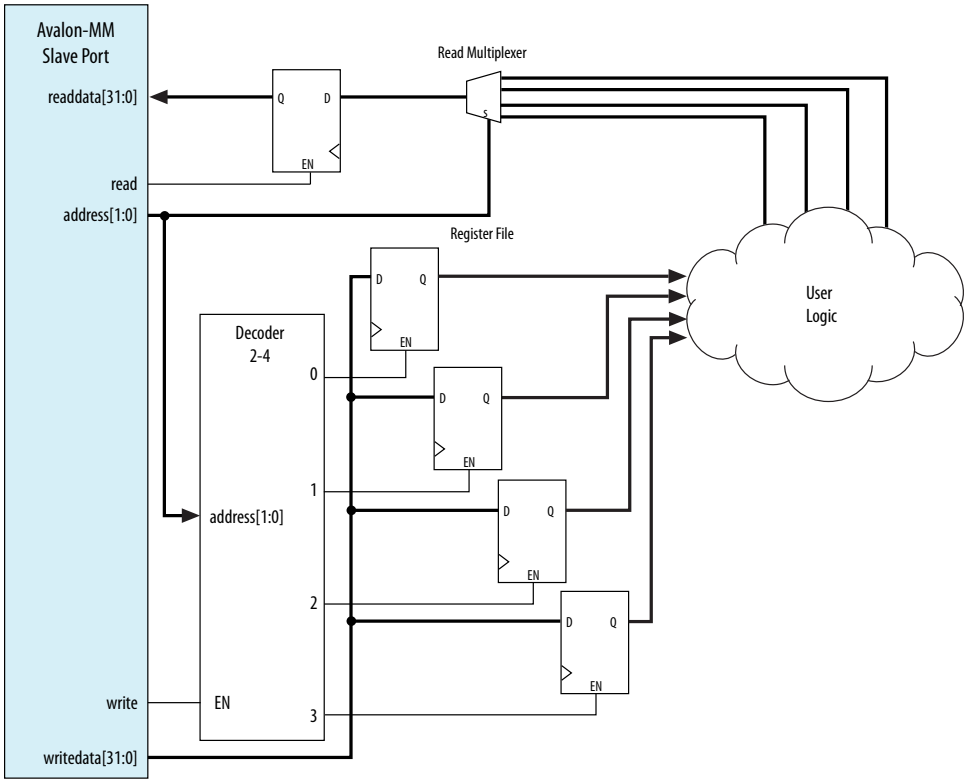
You can use a FIFO to back-pressure internally on the output side of the component so that the input can accept more data even if the output is back-pressured. Then, you can use the FIFO almost full flag to back-pressure the sink interface or input data when the FIFO has only enough space to satisfy 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.

2.1.2. 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 43. Control and Status Registers (CSR) in a Slave Component



This slave 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.

2.2. 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 slaves connected to each master 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-MM 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 44. Avalon-MM Pipeline Bridge

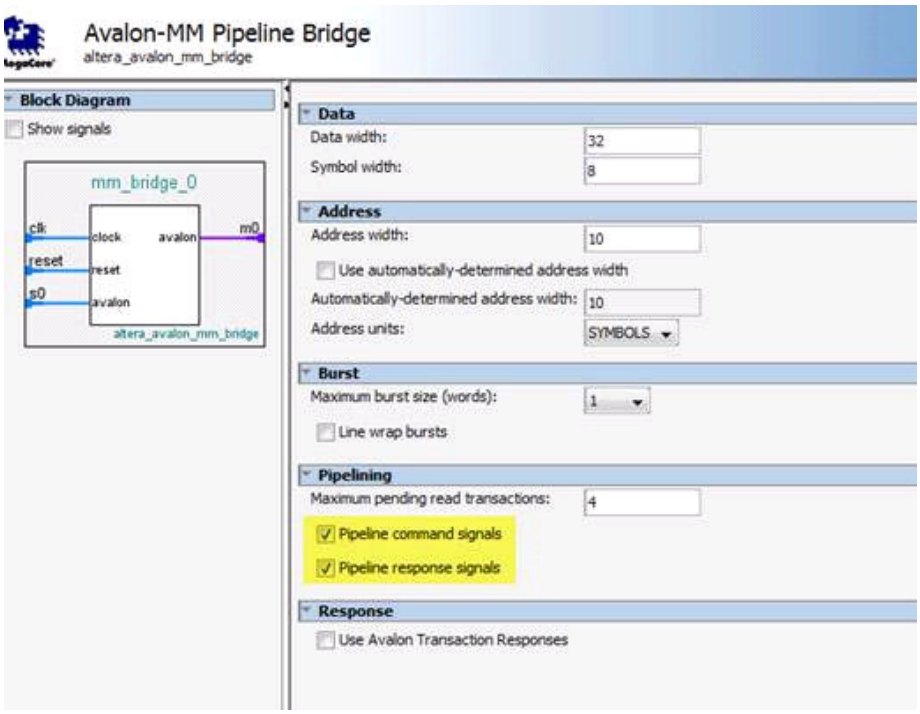
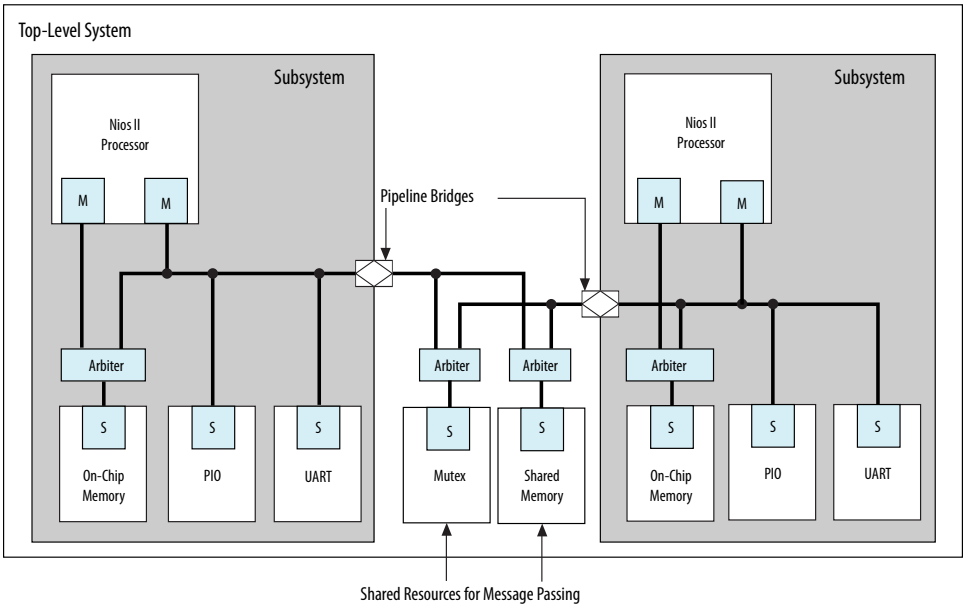
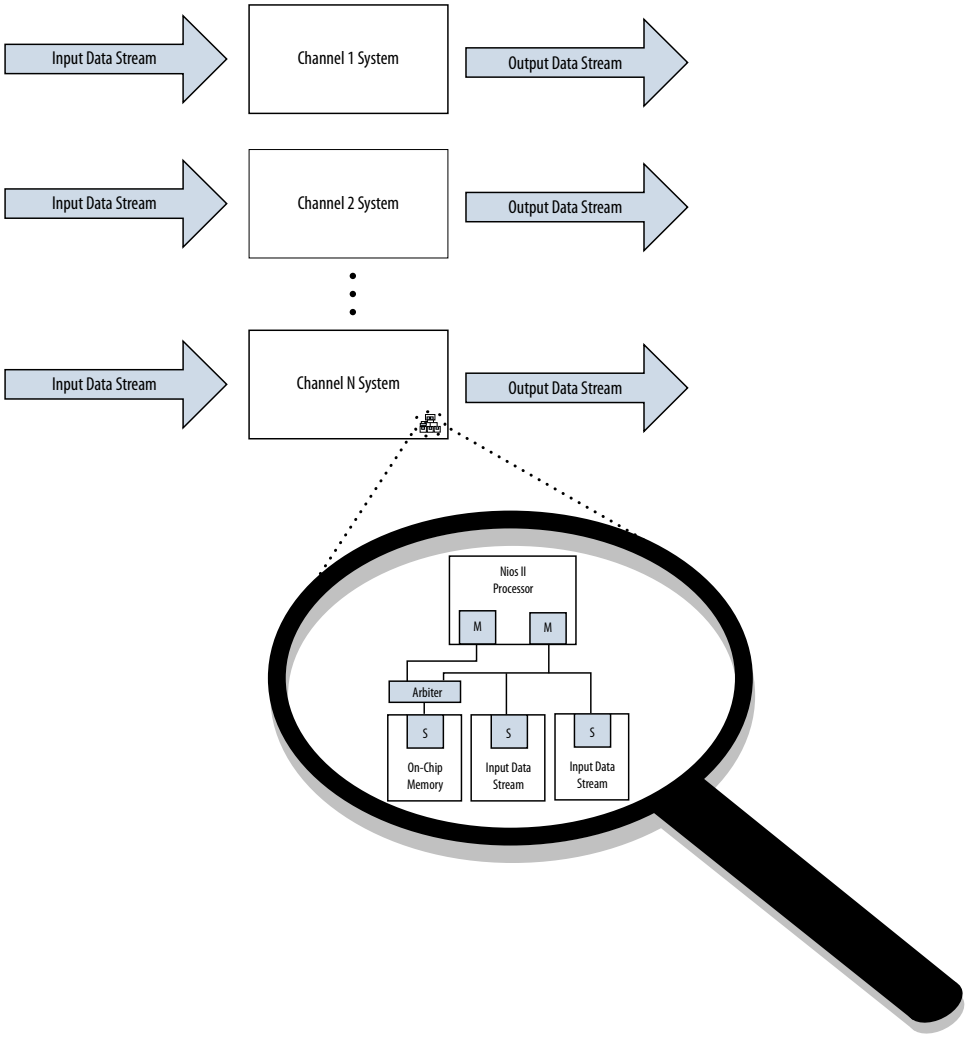


Figure 45. Passing Messages Between Subsystems



In this example, two Nios II processor subsystems share resources for message passing. Bridges in each subsystem export the Nios II data master 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 46. 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.

Related Information

Avalon-MM Pipeline Bridge

2.3. 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.

2.3.1. Implementing Concurrency With Multiple Masters

Implementing concurrency requires multiple masters in a Platform Designer system. Systems that include a processor contain at least two master interfaces because the processors include separate instruction and data masters. You can categorize master 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 slave-side arbitration, every master interface in a system can issue transfers concurrently, if they are not posting transfers to the same slave. Concurrency is limited by the number of master interfaces sharing any particular slave interface. If a design requires higher data throughput, you can increase the number of master and slave interfaces to increase the number of transfers that occur simultaneously. The example below shows a system with three master interfaces.

Figure 47. Avalon Multiple Master Parallel Access

In this Avalon example, the DMA engine operates with Avalon-MM read and write masters. The yellow lines represent active simultaneous connections.

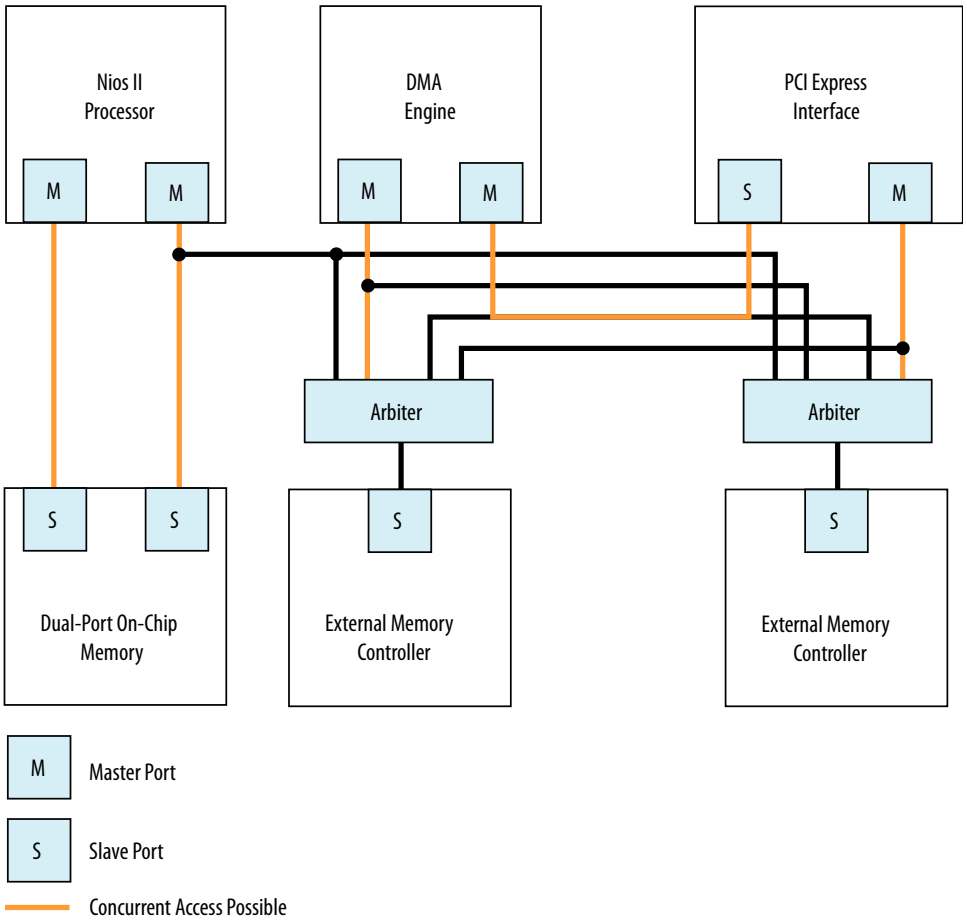
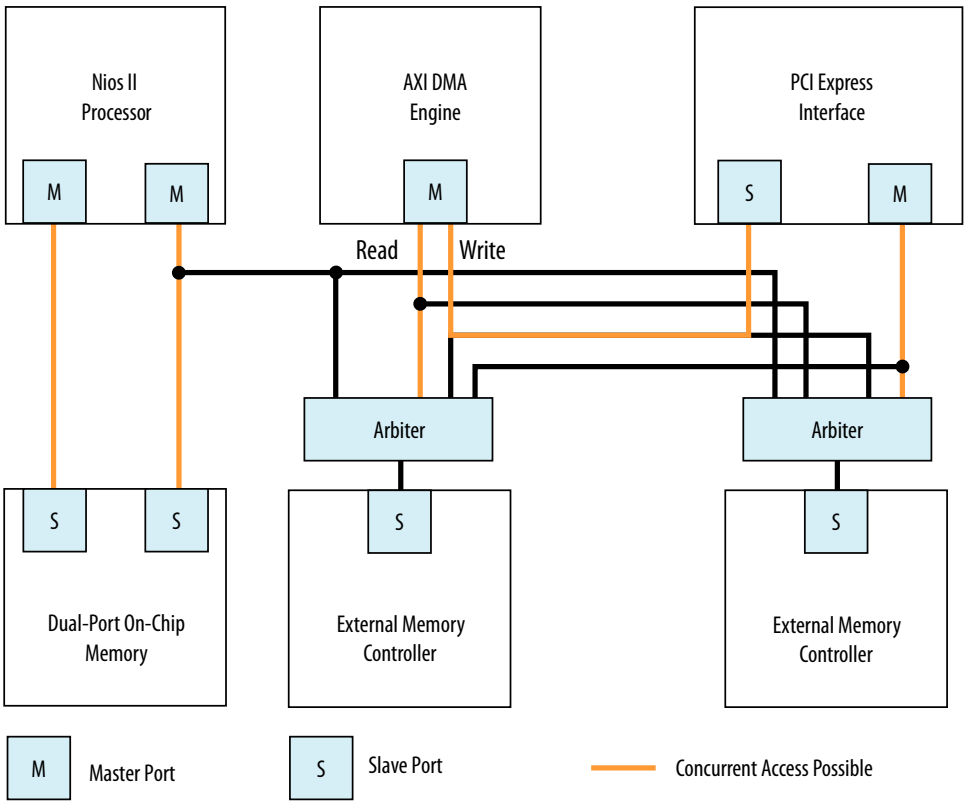


Figure 48. AXI Multiple Master Parallel Access

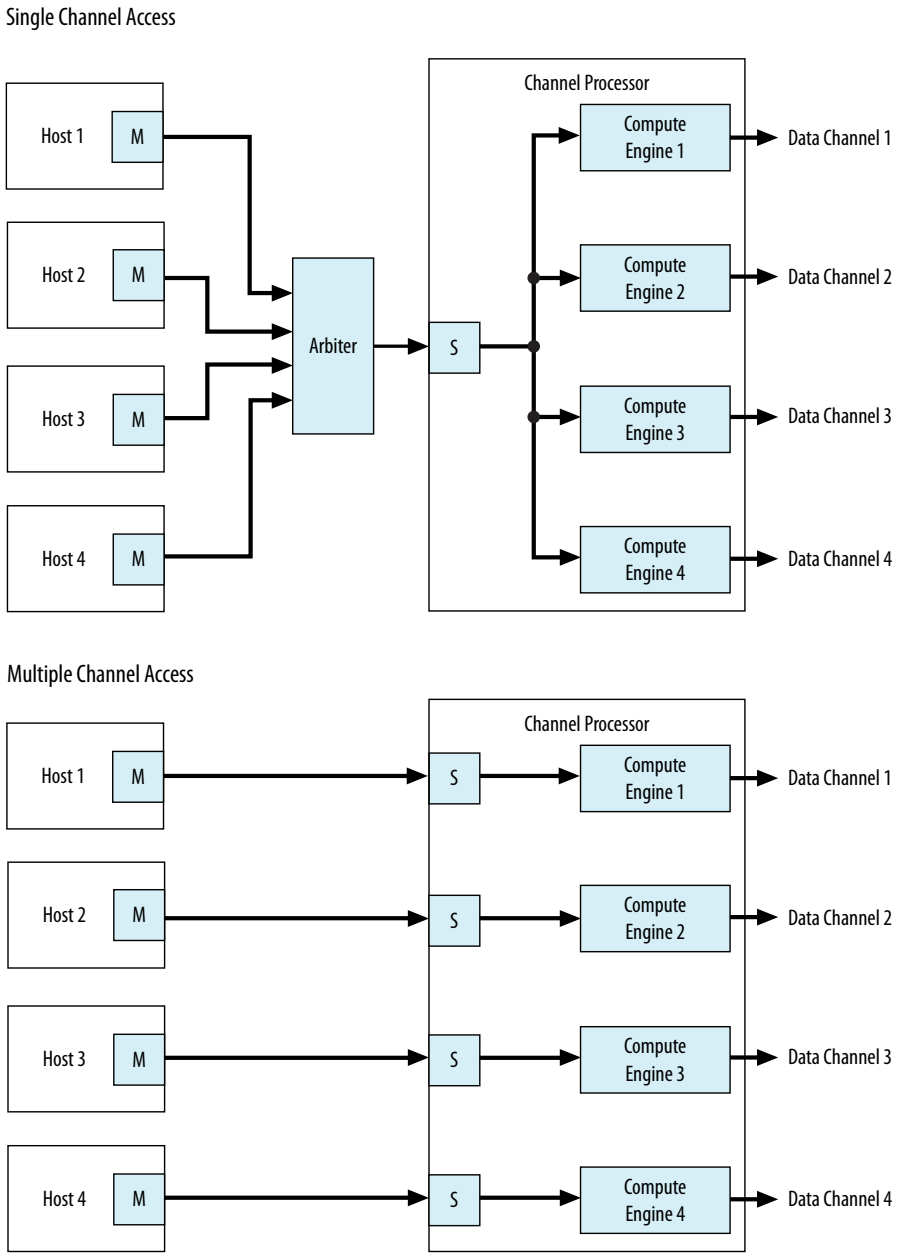
In this example, the DMA engine operates with a single master, because in AXI, the write and read channels on the master are independent and can process transactions simultaneously. There is concurrency between the read and write channels, with the yellow lines representing concurrent datapaths.



2.3.2. Implementing Concurrency With Multiple Slaves

You can create multiple slave interfaces for a particular function to increase concurrency in your design.

Figure 49. Single Interface Versus Multiple Interfaces

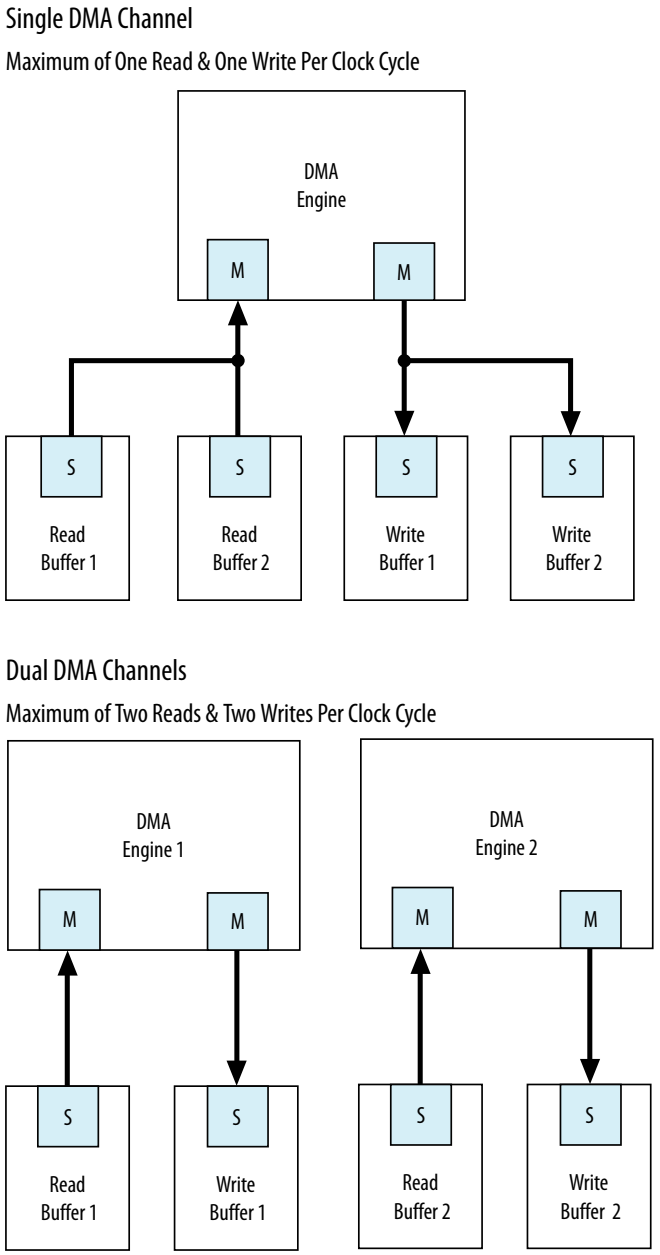


In this example, there are two channel processing systems. In the first, four hosts must arbitrate for the single slave interface of the channel processor. In the second, each host drives a dedicated slave interface, allowing all master interfaces to simultaneously access the slave interfaces of the component. Arbitration is not necessary when there is a single host and slave interface.

2.3.3. 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 50. 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-MM write and read masters. An AXI DMA typically has only one master, because in AXI, the write and read channels on the master are independent and can process transactions simultaneously.

2.4. 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-ST 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-MM Interfaces on page 106

2.5. 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-MM slave interface and an Avalon-MM master interface. You can have many components connected to the bridge slave interface, or many components connected to the bridge master interface. You can also have a single component connected to a single bridge slave or master 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-MM pipelined transfers with variable latency, and can also support configurable burst lengths.

Transfers to the bridge slave interface are propagated to the master 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 214
- AMBA 3 APB Protocol Specification Support (version 1.0) on page 192

2.5.1. Using Bridges to Increase System Frequency

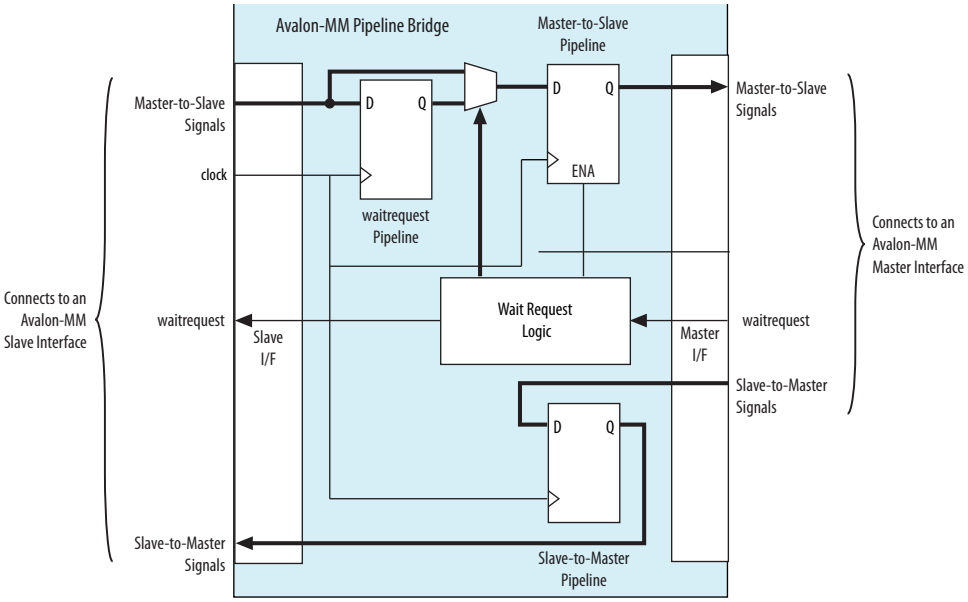
In Platform Designer, you can introduce interconnect pipeline stages or pipeline bridges to increase clock frequency in your system. Bridges control the system interconnect topology and allow you to subdivide the interconnect, giving you more control over pipelining and clock crossing functionality.

2.5.1.1. Inserting Pipeline Bridges

You can insert an Avalon-MM pipeline bridge to insert registers in the path between the bridges and its master and slaves. 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-MM 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 masters arbitrate for the bridge. You can use the Avalon-MM 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 51. Avalon-MM Pipeline Bridge

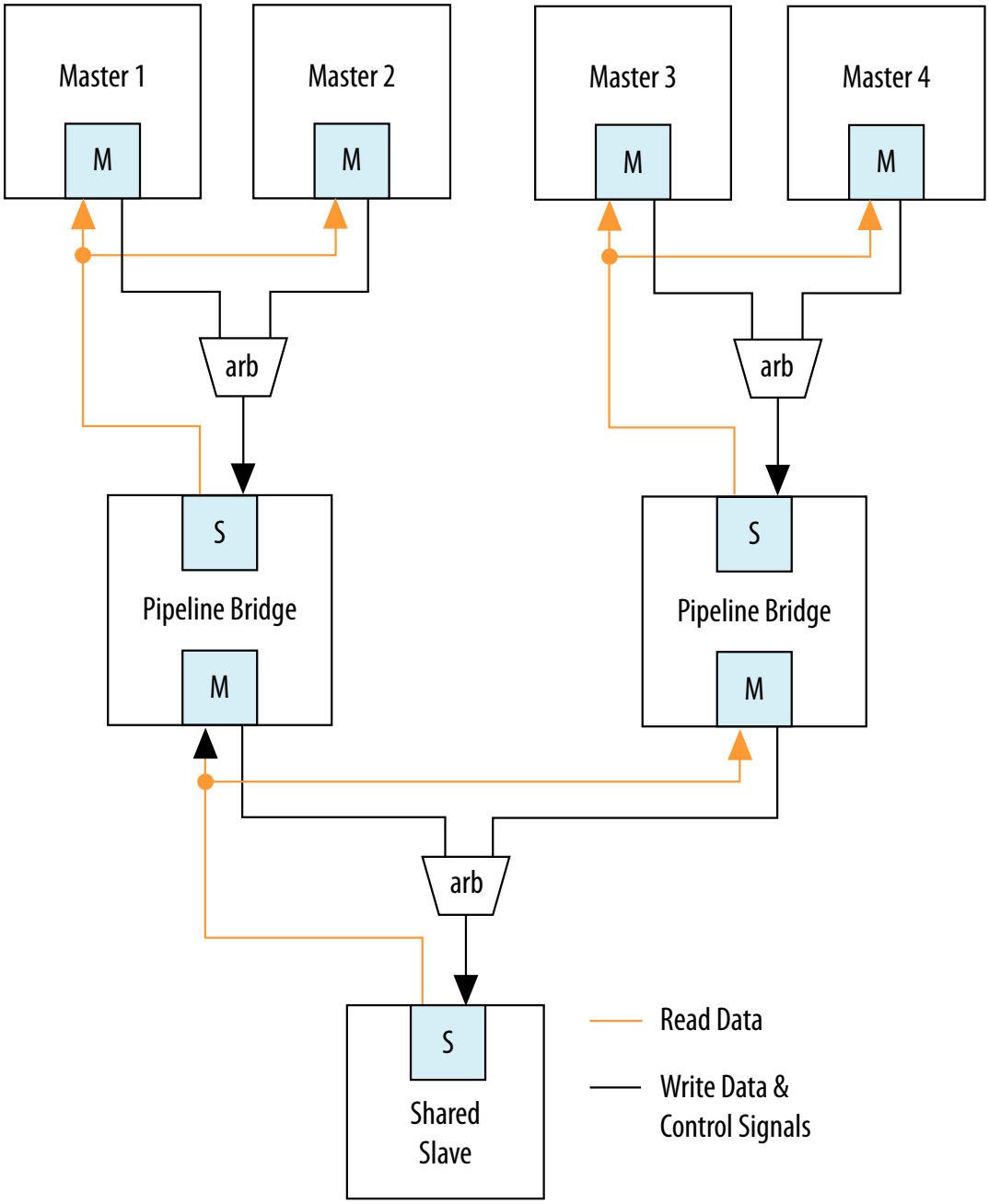


2.5.1.1.1. Implementing Command Pipelining (Master-to-Slave)

When multiple masters share a slave device, you can use command pipelining to improve performance.

The arbitration logic for the slave interface must multiplex the `address`, `writedata`, and `burstcount` signals. The multiplexer width increases proportionally with the number of masters connecting to a single slave 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 slave interface.

Figure 52. Tree of Bridges



2.5.1.1.2. Implementing Response Pipelining (Slave-to-Master)

When masters connect to multiple slaves that support read transfers, you can use slave-to-master pipelining to improve performance.

The interconnect inserts a multiplexer for every read datapath back to the master. As the number of slaves supporting read transfers connecting to the master 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}.

2.5.1.2. Using Clock Crossing Bridges

The clock crossing bridge contains a pair of clock crossing FIFOs, which isolate the master and slave interfaces in separate, asynchronous clock domains. Transfers to the slave interface are propagated to the master interface.

When you use a FIFO clock crossing bridge for the clock domain crossing, you add data buffering. Buffering allows pipelined read masters to post multiple reads to the bridge, even if the slaves 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 Intel 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.

2.5.2. 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.

2.5.2.1. Avoiding Speed Optimizations That Increase Logic

You can add an additional pipeline stage with a pipeline bridge between masters and slaves 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 Intel 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.

2.5.2.2. Limiting Concurrency

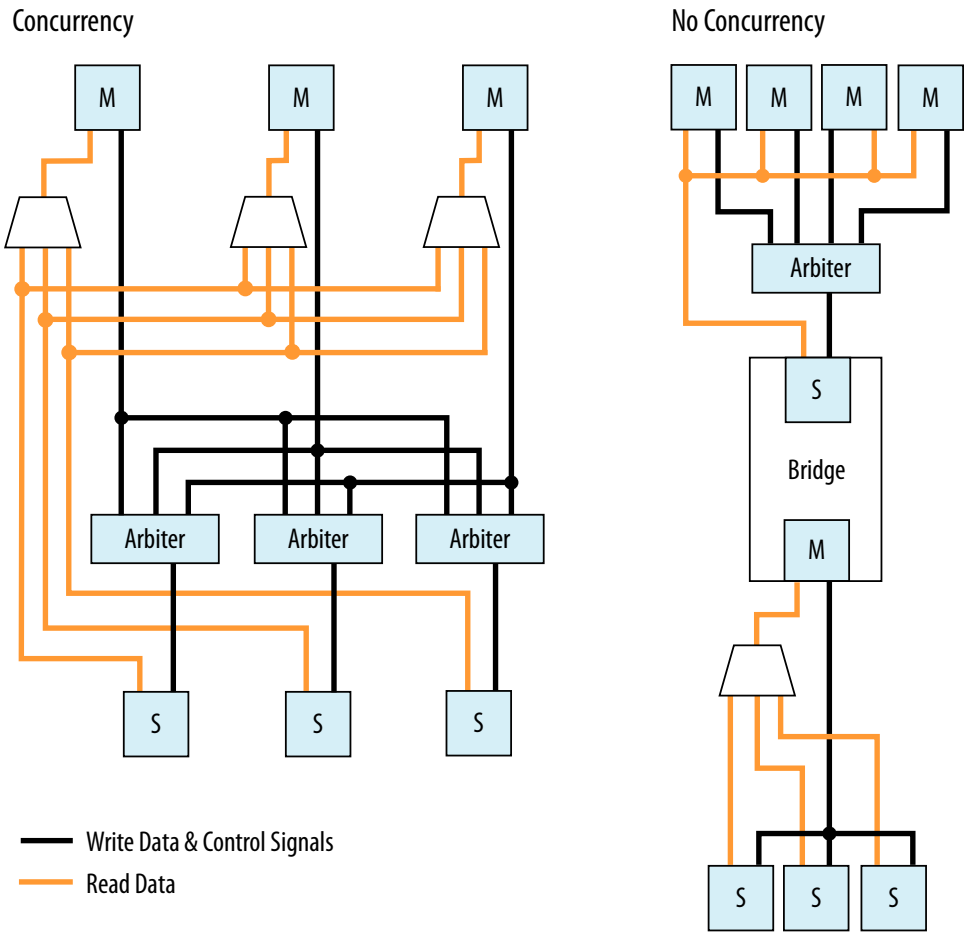
The amount of logic generated for the interconnect often increases as the system becomes larger because Platform Designer creates arbitration logic for every slave interface that is shared by multiple master interfaces. Platform Designer inserts multiplexer logic between master interfaces that connect to multiple slave 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 master or slave 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 master and three slave interfaces that are interconnected, Platform Designer generates three arbiters and three multiplexers for the read datapath. If these masters do not require a significant amount of simultaneous throughput, you can reduce the resources that your design consumes by connecting the three masters to a pipeline bridge. The bridge controls the three slave interfaces and reduces the interconnect into a bus structure. Platform Designer creates one arbitration block between the bridge and the three masters, and a single read datapath multiplexer between the bridge and three slaves, 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 53. Differences Between Systems With and Without a Pipeline Bridge



2.5.3. 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 master and slave interface pairs.

Platform Designer creates burst adapters when the maximum burst length of the master is greater than the master burst length of the slave. The adapter logic creates extra logic resources, which can be substantial when your system contains master 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.

2.5.3.1. Determining Effective Placement of Bridges

To determine the effective placement of a bridge, you should initially analyze each master in your system to determine if the connected slave 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 $^{(width(burstcount\ -1))}$, 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 master and slave interfaces, check the **Clock** column for the master and slave interfaces. If the clock is different for the master and slave interfaces, Platform Designer inserts a clock crossing adapter between them. To avoid creating multiple adapters, you can place the components containing slave 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 slaves that are connected to an AXI master, you can consider inserting a bridge to access the adaptation logic once before the bridge, instead of once per slave. This implementation results in latency, and you would also lose concurrency between reads and writes.

2.5.3.2. Changing the Response Buffer Depth

When you use automatic clock-crossing adapters, Platform Designer determines the required depth of FIFO buffering based on the slave properties. If a slave has a high **Maximum Pending Reads** parameter, the resulting deep response buffer FIFO that Platform Designer inserts between the master and slave 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 masters that cannot saturate the slave, you do not need response buffering. Using a bridge reduces the FIFO memory depth and reduces the **Maximum Pending Reads** available from the slave.

2.5.4. 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.

2.5.4.1. Increased Latency

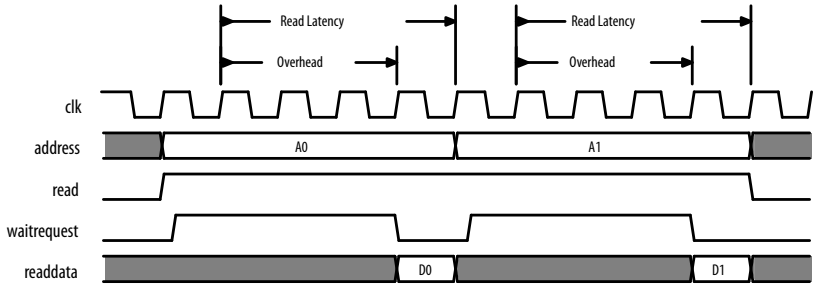
Adding a bridge to a design has an effect on the read latency between the master and the slave. Depending on the system requirements and the type of master and slave, this latency increase may not be acceptable in your design.

2.5.4.1.1. 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 master 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.

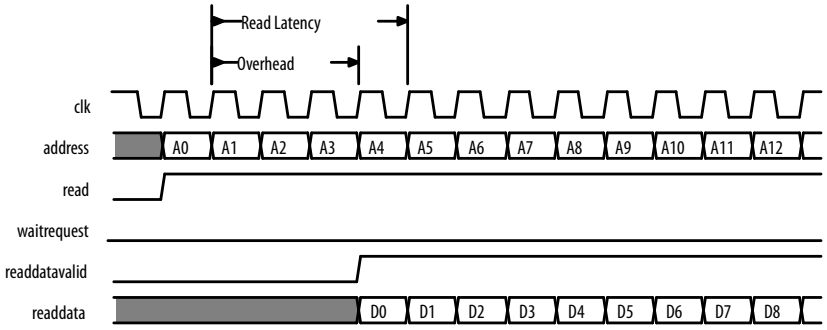
For example, if you use a pipelined read master such as a DMA controller to read data from a component with a fixed read latency of four clock cycles, but only perform a single word transfer, the overhead is three clock cycles out of the total of four. This is true when there is no additional pipeline latency in the interconnect. The read throughput is only 25%.

Figure 54. 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 55. High Efficiency Read Transfer

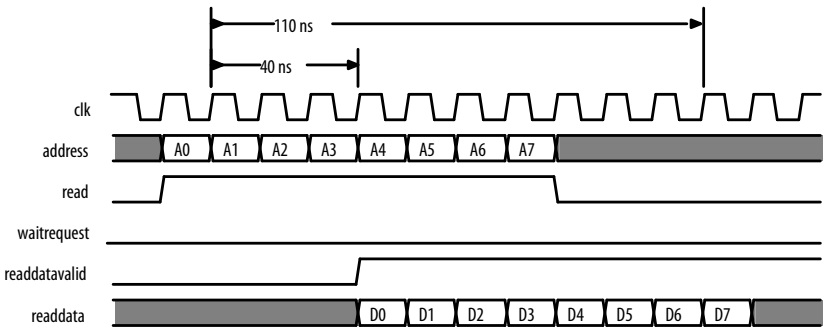


2.5.4.1.2. Unacceptable Latency Increase

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 master, determine whether the clock frequency increase justifies the added latency.

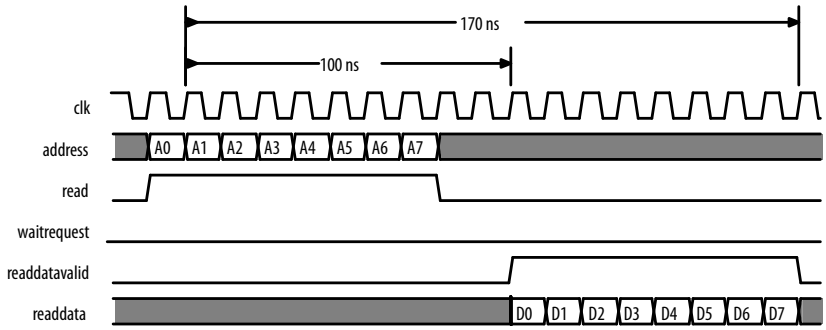
A Nios II processor instruction master has a cache memory with a read latency of four cycles, which is eight sequential words of data return for each read. At 100 MHz, the first read takes 40 ns to complete. Each successive word takes 10 ns so that eight reads complete in 110 ns.

Figure 56. 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 master operates limits the throughput.

Figure 57. Performance of a Nios II Processor and Eight Reads with Ten Cycles Latency

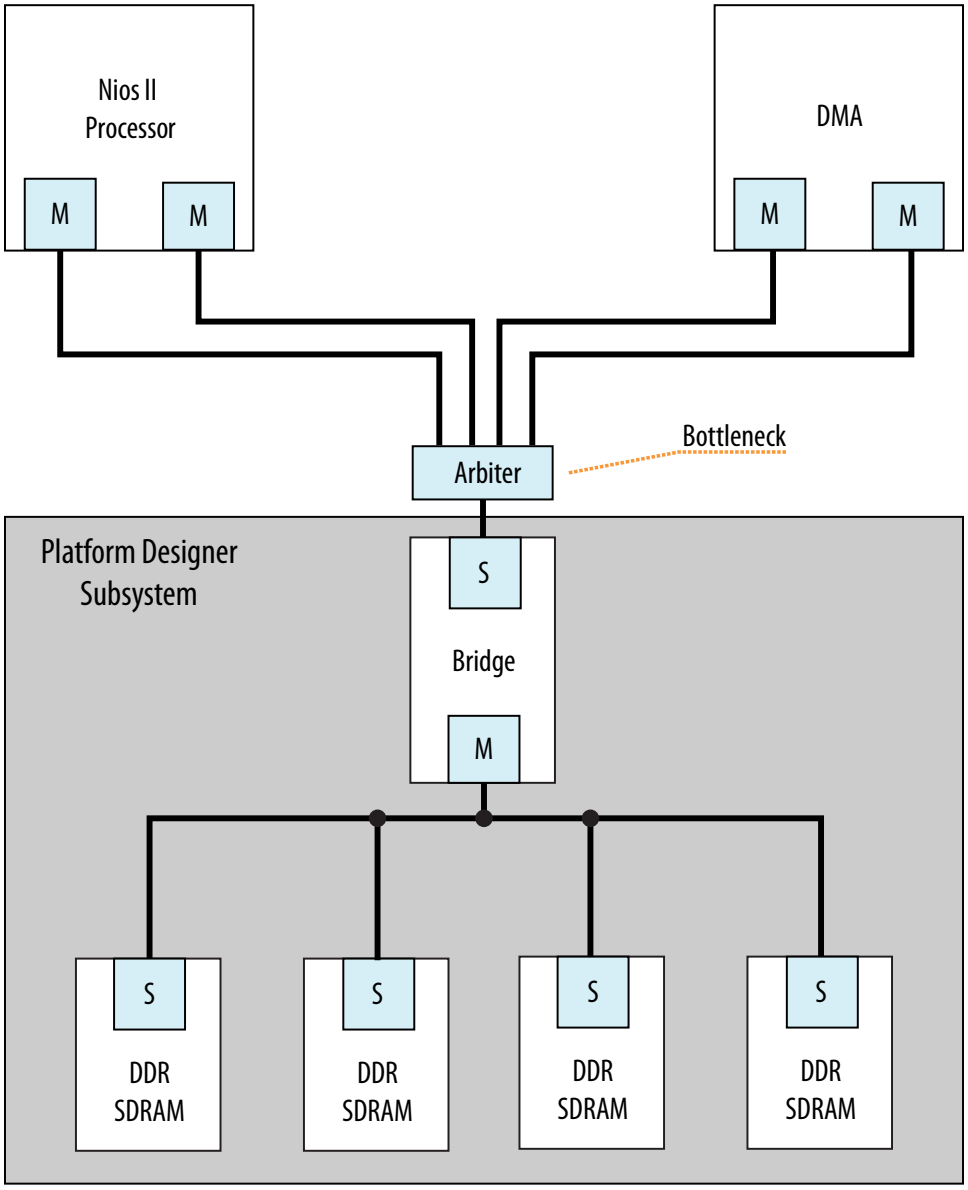


2.5.4.2. Limited Concurrency

Placing a bridge between multiple master and slave interfaces limits the number of concurrent transfers your system can initiate. This limitation is the same when connecting multiple master interfaces to a single slave interface. The slave interface of the bridge is shared by all the masters 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 masters, 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 slave interfaces to appear as one large memory to the masters accessing the bridge; all masters must access the same slave interface.

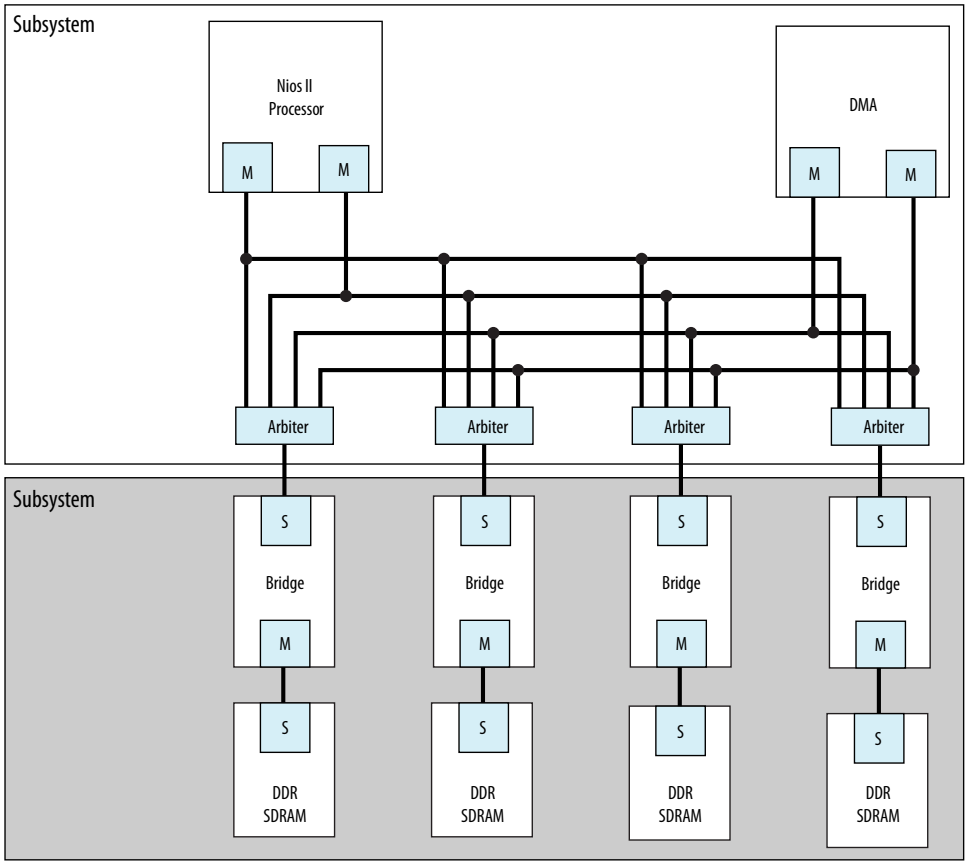
Figure 58. Inappropriate Use of a Bridge in a Hierarchical System



A memory subsystem with one bridge that acts as a single slave interface for the Avalon-MM Nios II and DMA masters, which results in a bottleneck architecture. The bridge acts as a bottleneck between the two masters 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 59. Efficient Memory Pipelining Without a Bottleneck in a Hierarchical System

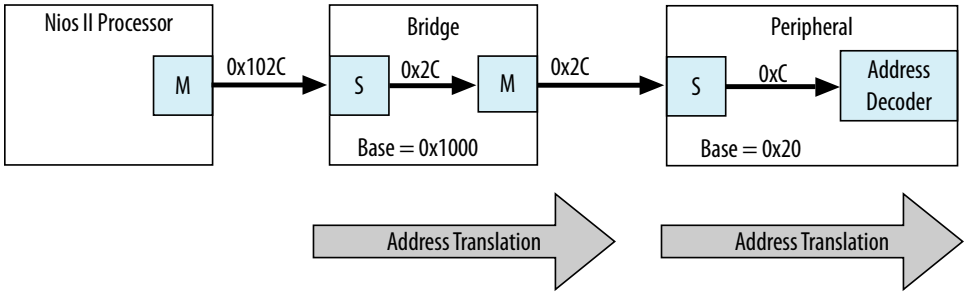


2.5.4.3. Address Space Translation

The slave 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 slave 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 master interface of the bridge drives only the address bits that represent the offset from the base address of the bridge slave interface. Any time a master accesses a slave through a bridge, both addresses must be added together, otherwise the transfer fails. The **Address Map** tab displays the addresses of the slaves connected to each master and includes address translations caused by system bridges.

Figure 60. Bridge Address Translation

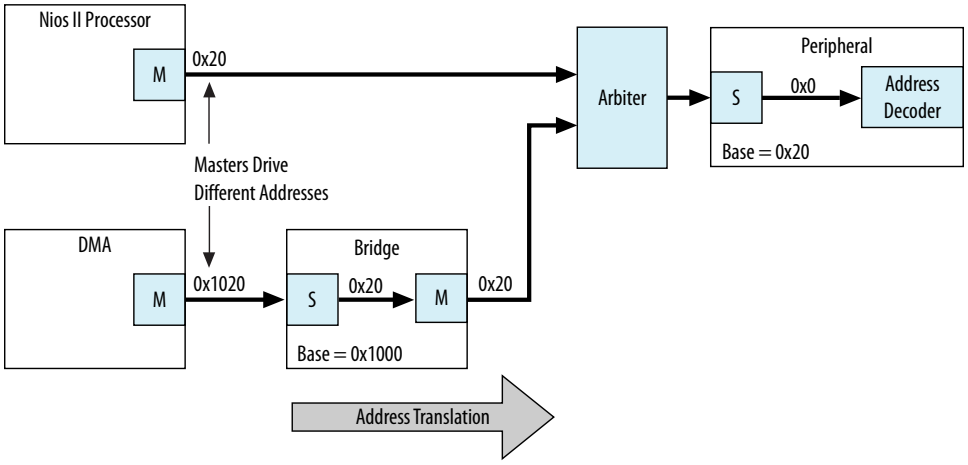


In this example, the Nios II processor connects to a bridge located at base address 0x1000, a slave connects to the bridge master interface at an offset of 0x20, and the processor performs a write transfer to the fourth 32-bit or 64-bit word within the slave. Nios II drives the address 0x102C to interconnect, which is within the address range of the bridge. The bridge master interface drives 0x2C, which is within the address range of the slave, and the transfer completes.

2.5.4.4. Address Coherency

To simplify the system design, all masters should access slaves at the same location. In many systems, a processor passes buffer locations to other mastering components, such as a DMA controller. If the processor and DMA controller do not access the slave at the same location, Platform Designer must compensate for the differences.

Figure 61. Slaves at Different Addresses and Complicating the System

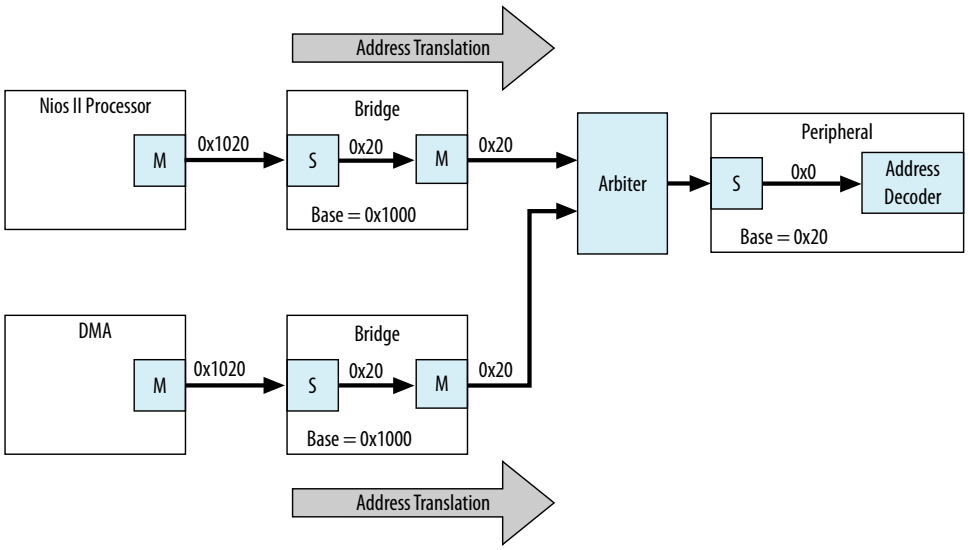


A Nios II processor and DMA controller access a slave interface located at address 0x20. The processor connects directly to the slave interface. The DMA controller connects to a pipeline bridge located at address 0x1000, which then connects to the slave interface. Because the DMA controller accesses the pipeline bridge first, it must drive 0x1020 to access the first location of the slave interface. Because the processor accesses the slave from a different location, you must maintain two base addresses for the slave 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 slave interface with the same address range.

Figure 62. Address Translation Corrected With Bridge



2.6. Increasing Transfer Throughput

Increasing the transfer efficiency of the master and slave 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 master 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 Verification IP Suite User Guide
- Mentor Graphics* Verification IP Altera Edition AMBA 3 AXI and AMBA 4 AXI User Guide

2.6.1. Using Pipelined Transfers

Pipelined transfers increase the read efficiency by allowing a master to post multiple reads before data from an earlier read returns. Masters that support pipelined transfers post transfers continuously, relying on the `readdatavalid` signal to indicate valid data. Slaves support pipelined transfers by including the `readdatavalid` signal or operating with a fixed read latency.

AXI masters declare how many outstanding writes and reads it can issue with the `writeIssuingCapability` and `readIssuingCapability` parameters. In the same way, a slave can declare how many reads it can accept with the `readAcceptanceCapability` parameter. AXI masters with a read issuing capability greater than one are pipelined in the same way as Avalon masters and the `readdatavalid` signal.

2.6.1.1. Using the Maximum Pending Reads Parameter

If you create a custom component with a slave 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 slave component can process. If the number of reads presented to the slave interface exceeds the **Maximum Pending Reads** parameter, then the slave 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 master 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 master interface to stall with a `waitrequest` until the slave 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 master connected to the slave. 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 slave that has the highest value. You can limit the maximum pending reads of a slave 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 slave 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 master 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 masters accessing the bridge.

2.6.2. Arbitration Shares and Bursts

Arbitration shares provide control over the arbitration process. By default, the arbitration algorithm allocates evenly, with all masters receiving one share.

You can adjust the arbitration process by assigning a larger number of shares to masters that need greater throughput. The larger the arbitration share, the more transfers are allocated to the master to access a slave. The masters gets uninterrupted access to the slave for its number of shares, as long as the master is reading or writing.

If a master cannot post a transfer, and other masters are waiting to gain access to a particular slave, the arbiter grants access to another master. This mechanism prevents a master 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 master to maintain access to a slave for more than a single word transfer. If a bursting master 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-MM bursting master and AXI masters (which are always considered a bursting master). Each share consists of one burst transaction (such as multi cycle write), and allows a master to complete a number of bursts before arbitration switches to the next master.

Related Information

Arbitration on page 140

2.6.2.1. 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 master posts a burst transfer, the arbitration is locked for that master; consequently, the bursting master should be capable of sustaining transfers for the duration of the locked period. If, after the fourth write, the master deasserts the write signal (Avalon-MM write or AXI `wvalid`) for fifty cycles, all other masters continue to wait for access during this stalled period.

To avoid wasted bandwidth, your master designs should wait until a full burst transfer is ready before requesting access to a slave 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 master 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 slave is capable of handling a larger burst; however, this strategy prevents stalling and allows access for other masters in the system.

Sequential Addressing

An Avalon-MM 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 master must access non-sequential addresses. Consequently, a bursting master 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 master 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 master and slave interfaces. This design strategy allows you to connect bursting master and slave interfaces that support different maximum burst lengths, with Platform Designer generating burst adapters when appropriate.

Platform Designer inserts a burst adapter whenever a master interface burst length exceeds the burst length of the slave interface, or if the master issues a burst type that the slave cannot support. For example, if you connect an AXI master to an Avalon slave, a burst adapter is inserted. Platform Designer assigns non-bursting masters and slave 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 master and slave interfaces.

2.6.2.2. Choosing Avalon-MM Interface Types

To avoid inefficient Avalon-MM transfers, custom master or slave interfaces must use the appropriate simple, pipelined, or burst interfaces.

2.6.2.2.1. Simple Avalon-MM 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 masters and infrequently used slave interfaces. In Platform Designer, the PIO, UART, and Timer include slave interfaces that use simple transfers.

2.6.2.2.2. Pipelined Avalon-MM Interfaces

Pipelined read transfers allow a pipelined master interface to start multiple read transfers in succession without waiting for prior transfers to complete. Pipelined transfers allow master-slave pairs to achieve higher throughput, even though the slave 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 slave interfaces. If your slave 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 masters. You can use pipelined masters as the default starting point for new master components. Use the `readdatavalid` signal for these master interfaces.

Because master and slaves sometimes have mismatched pipeline latency, the interconnect contains logic to reconcile the differences.

Table 21. Pipeline Latency in a Master-Slave Pair

Master	Slave	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 master to wait through any slave-side latency cycles. This master-slave pair gains no benefits from pipelining, because the master waits for each transfer to complete before beginning a new transfer. However, while the master is waiting, the slave can accept transfers from a different master.
Pipelined	No pipeline	Platform Designer interconnect carries out the transfer as if neither master nor slave were pipelined, causing the master to wait until the slave returns data. An example of a non-pipeline slave is an asynchronous off-chip interface.
Pipelined	Pipelined with fixed latency	Platform Designer interconnect allows the master to capture data at the exact clock cycle when data from the slave is valid, to enable maximum throughput. An example of a fixed latency slave is an on-chip memory.
Pipelined	Pipelined with variable latency	The slave asserts a signal when its `readdata` is valid, and the master captures the data. The master-slave pair can achieve maximum throughput if the slave has variable latency. Examples of variable latency slaves include SDRAM and FIFO memories.

2.6.2.2.3. Burst Avalon-MM 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 slave interface efficiently, you must connect to a bursting master. Components that require bursting to operate efficiently typically have an overhead penalty associated with short bursts or non-bursting transfers.

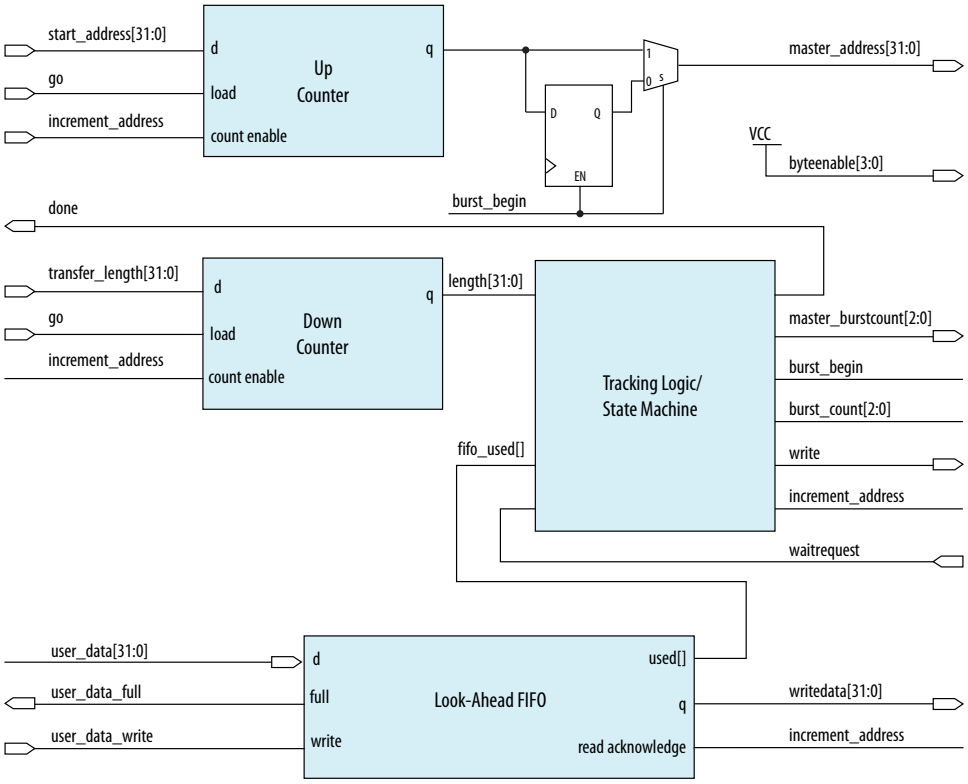
You can use a burst-capable slave interface if you know that your component requires sequential transfers to operate efficiently. Because SDRAM memories incur a penalty when switching banks or rows, performance improves when SDRAM memories are accessed sequentially with bursts.

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.

2.6.2.3. Avalon-MM Burst Master Example

Figure 63. Avalon Bursting Write Master

This example shows the architecture of a bursting write master that receives data from a FIFO and writes the contents to memory. You can use a bursting master as a starting point for your own bursting components, such as custom DMAs, hardware accelerators, or off-chip communication interfaces.



The master 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 `master_address` and `master_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 master posts a burst when enough data is buffered in the FIFO. To maximize the burst efficiency, the master should stall only when a slave 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.

Related Information

Avalon Memory-Mapped Master Templates

2.7. 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.

2.7.1. Minimizing Interconnect Logic to Reduce Logic Unitization

In Platform Designer, changes to the connections between master and slave reduce the amount of interconnect logic required in the system.

Related Information

Limited Concurrency on page 99

2.7.1.1. Creating Dedicated Master and Slave Connections to Minimize Interconnect Logic

You can create a system where a master interface connects to a single slave interface. This configuration eliminates address decoding, arbitration, and return data multiplexing, which simplifies the interconnect. Dedicated master-to-slave connections attain the same clock frequencies as Avalon-ST 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 a slave and all other master interfaces, the logic between the bridge master and slave interface is reduced to wires. If a hardware accelerator connects only to a dedicated memory, no system interconnect logic is generated between the master and slave pair.

2.7.1.2. Removing Unnecessary Connections to Minimize Interconnect Logic

The number of connections between master and slave interfaces affects the f_{MAX} of your system. Every master interface that you connect to a slave 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 masters and slaves only when necessary.

When you connect a master interface to many slave 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 (Master-to-Slave) on page 92

2.7.1.3. 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.

2.7.2. 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 slave interface that is shared by multiple master interfaces. The width of the read data multiplexer increases as the number of slave interfaces supporting read transfers increases on a per master interface basis. For these reasons, consider implementing multiple blocks of logic as a single interface to reduce interconnect logic utilization.

2.7.2.1. 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 master components and four slave interfaces, it can initiate four concurrent accesses. If you consolidate the four slave interfaces into a single interface, then the four masters 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 slave 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

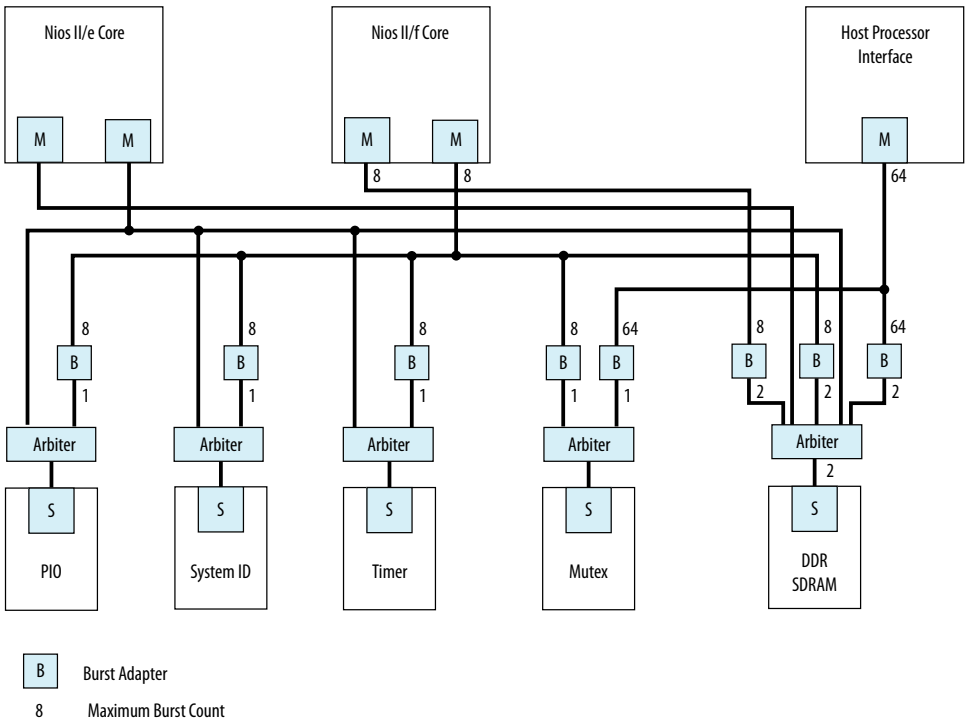
Using Concurrency in Memory-Mapped Systems on page 85

2.7.2.2. Consolidating Interfaces

In this example, we have 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 64. Mixed Bursting System

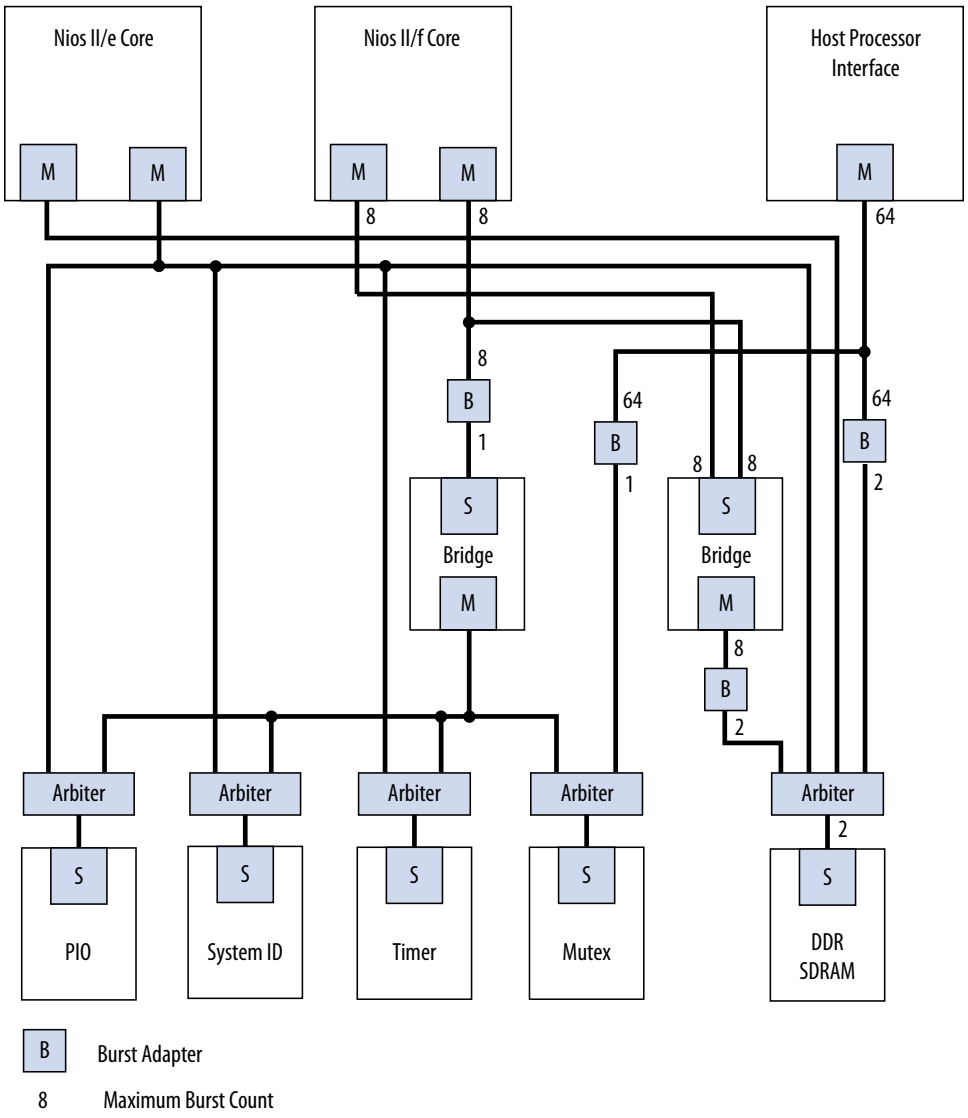


Platform Designer automatically inserts burst adapters to compensate for burst length mismatches. The adapters reduce bursts to a single transfer, or the length of two transfers. For the external processor interface connecting to DDR SDRAM, a burst of 64 words is divided into 32 burst transfers, each 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 masters and slave pairs that do not require bursting, if the master 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 65. 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.



2.7.3. 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

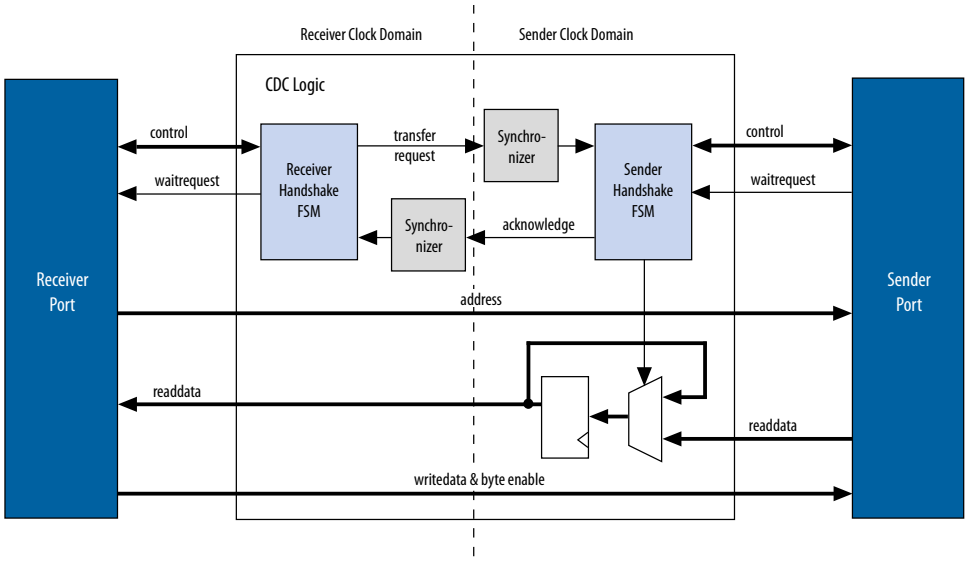
masters do not need to incorporate clock adapters in order to interface to slaves on a different domain. Platform Designer interconnect logic propagates transfers across clock domain boundaries automatically.

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 masters to access any slave without communication with the slave 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 master `waitrequest` signals) across the clock boundary.

Figure 66. Clock Crossing Adapter



This example illustrates a clock domain adapter between one master and one slave. 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 master asserts address, data, and control signals.
- The master handshake FSM captures the control signals and immediately forces the master to wait. The FSM uses only the control signals, not address and data. For example, the master simply holds the address signal constant until the slave side has safely captured it.
- The master handshake FSM initiates a transfer request to the slave handshake FSM.
- The transfer request is synchronized to the slave clock domain.
- The slave handshake FSM processes the request, performing the requested transfer with the slave.
- When the slave transfer completes, the slave handshake FSM sends an acknowledge back to the master handshake FSM. The acknowledge is synchronized back to the master clock domain.
- The master handshake FSM completes the transaction by releasing the master from the wait condition.

Transfers proceed as normal on the slave and the master side, without a special protocol to handle crossing clock domains. From the perspective of a slave, there is nothing different about a transfer initiated by a master in a different clock domain. From the perspective of a master, 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 slave side), the Platform Designer forces the master to wait until the transfer terminates. As a result, pipeline master 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 master and slave pair independently, and generates CDC logic wherever necessary.

2.7.4. Duration of Transfers Crossing Clock Domains

CDC logic extends the duration of master transfers across clock domain boundaries. In the worst case, which is for reads, each transfer is extended by five master clock cycles and five slave clock cycles. Assuming the default value of 2 for the master domain synchronizer length and the slave domain synchronizer length, the components of this delay are the following:

- Four additional master clock cycles, due to the master-side clock synchronizer.
- Four additional slave clock cycles, due to the slave-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-MM 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.

2.8. 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.

2.8.1. 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).

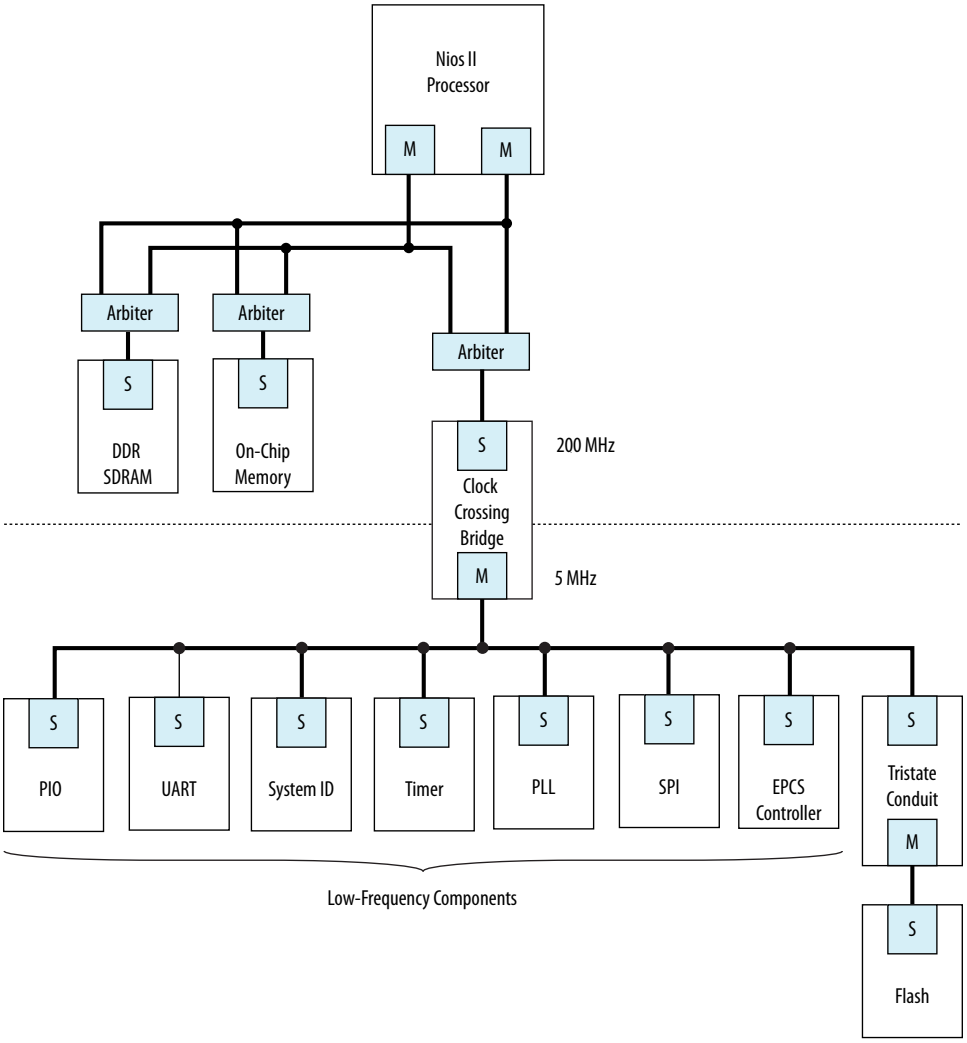
You can use clock crossing in Platform Designer to reduce the clock frequency of the logic that does not require a high frequency clock, which allows you to reduce power consumption. You can use either handshaking clock crossing bridges or handshaking clock crossing adapters to separate clock domains.

You can use the clock crossing bridge to connect master interfaces operating at a higher frequency to slave 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 67. Reducing Power Utilization Using a Bridge to Separate Clock Domains



Platform Designer automatically inserts clock crossing adapters between master and slave 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.

Related Information

Power Optimization

2.8.2. 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

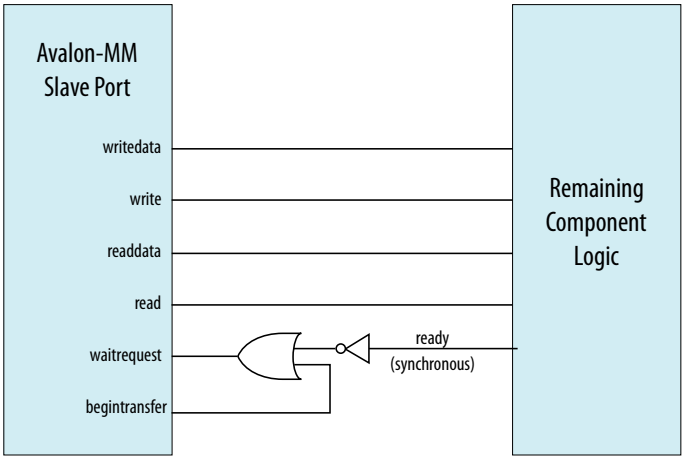
Platform Designer interconnect is uniquely combinational when no adapters or bridges are present and there is no interconnect pipelining. When a slave interface is not selected by a master, various signals may toggle and propagate into the component. By registering the boundary of your component at the master or slave 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-MM `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 master asserts read or write to in order to prolong the transfer. A master 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 combinatorially on the AXI `valid` signal. Therefore, Platform Designer typically buffers AXI component boundaries for the `ready` signal.

For slave 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 68. 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 slave 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 slave interface are repeated on the master interface. If the bridge is not accessed, the components connected to its master interface are also not accessed. The master interface of the bridge remains idle until a master accesses the bridge slave interface.

Bridges can also reduce the toggle rates of signals that are inputs to other master interfaces. These signals are typically `readdata`, `readdatavalid`, and `waitrequest`. Slave interfaces that support read accesses drive the `readdata`, `readdatavalid`, and `waitrequest` signals. A bridge inserts either a register or clock crossing FIFO between the slave interface and the master to reduce the toggle rate of the master input signals.

2.8.3. 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

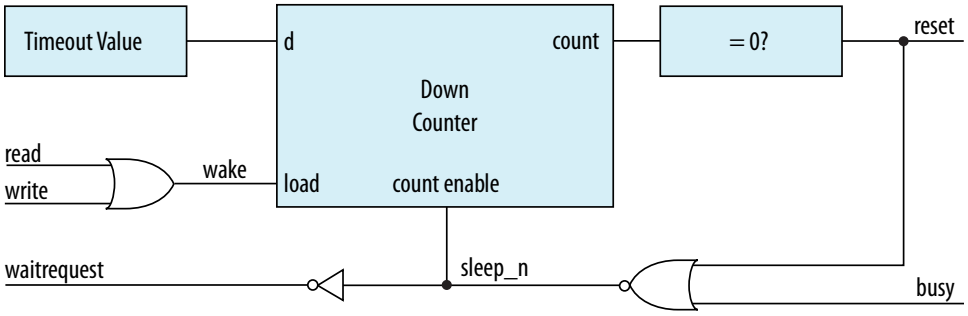
To design a component that supports software-controlled sleep mode, create a single memory-mapped location that enables and disables logic by writing a zero or one. You can use the register's output as a clock enable or reset, depending on whether the component has non-volatile requirements. The slave interface must remain active during sleep mode so that the enable bit is set when the component needs to be activated.

If multiple masters can access a component that supports sleep mode, you can use the mutex core 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 master 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 69. 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 slave 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

2.9. 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 70. Reset Signal (Active-High)

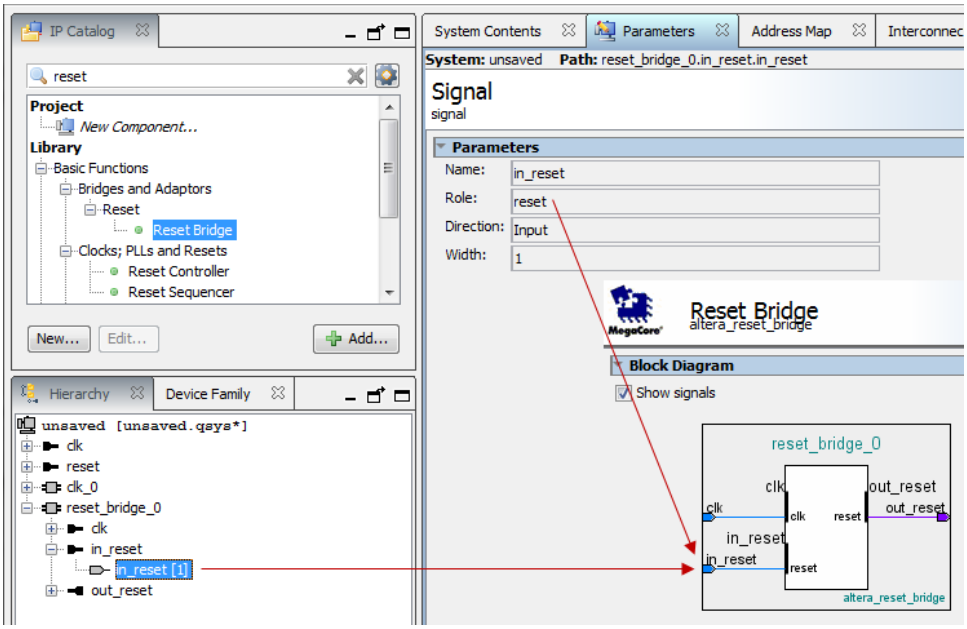
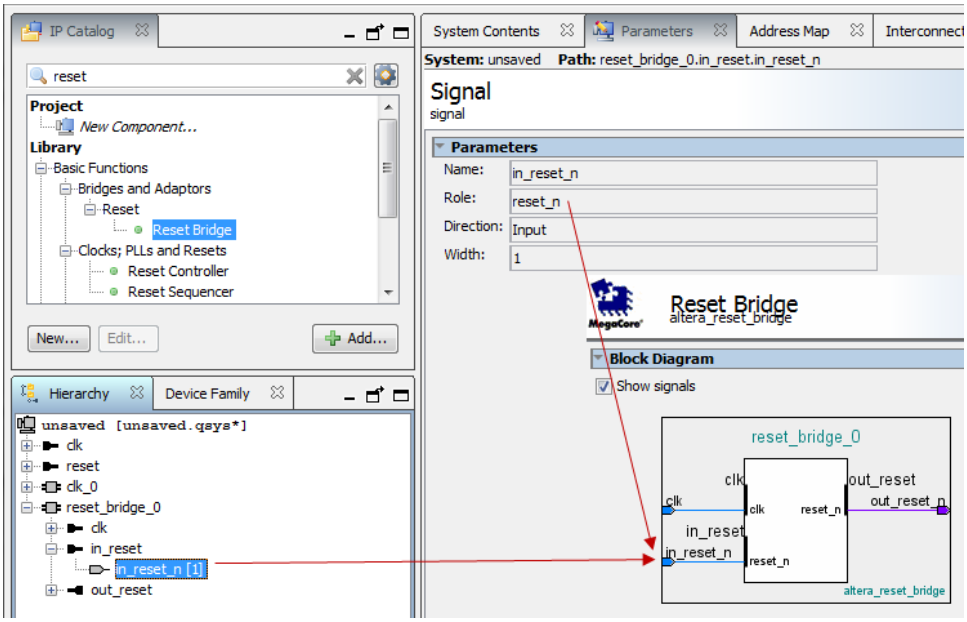


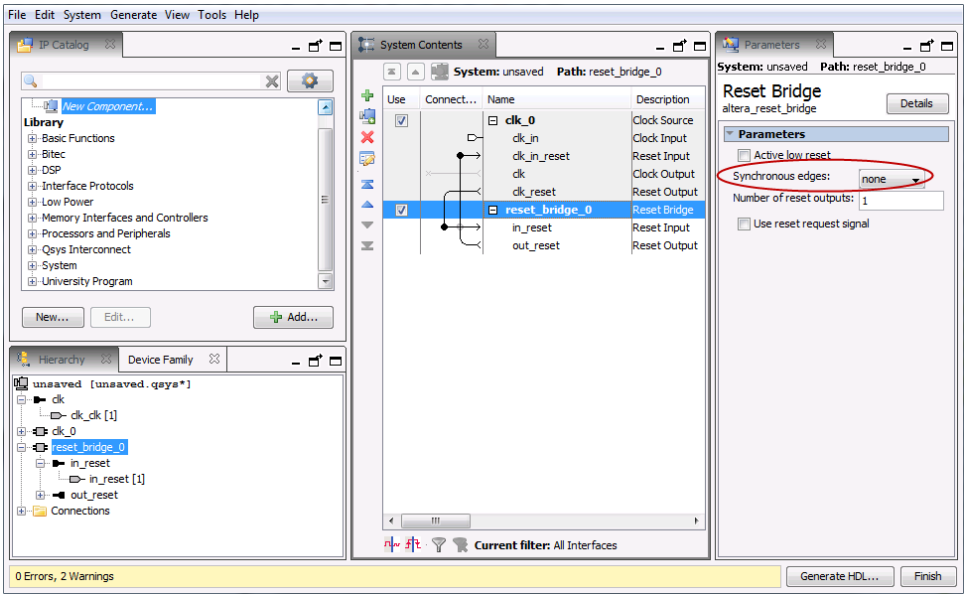
Figure 71. Reset Signal Active-Low



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:

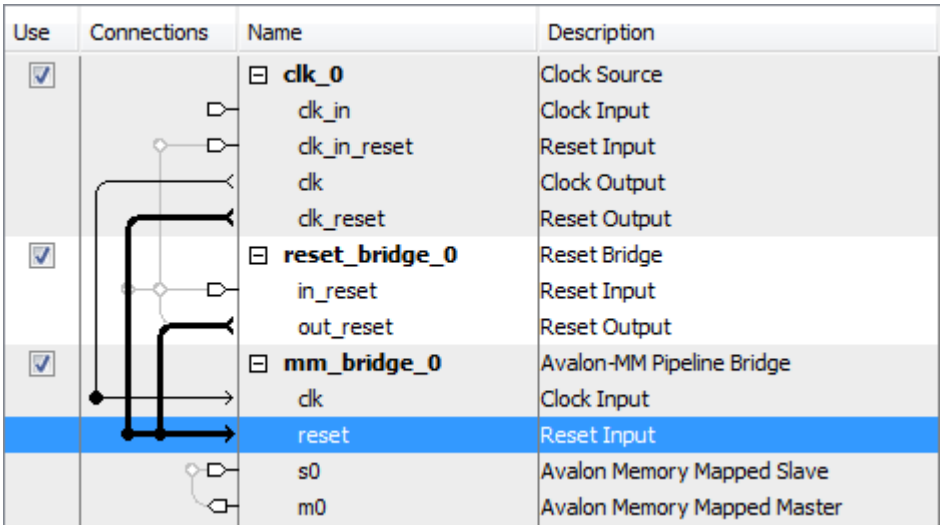
- **None**—There is no synchronization on this reset.
- **Both**—The reset is synchronously asserted and deasserted with respect to the input clock.
- **Deassert**—The reset is synchronously asserted with respect to the input clock, and asynchronously deasserted.

Figure 72. Synchronous Edges Parameter



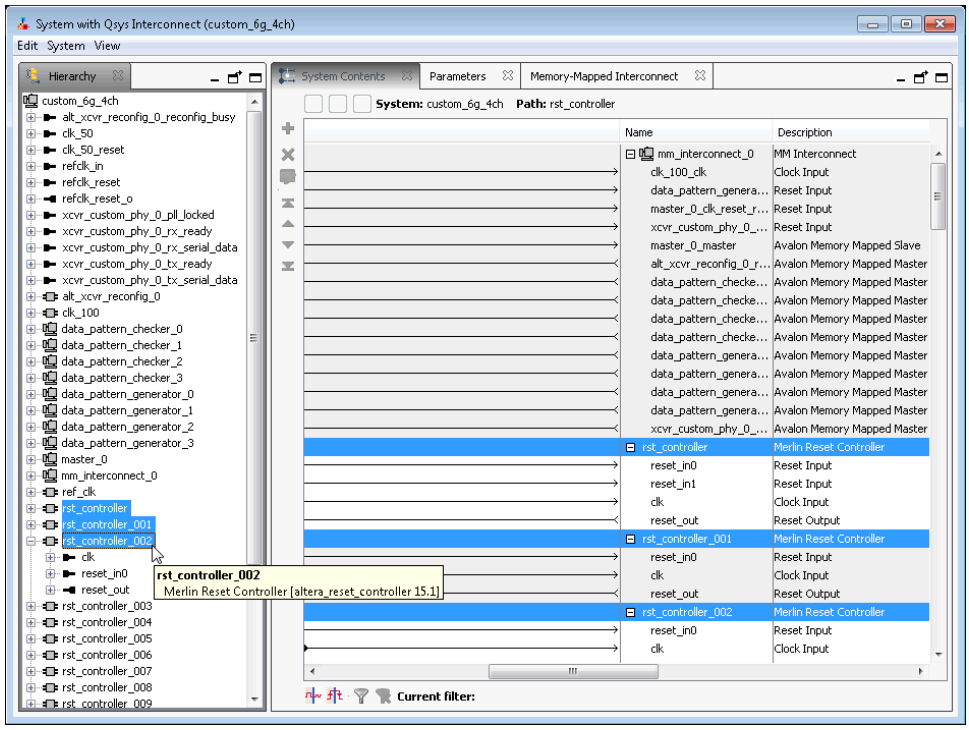
You can combine multiple reset sources to reset a particular component.

Figure 73. 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 74. Platform Designer Interconnect



2.10. Optimizing Platform Designer System Performance Design Examples

Avalon Pipelined Read Master Example on page 123

Multiplexer Examples on page 125

2.10.1. Avalon Pipelined Read Master Example

For a high throughput system using the Avalon-MM standard, you can design a pipelined read master that allows a system to issue multiple read requests before data returns. Pipelined read masters hide the latency of read operations by posting reads as frequently as every clock cycle. You can use this type of master when the address logic is not dependent on the data returning.

2.10.1.1. Avalon Pipelined Read Master Example Design Requirements

You must carefully design the logic for the control and datapaths of pipelined read masters. The control logic must extend a read cycle whenever the `waitrequest` signal is asserted. This logic must also control the master `address`, `byteenable`,

and `read` signals. To achieve maximum throughput, pipelined read masters should post reads continuously while `waitrequest` is deasserted. While `read` is asserted, the address presented to the interconnect is stored.

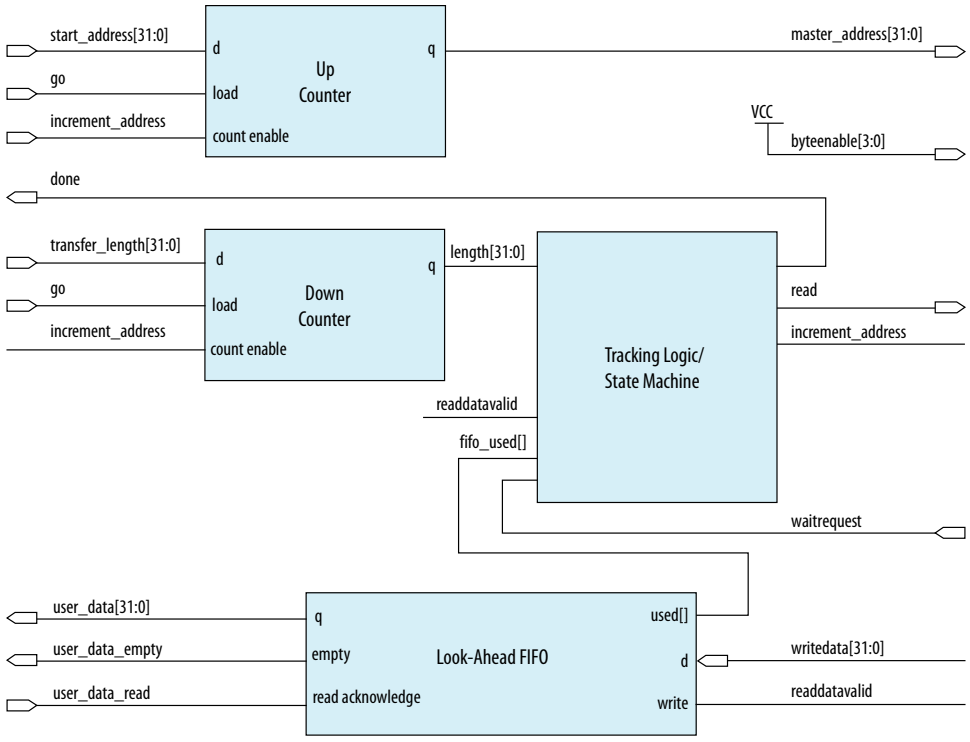
The datapath logic includes the `readdata` and `readdatavalid` signals. If your master can accept data on every clock cycle, you can register the data with the `readdatavalid` as an enable bit. If your master cannot process a continuous stream of read data, it must buffer the data in a FIFO. The control logic must stop issuing reads when the FIFO reaches a predetermined fill level to prevent FIFO overflow.

2.10.1.2. Expected Throughput Improvement

The throughput improvement that you can achieve with a pipelined read master is typically directly proportional to the pipeline depth of the interconnect and the slave interface. For example, if the total latency is two cycles, you can double the throughput by inserting a pipelined read master, assuming the slave interface also supports pipeline transfers. If either the master or slave 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 master and slave interfaces support pipelined read transfers, data flows in a continuous stream after the initial latency. You can use a pipelined read master that stores data in a FIFO to implement a custom DMA, hardware accelerator, or off-chip communication interface.

Figure 75. Pipelined Read Master



This example shows a pipelined read master that stores data in a FIFO. The master 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 master registers the `start_address` and `transfer_length` signals. The master 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 master posts a read transfer every time the `read` signal is asserted and the `waitrequest` is deasserted. The master 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.

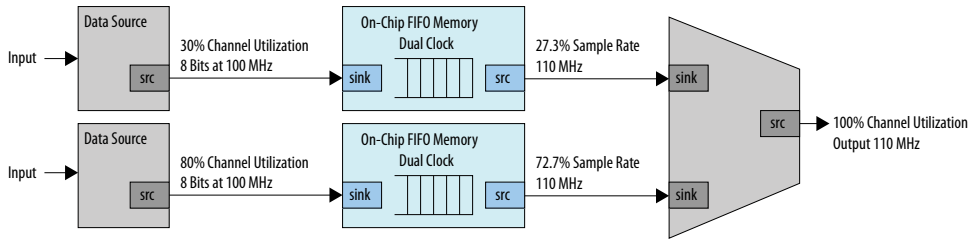
When the `length` register and the tracking logic counter reach zero, all the reads have completed and the `done` bit is asserted. The `done` bit is important if a second master overwrites the memory locations that the pipelined read master accesses. This bit guarantees that the reads have completed before the original data is overwritten.

2.10.2. 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.

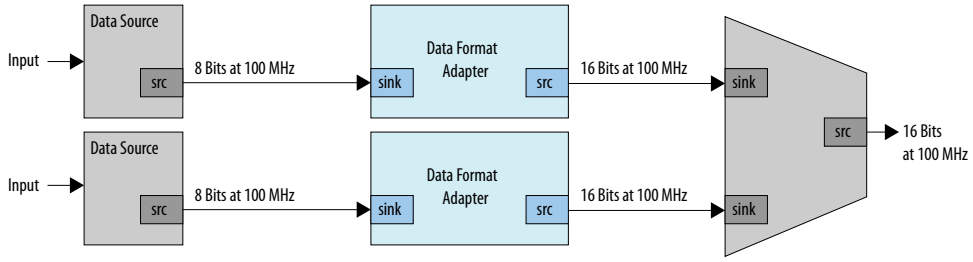
The diagram below illustrates a datapath that uses the dual clock version of the on-chip FIFO memory to boost the frequency of input data from 100 MHz to 110 MHz by sampling two input streams at differential rates. The on-chip FIFO memory has an input clock frequency of 100 MHz, and an output clock frequency 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 76. Datapath that Doubles the Clock Frequency



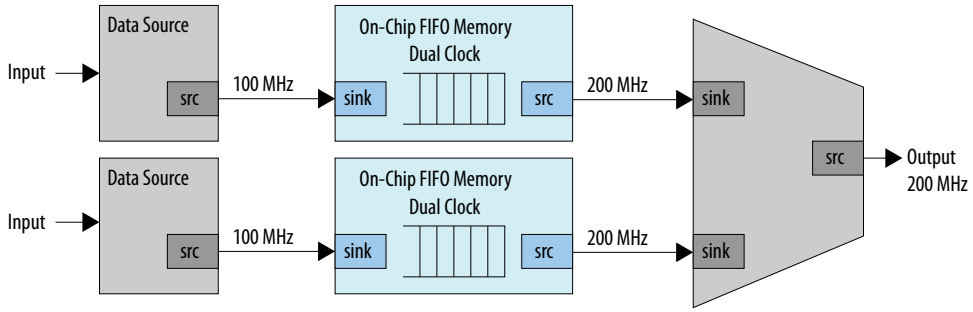
The diagram below illustrates a datapath that uses a data format adapter and Avalon-ST channel multiplexer to merge the 8-bit 100 MHz input from two streaming data sources into a single 16-bit 100 MHz streaming output. This example shows an output with double the throughput of each interface with a corresponding doubling of the data width.

Figure 77. 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-ST 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 78. Datapath to Boost the Clock Frequency



2.11. Optimizing Platform Designer System Performance Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2017.11.06	17.1.0	• Changed instances of *Qsys* to *Platform Designer (Standard)*
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.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

3. 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.

Note: Intel now refers to Qsys as Platform Designer (Standard).

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.

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.

Related Information

- Avalon Interface Specifications
- Creating a System with Platform Designer on page 10
- Creating Platform Designer Components on page 286
- Platform Designer System Design Components on page 214
- AMBA AXI and Intel Avalon Interoperation Using Platform Designer
- Specifying Interconnect Requirements on page 39

3.1. 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 masters and slaves in packets. The command network transports read and write packets from master interfaces to slave interfaces. The response network transports response packets from slave interfaces to master interfaces.

For each component interface, Platform Designer interconnect manages memory-mapped transfers and interacts with signals on the connected interface. Master and slave interfaces can implement different signals based on interface parameterizations, and Platform Designer interconnect provides any necessary adaptation between them. In the path between master and slaves, 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Platform Designer interconnect supports the following implementation scenarios:

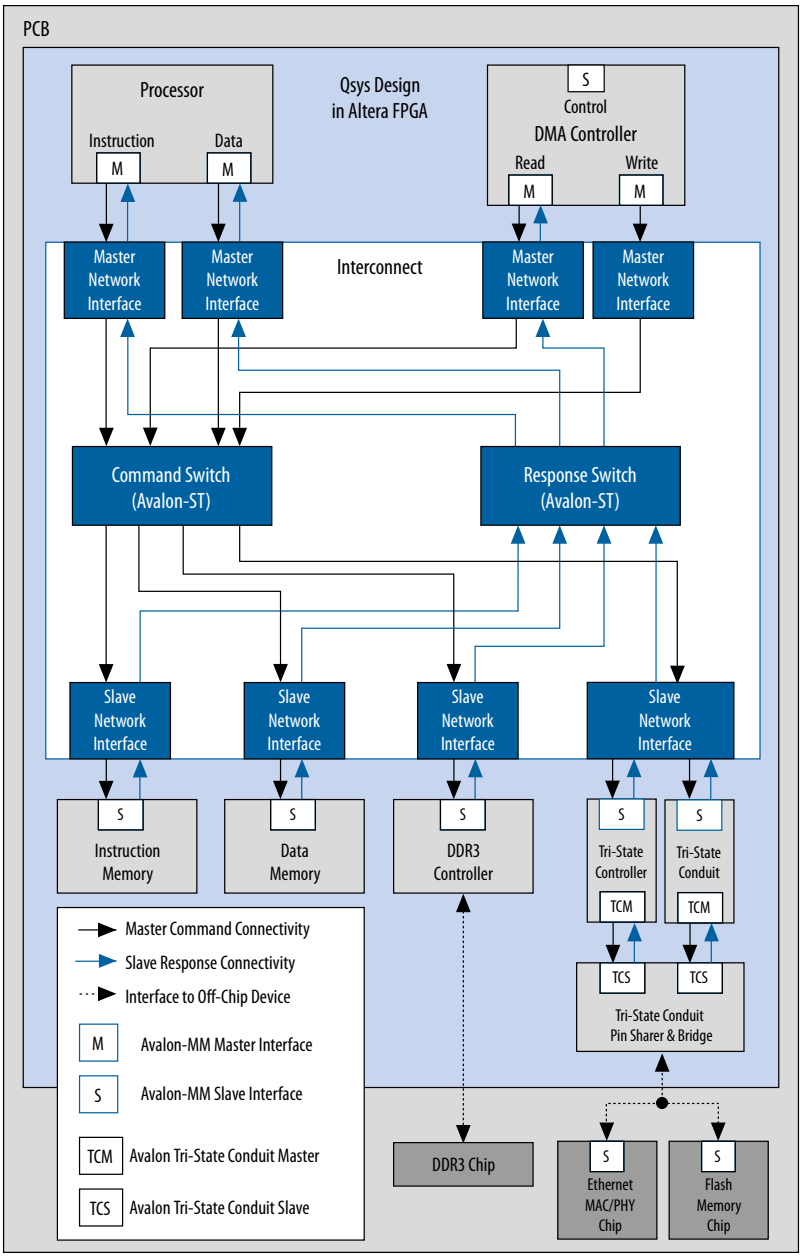
- Any number of components with master and slave interfaces. The master-to-slave relationship can be one-to-one, one-to-many, many-to-one, or many-to-many.
- Masters and slaves of different data widths.
- Masters and slaves operating in different clock domains.
- IP Components with different interface 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 mastering the system, a processor and a DMA controller, each with two master interfaces. The masters connect through the Platform Designer interconnect to slaves 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 Intel Quartus Prime software design, and show how to export component interfaces and how to connect these interfaces to external devices.

Figure 79. Platform Designer interconnect for an Avalon-MM System with Multiple Masters



3.1.1. Platform Designer Packet Format

The Platform Designer packet format supports Avalon, AXI, and APB transactions. Memory-mapped transactions between masters and slaves are encapsulated in Platform Designer packets. For Avalon systems without AXI or APB interfaces, some fields are ignored or removed.

3.1.1.1. 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 22. Platform Designer Packet Format for Memory-Mapped Master and Slave 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 master to slave. 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 masters and slaves: • **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 master or slave that initiated the command or response.
`Destination_ID`	The ID of the master or slave 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 masters, 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 masters, Platform Designer adaptation logic sets a hardwired value for the burstwrap field, according the declared master burst properties. For example, for a master that declares sequential bursting, the burstwrap field is set to ones. Similarly, masters that declare burst have their burstwrap field set to the appropriate constant value. AXI masters 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 masters.
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-MM interfaces, this field maps directly to the privileged access signal, which allows a memory-mapped master 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 master to the AXI slave. Transactions from AMBA 3 AXI and Avalon masters have the default value 4'b0000, that indicates that they are not participating in the QoS scheme. QoS values are dropped for slaves 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.

3.1.1.2. Transaction Types for Memory-Mapped Interfaces

Table 23. Transaction Types for Memory-Mapped Interfaces

The table below describes the information that each bit transports in the packet format's transaction field.

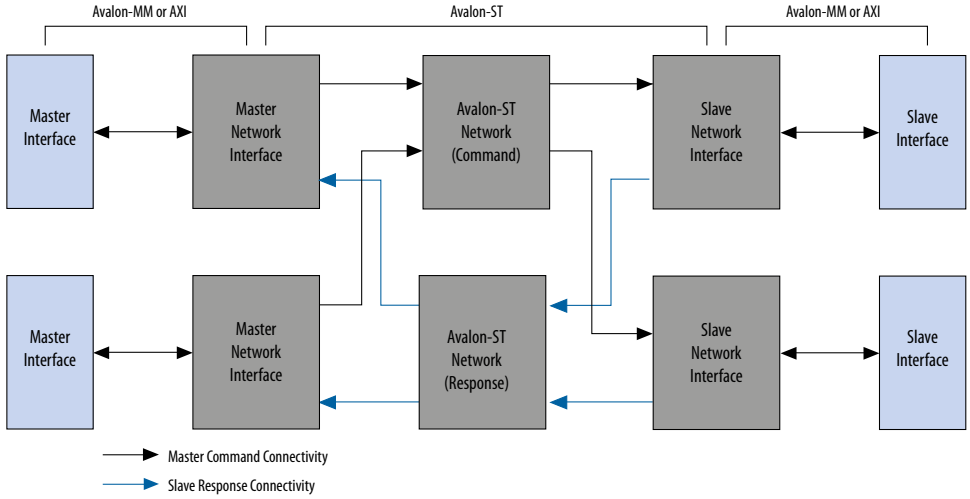
Bit	Name	Definition
0	PKT_TRANS_READ	When asserted, indicates a read transaction.
1	PKT_TRANS_COMPRESSED_READ	For read transactions, specifies whether the read command can be expressed in a single cycle (all byteenables asserted on every cycle).
2	PKT_TRANS_WRITE	When asserted, indicates a write transaction.
3	PKT_TRANS_POSTED	When asserted, no response is required.
4	PKT_TRANS_LOCK	When asserted, indicates arbitration is locked. Applies to write packets.

3.1.1.3. Platform Designer Transformations

The memory-mapped master and slave components connect to network interface modules that encapsulate the transaction in Avalon-ST 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 80. Transformation when Generating a System with Memory-Mapped and Slave Components

Platform Designer components that implement the blocks appear shaded.



Related Information

- Master Network Interfaces on page 135
- Slave Network Interfaces on page 138

3.1.2. Interconnect Domains

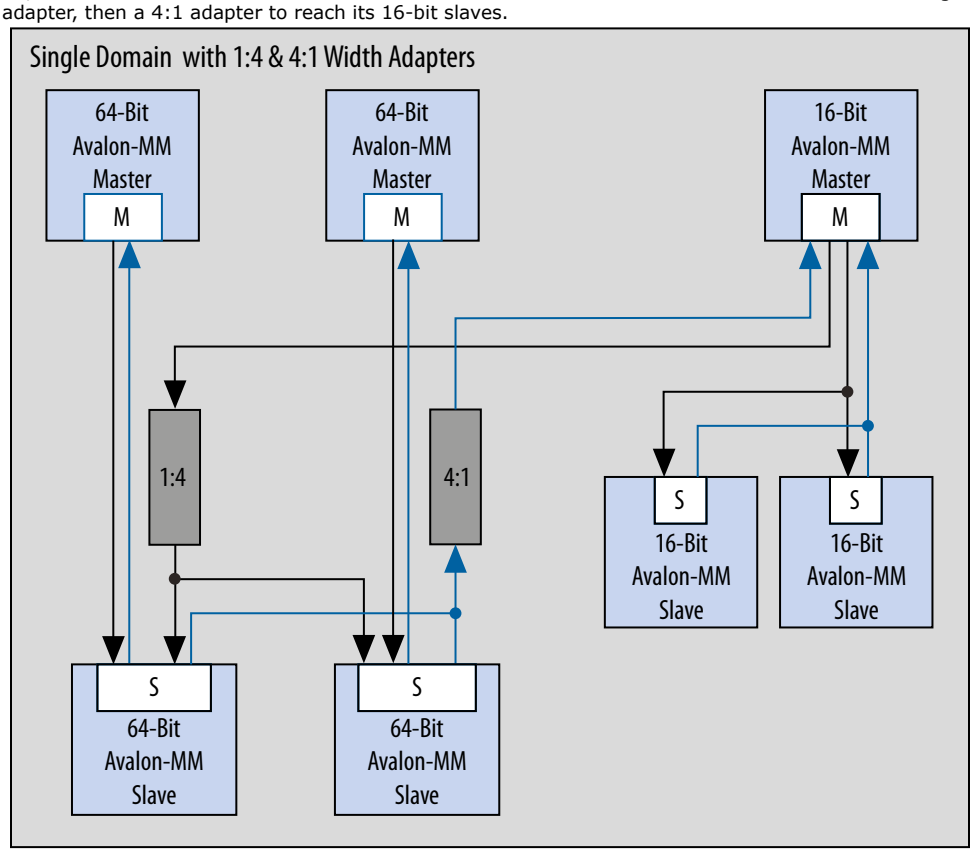
An interconnect domain is a group of connected memory-mapped masters and slaves that share the same interconnect. The components in a single interconnect domain share the same packet format.

3.1.2.1. Using One Domain with Width Adaptation

When one of the masters in a system connects to all the slaves, 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 master and 64-bit slaves.

Figure 81. One Domain with 1:4 and 4:1 Width Adapters

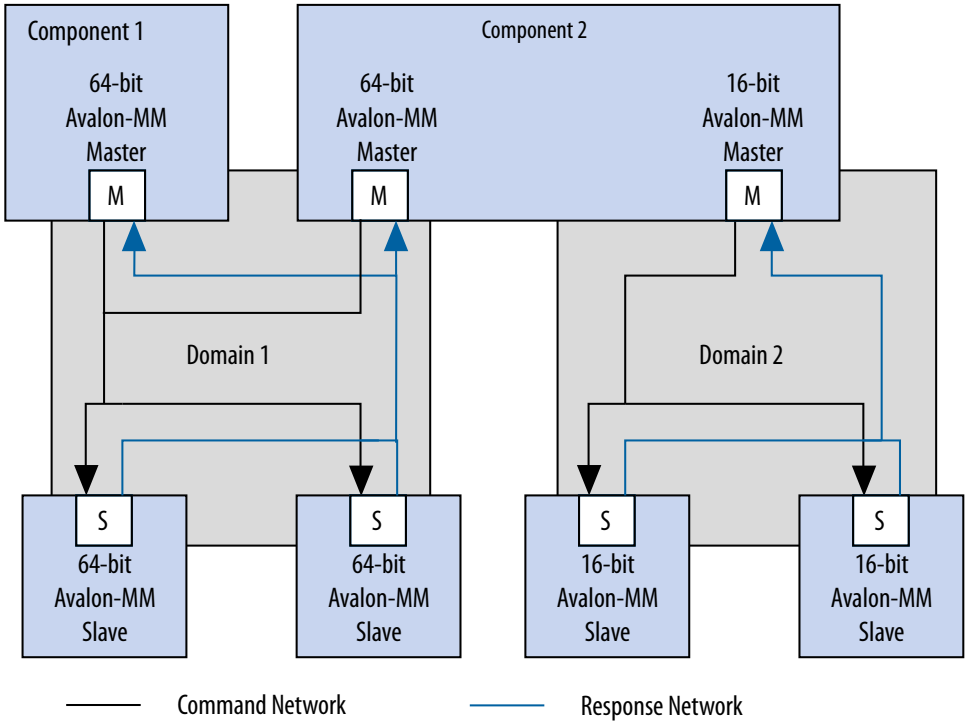
In this system example, there are two 64-bit masters that access two 64-bit slaves. It also includes one 16-bit master, that accesses two 16-bit slaves and two 64-bit slaves. The 16-bit Avalon master connects through a 1:4 adapter, then a 4:1 adapter to reach its 16-bit slaves.



3.1.2.2. Using Two Separate Domains

Figure 82. Two Separate Domains

In this system example, Platform Designer uses two separate domains. The first domain includes two 64-bit masters connected to two 64-bit slaves. A second domain includes one 16-bit master connected to two 16-bit slaves. 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.



3.1.3. Master Network Interfaces

Figure 83. Avalon-MM Master Network Interface

Avalon network interfaces drive default values for the QoS and BUSER, WUSER, and RUSER packet fields in the master agent, and drop the packet fields in the slave agent.

Note: The `response` signal from the Limiter to the Agent is optional.

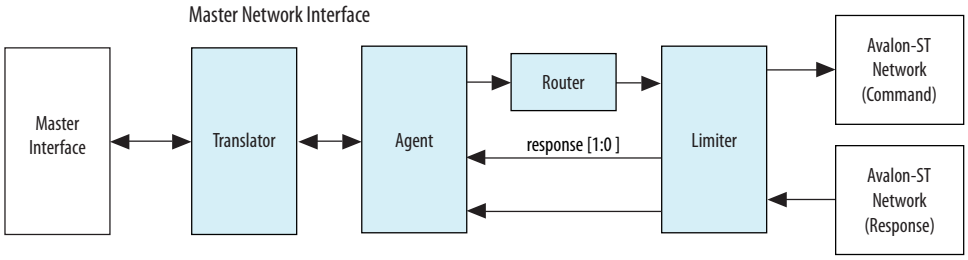
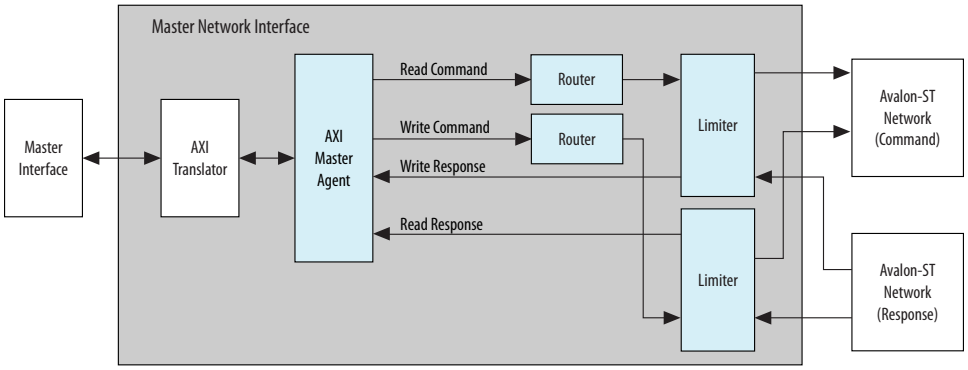


Figure 84. AXI Master Network Interface

An AMBA 4 AXI master supports INCR bursts up to 256 beats, QoS signals, and data sideband signals.



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 10

3.1.3.1. Avalon-MM Master Agent

The Avalon-MM Master Agent translates Avalon-MM master transactions into Platform Designer command packets and translates the Platform Designer Avalon-MM slave response packets into Avalon-MM responses.

3.1.3.2. Avalon-MM Master Translator

The Avalon-MM Master Translator interfaces with an Avalon-MM master component and converts the Avalon-MM master interface to a simpler representation for use in Platform Designer.

The Avalon-MM Master translator performs the following functions:

- Translates active-low signaling to active-high signaling
- Inserts wait states to prevent an Avalon-MM master 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

3.1.3.3. AXI Master Agent

An AXI Master 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. Avalon master 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 131

3.1.3.4. 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 an Avalon or AMBA 4 AXI slave is connected to a master without response ports, the interconnect could ignore transaction responses such as SLAVEERROR or DECODEERROR. This situation could lead to returning invalid data to the master.

The AXI translator is inserted for both AMBA 4 AXI masters and slaves and performs the following functions:

- Matches ID widths between the master and slave 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 master connects to an AMBA 4 AXI slave in 1x1 systems.

Related Information

Arm AMBA Protocol Specifications

3.1.3.5. APB Master Agent

An APB master agent accepts APB commands and produces or generates Platform Designer command packets. It also converts Platform Designer response packets to APB responses.

3.1.3.6. APB Slave Agent

An APB slave agent issues resulting transaction to the APB interface. It also accepts creates Platform Designer response packets.

3.1.3.7. 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 master and slave and performs the following functions:

- Sets the response value default to OKAY if the APB slave 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).

3.1.3.8. AHB Slave Agent

The Platform Designer interconnect supports non-bursting Advanced High-performance Bus (AHB) slave interfaces.

3.1.3.9. Memory-Mapped Router

The Memory-Mapped Router routes command packets from the master to the slave, and response packets from the slave to the master. For master command packets, the router uses the address to set the `Destination_ID` and Avalon-ST channel. For the slave response packet, the router uses the `Destination_ID` to set the Avalon-ST channel. The demultiplexers use the Avalon-ST channel to route the packet to the correct destination.

3.1.3.10. Memory-Mapped Traffic Limiter

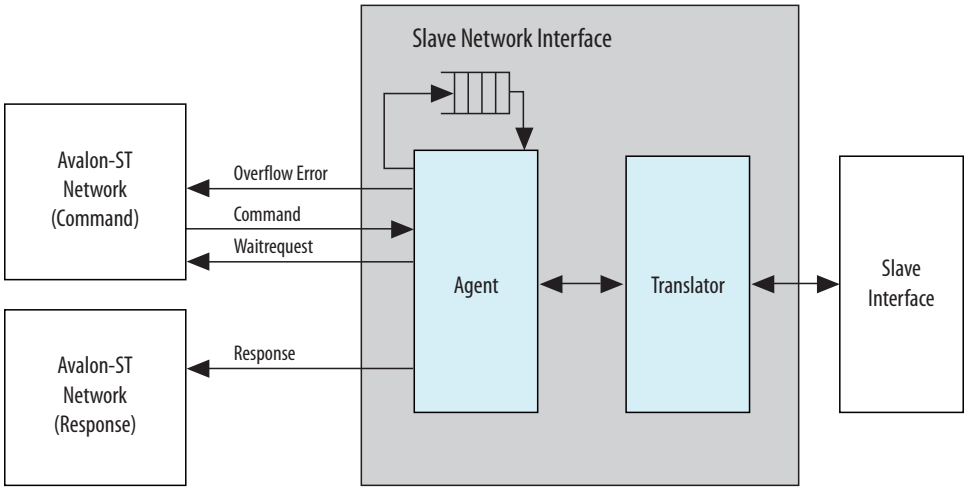
The Memory-Mapped Traffic Limiter ensures the responses arrive in order. It prevents any command from being sent if the response could conflict with the response for a command that has already been issued. By guaranteeing in-order responses, the Traffic Limiter simplifies the response network.

3.1.4. Slave Network Interfaces

3.1.4.1. Avalon-MM Slave Translator

The Avalon-MM Slave Translator converts the Avalon-MM slave interface to a simplified representation that the Platform Designer network can use.

Figure 85. Avalon-MM Slave Network Interface



An Avalon-MM Slave Translator performs the following functions:

- Drives the `beginbursttransfer` and `byteenable` signals.
- Supports Avalon-MM slaves that operate using fixed timing and or slaves that use the `readdatavalid` signal to identify valid data.
- Translates the `read`, `write`, and `chipselect` signals into the representation that the Avalon-ST slave 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

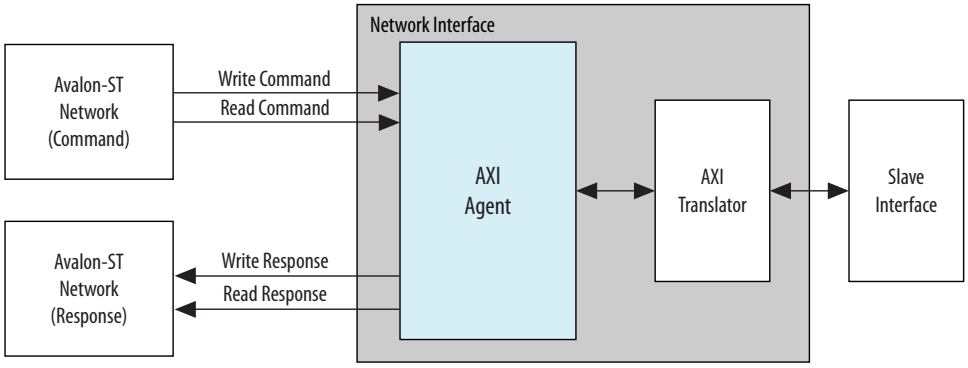
Slave Network Interfaces on page 138

3.1.4.2. 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 86. AXI Slave Network Interface

An AMBA 4 AXI slave supports up to 256 beat `INCR` bursts, QoS signals, and data sideband signals.



The AXI translator is inserted for both AMBA 4 AXI master and slave, and performs the following functions:

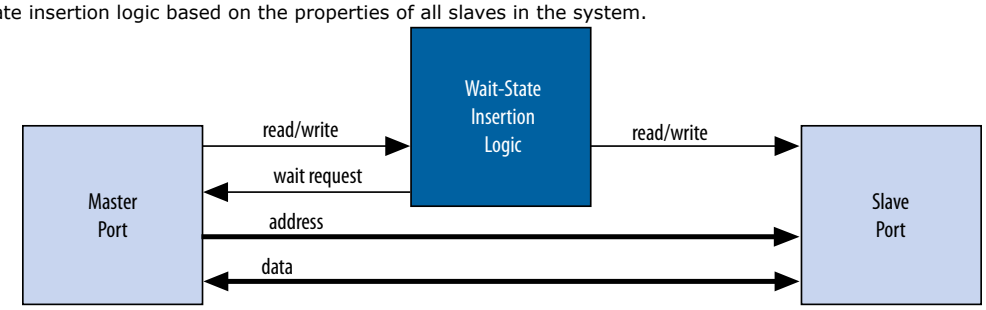
- Matches ID widths between master and slave in 1x1 systems.
- Drives default values as defined in the *AMBA Protocol Specifications* for missing signals.
- Performs lock transaction bit conversion when an AMBA 3 AXI master connects to an AMBA 4 AXI slave in 1x1 systems.

3.1.4.3. 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 slave, and causes the master to wait until the slave can proceed. Platform Designer interconnect inserts wait states into a transfer when the target slave cannot respond in a single clock cycle, as well as in cases when slave `read` and `write` signals have setup or hold time requirements.

Figure 87. Wait State Insertion Logic for One Master and One Slave

Wait state insertion logic is a small finite-state machine that translates control signal sequencing between the slave side and the master side. Platform Designer interconnect can force a master to wait for the wait state needs of a slave; for example, arbitration logic in a multi-master system. Platform Designer generates wait state insertion logic based on the properties of all slaves in the system.



3.1.4.4. Avalon-MM Slave Agent

The Avalon-MM Slave Agent accepts command packets and issues the resulting transactions to the Avalon interface. For pipelined slaves, an Avalon-ST FIFO stores information about pending transactions. The size of this FIFO is the maximum number of pending responses that you specify when creating the slave component. The Avalon-MM Slave Agent also `backpressures` the Avalon-MM master command interface when the FIFO is full if the slave component includes the `waitrequest` signal.

3.1.4.5. AXI Slave Agent

An AXI Slave Agent works like a reverse master agent. The AXI Slave 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.

3.1.5. Arbitration

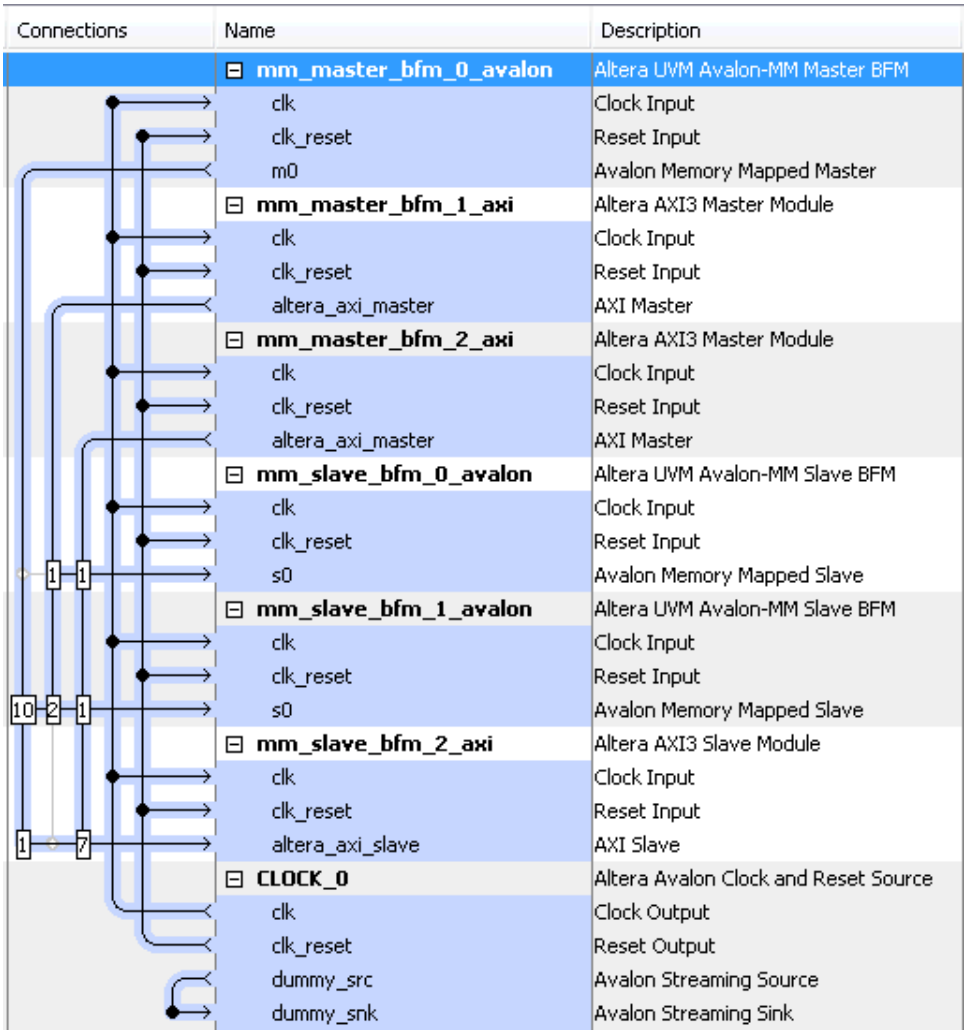
When multiple masters contend for access to a slave, Platform Designer automatically inserts arbitration logic, which grants access in fairness-based, round-robin order. You can alternatively choose to designate a slave as a fixed priority arbitration slave, and then manually assign priorities in the Platform Designer GUI.

3.1.5.1. Round-Robin Arbitration

When multiple masters contend for access to a slave, Platform Designer automatically inserts arbitration logic which grants access in fairness-based, round-robin order.

In a fairness-based arbitration protocol, each master has an integer value of transfer *shares* with respect to a slave. 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 masters. You can change the arbitration scheme to weighted round-robin by specifying a relative number of arbitration shares to the masters that access a given slave. AXI slaves 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 88. Arbitration Shares in the Connections Column



3.1.5.1.1. Fairness-Based Shares

In a fairness-based arbitration scheme, each master-to-slave connection provides a transfer share count. This count is a request for the arbiter to grant a specific number of transfers to this master before giving control to a different master. One share represents permission to perform one transfer.

Figure 89. Arbitration of Continuous Transfer Requests from Two Masters

Consider a system with two masters connected to a single slave. Master 1 has its arbitration shares set to three, and Master 2 has its arbitration shares set to four. Master 1 and Master 2 continuously attempt to perform back-to-back transfers to the slave. The arbiter grants Master 1 access to the slave for three transfers, and then grants Master 2 access to the slave for four transfers. This cycle repeats indefinitely. The figure below describes the waveform for this scenario.

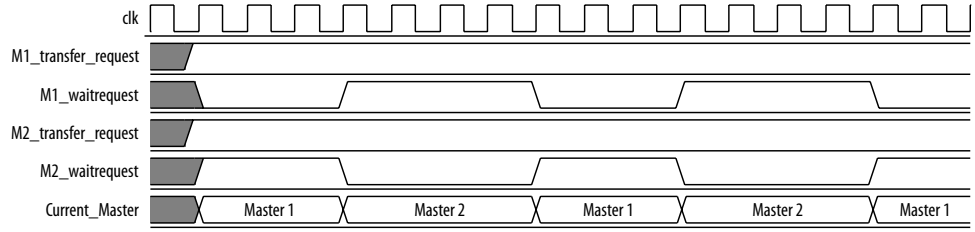
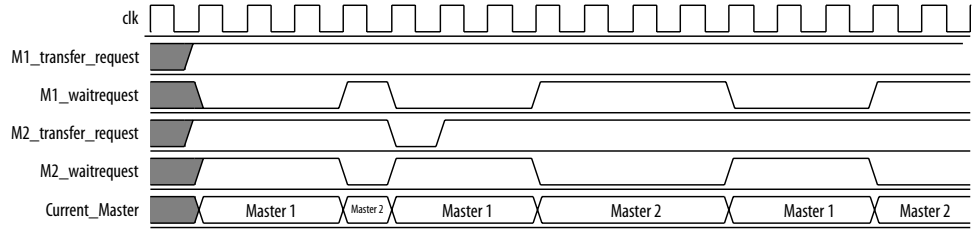


Figure 90. Arbitration of Two Masters with a Gap in Transfer Requests

If a master stops requesting transfers before it exhausts its shares, it forfeits all its remaining shares, and the arbiter grants access to another requesting master. After completing one transfer, Master 2 stops requesting for one clock cycle. As a result, the arbiter grants access back to Master 1, which gets a replenished supply of shares.



3.1.5.1.2. Round-Robin Scheduling

When multiple masters contend for access to a slave, the arbiter grants shares in round-robin order. Platform Designer includes only requesting masters in the arbitration for each slave transaction.

3.1.5.2. 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 slave in a Platform Designer system. You can design Platform Designer systems where a subset of slaves use the default round-robin arbitration, and other slaves use fixed priority arbitration. Fixed priority arbitration uses a fixed priority algorithm to grant access to a slave amongst its connected masters.

To set up fixed priority arbitration, you must first designate a fixed priority slave in your Platform Designer system in the **Interconnect Requirements** tab. You can then assign an arbitration priority number for each master connected to a fixed priority slave in the **System View** tab, where the highest numeric value receives the highest priority. When multiple masters request access to a fixed priority arbitrated slave, the arbiter gives the master with the highest priority first access to the slave.

For example, when a fixed priority slave receives requests from three masters on the same cycle, the arbiter grants the master with highest assigned priority first access to the slave, and backpressures the other two masters.

Note: When you connect an AXI master to an Avalon-MM slave designated to use a fixed priority arbitrator, the interconnect instantiates a command-path intermediary round-robin multiplexer in front of the designated slave.

3.1.5.2.1. Designate a Platform Designer Slave to Use Fixed Priority Arbitration

You can designate any slave in your Platform Designer system to use fixed priority arbitration. You must assign each master connected to a fixed priority slave a numeric priority. The master with the highest higher priority receives first access to the slave. No two masters can have the same priority.

1. In Platform Designer, navigate to the **Interconnect Requirements** tab.
2. Click **Add** to add a new requirement.
3. In the **Identifier** column, select the slave for fixed priority arbitration.
4. In the **Setting** column, select **qsys mm.arbitrationScheme**.
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 slave, and then select **Show Arbitration Shares**.
8. For each master connected to the fixed priory arbitration slave, type a numerical arbitration priority in the box that appears in place of the connection circle.
9. Right click the designated fixed priority slave and uncheck **Show Arbitration Shares** to return to the connection circles.

3.1.5.2.2. Fixed Priority Arbitration with AXI Masters and Avalon-MM Slaves

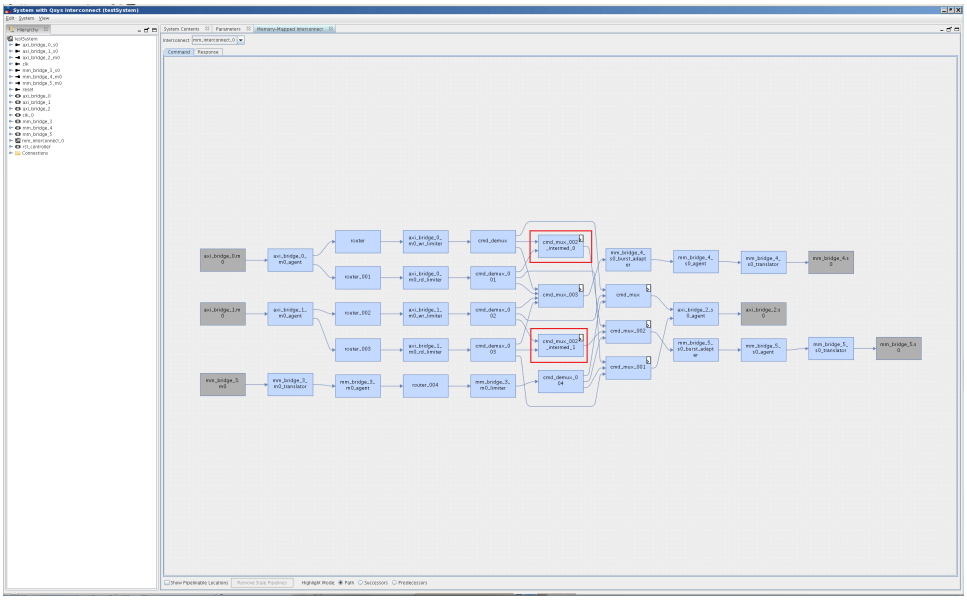
When an AXI master is connected to a designated fixed priority arbitration Avalon-MM slave, Platform Designer interconnect automatically instantiates an intermediary multiplexer in front of the Avalon-MM slave.

Since AXI masters have separate read and write channels, each channel appears as two separate masters to the Avalon-MM slave. To support fairness between the AXI master's read and write channels, the instantiated round-robin intermediary multiplexer arbitrates between simultaneous read and write commands from the AXI master to the fixed-priority Avalon-MM slave.

When an AXI master is connected to a fixed priority AXI slave, the master's read and write channels are directly connected to the AXI slave'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 circles indicate placement of the intermediary multiplexer between the AXI master and Avalon-MM slave due to the separate read and write channels of the AXI master.

Figure 91. Intermediary Multiplexer Between AXI Master and Avalon-MM Slave

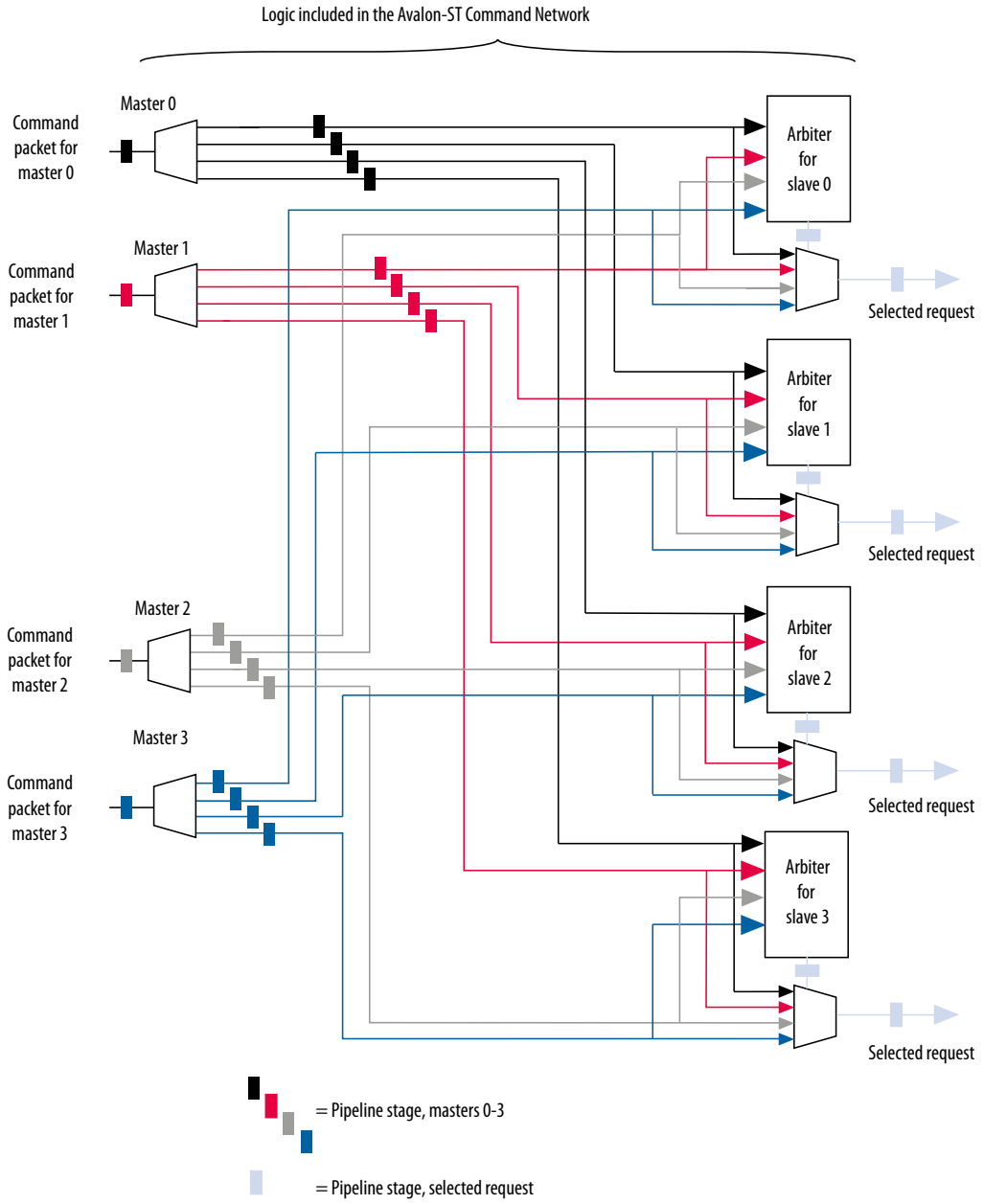


3.1.6. Memory-Mapped Arbiter

The input to the Memory-Mapped Arbiter is the command packet for all masters requesting access to a specific slave. The arbiter outputs the channel number for the selected master. This channel number controls the output of a multiplexer that selects the slave device.

Figure 92. Arbitration Logic

In this example, four Avalon-MM masters connect to four Avalon-MM slaves. In each cycle, an arbiter positioned in front of each Avalon-MM slave selects among the requesting Avalon-MM masters.



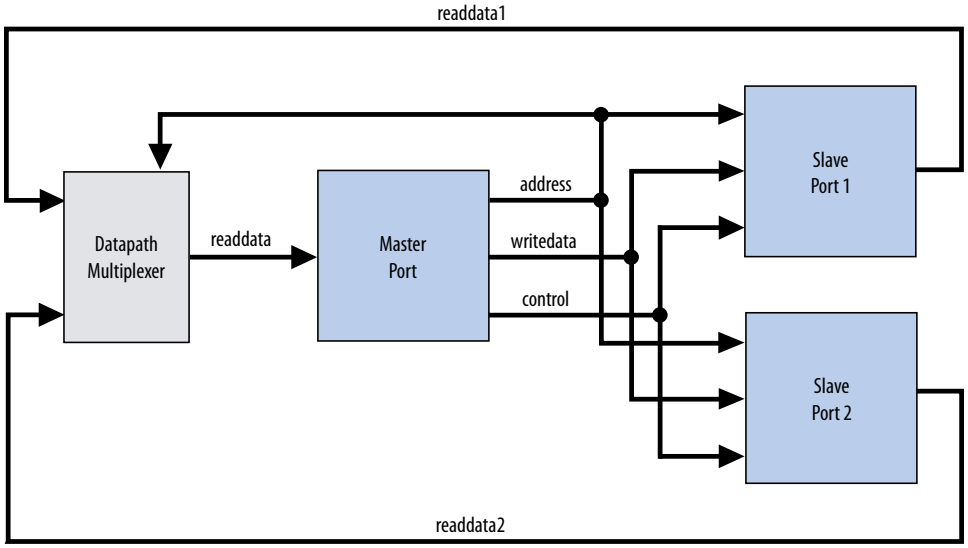
Note: If you specify a **Limit interconnect pipeline stages to**parameter greater than zero, the output of the Arbiter is registered. Registering this output reduces the amount of combinational logic between the master and the interconnect, increasing the f_{MAX} of the system.

Note: You can use the Memory-Mapped Arbiter for both round-robin and fixed priority arbitration.

3.1.7. Datapath Multiplexing Logic

Datapath multiplexing logic drives the `writedata` signal from the granted master to the selected slave, and the `readdata` signal from the selected slave back to the requesting master. Platform Designer generates separate datapath multiplexing logic for every master in the system (`readdata`), and for every slave in the system (`writedata`). Platform Designer does not generate multiplexing logic if it is not needed.

Figure 93. Datapath Multiplexing Logic for One Master and Two Slaves



3.1.8. Width Adaptation

Platform Designer width adaptation converts between Avalon memory-mapped master and slaves 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.

3.1.8.1. Memory-Mapped Width Adapter

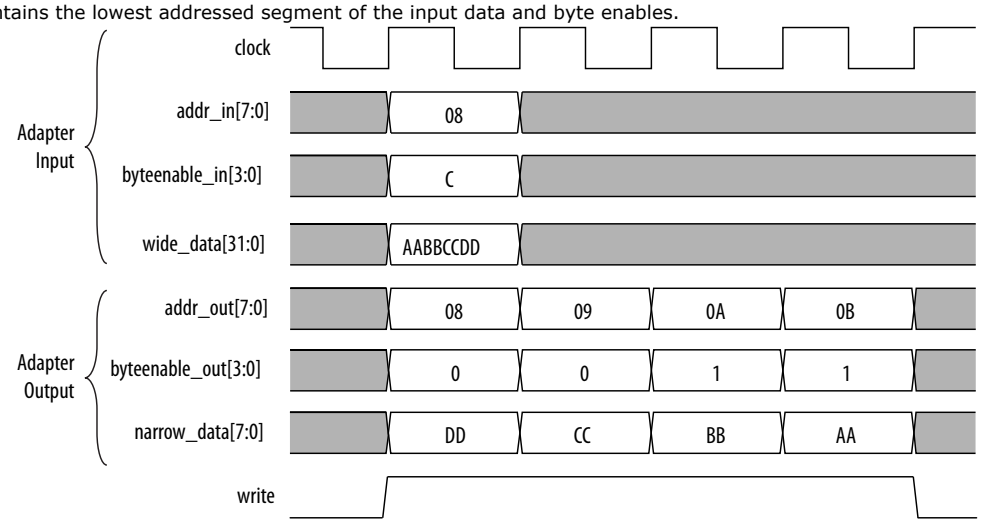
The Memory-Mapped Width Adapter is used in the Avalon-ST domain and operates with information contained in the packet format.

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 94. 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.



3.1.8.1.1. 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 24. 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 slave, 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 slave can accept is 16, which is the case for AMBA 3 AXI slaves. Avalon slaves 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 slave, 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.

3.1.8.1.2. AXI Narrow-to-Wide Adaptation

Table 25. 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.

3.1.9. 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 master may be capable of initiating a burst longer than a slave's maximum supported burst length. In this case, the burst adapter translates the large master burst into smaller bursts, or into individual slave transfers if the slave does not support bursting. Until the master completes the burst, arbiter logic prevents other masters from accessing the target slave. For example, if a master initiates a burst of 16 transfers to a slave with maximum burst length of 8, the burst adapter initiates 2 bursts of length 8 to the slave.

Avalon-MM and AXI burst transactions allow a master uninterrupted access to a slave for a specified number of transfers. The master specifies the number of transfers when it initiates the burst. Once a burst begins between a master and slave, arbiter logic is locked until the burst completes. For burst masters, the length of the burst is the number of cycles that the master has access to the slave, and the selected arbitration shares have no effect.

Note: AXI masters 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 slaves, 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 slaves.

Avalon-MM masters 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 slave 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 masters. Some bursting masters 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.

3.1.9.1. Burst Adapter Implementation Options

Platform Designer automatically inserts burst adapters into your system depending on your master and slave 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 masters or slaves 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 10

3.1.9.2. Burst Adaptation: AXI to Avalon

Table 26. Burst Adaptation: AXI to Avalon

Entries specify the behavior when converting between AXI and Avalon burst types.

Burst Type	Behavior
Incrementing	**Sequential Slave** 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 slave with a maximum burst length of 4, an `INCR7` burst is converted to `INCR4 + INCR3`. **Wrapping Slave** 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 slave's wrapping boundary.
Wrapping	**Sequential Slave** 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 Slave** If the WRAP transaction's boundary matches the slave's boundary, then the burst passes through. Otherwise, the burst is converted to sequential bursts that respect both the transaction and slave 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.

3.1.9.3. Burst Adaptation: Avalon to AXI

Table 27. Burst Adaptation: Avalon to AXI

Entries specify the behavior when converting between Avalon and AXI burst types.

Burst Type	Definition
Sequential	Bursts of length greater than16 are converted to multiple `INCR` bursts of a length less than or equal to16. Bursts of length less than or equal to16 are not converted.
Wrapping	Only Avalon masters with `alwaysBurstMaxBurst = true` are supported. The WRAP burst is passed through if the length is less than or equal to16. 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}.

3.1.10. Waitrequest Allowance Adapter

The Waitrequest Allowance Adapter allows a connection between a master and a slave 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 master and slave 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 master and slave with different `waitrequestAllowance` property are connected.

When the slave has a `waitrequestAllowance = n` the master must deassert `read` or `write` signals after <*n*> transfers when `waitrequest` is asserted.

Table 28. Interconnect Scenarios Requiring waitrequestAllowance

Master *(m)* / Slave *(n)* waitrequestAllowance	Adaptation Required	Description	Adapter Function
m = *n*	No	The master `waitrequestAllowance` is equal to the slave's `waitrequestAllowance`.	All signals are passed through.
m = 0; *n* > 0	Yes	The master cannot send when `waitrequest=1`, but holds the value on the bus. This would result in the slave receiving multiple copies. Requires adaptation to prevent.	The adapter deasserts `valid` when input `waitrequest` is asserted.
m < *n*; *m* != 0	No	The master can send <*m*> transfers after `waitrequest` is asserted. The slave receives fewer than <*n*> transfers, which is acceptable.	All signals are passed through.
m > *n*; *n* = 0	Yes	The slave cannot accept transfers when `waitrequest` is asserted. Transfers sent when `waitrequest=1` can be lost.	If the input `waitrequest` is asserted, the adapter buffers the input data.

continued...

Master *(m)* / Slave *(n)* waitrequestAllowance	Adaptation Required	Description	Adapter Function
		Prevention requires adaptation in the form of transfer buffering.	
m > *n*; *n* > 0	Yes	The slave cannot accept more than <*n*> transfers after `waitrequest` is asserted, however the master 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.

3.1.11. 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 master with write responses and a slave without write responses can be costly, especially if there are multiple slaves, or if the slave supports bursts. To minimize the cost of logic between slaves, consider placing the slaves 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 slave interface.

The following table describes what happens when there is a mismatch in response support between the master and slave.

Table 29. Response Support for Mismatched Master and Slave

	Slave with Response	Slave Without Response
Master with Response	Interconnect delivers response from the slave to the master. Response merging or duplication may be necessary for bus sizing.	Interconnect delivers an OKAY response to the master
Master without Response	Master ignores responses from the slave	No need for responses. Master, slave and interconnect operate without response support.

Note: If there is a bridge between the master and the endpoint slave, and the responses must come from the endpoint slave, ensure that the bridge passes the appropriate response signals through from the endpoint slave to the master.

If the bridge does not support responses, then the responses are generated by the interconnect at the slave interface of the bridge, and responses from the endpoint slave 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 slave. For information about Platform Designer system security, refer to Manage System Security. For information about specifying a default slave, refer to *Error Response Slave* in *Platform Designer System Design Components*.

Note: Avalon-MM slaves without a `response` signal are not able to notify a connected master that a transaction has not completed successfully. As a result, Platform Designer interconnect generates an OKAY response on behalf of the Avalon-MM slave.

Related Information

- Master Network Interfaces on page 135
- Error Response Slave on page 237
- Error Correction Coding (ECC) in Platform Designer Interconnect on page 188

3.1.12. Platform Designer Address Decoding

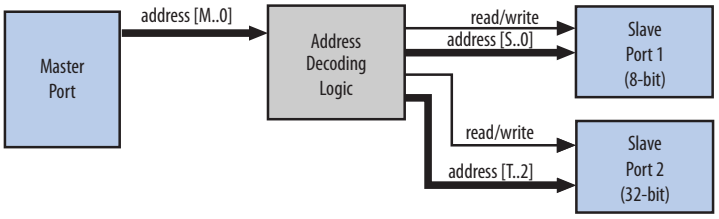
Address decoding logic forwards appropriate addresses to each slave.

Address decoding logic simplifies component design in the following ways:

- The interconnect selects a slave whenever it is being addressed by a master. Slave components do not need to decode the address to determine when they are selected.
- Slave addresses are properly aligned to the slave interface.
- Changing the system memory map does not involve manually editing HDL.

Figure 95. Address Decoding for One Master and Two Slaves

In this example, Platform Designer generates separate address decoding logic for each master in a system. The address decoding logic processes the difference between the master address width (<M>) and the individual slave address widths (<S>) and (<T>). The address decoding logic also maps only the necessary master address bits to access words in each slave'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 a slave 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}.

3.2. Avalon Streaming Interfaces

High bandwidth components with streaming data typically use Avalon-ST 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-ST interconnect always creates a point-to-point connection between a single data source and data sink.

Figure 96. Memory-Mapped and Avalon-ST Interfaces

In this example, there are 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**.

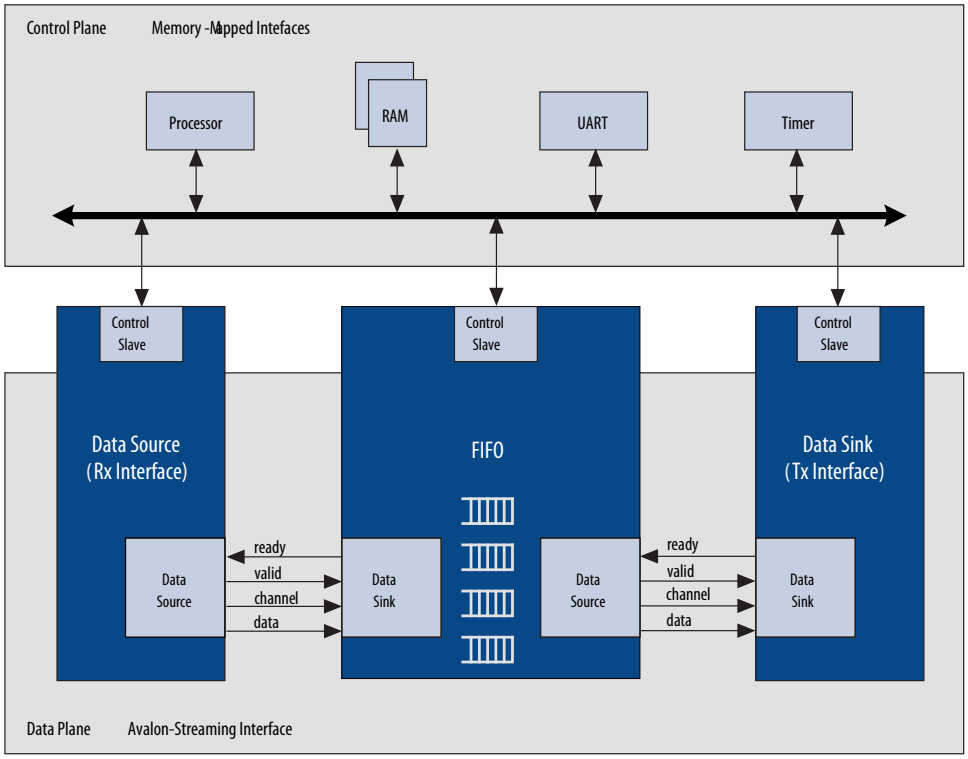


Figure 97. Avalon-ST Connection Between the Source and Sink

This source-sink pair includes only the `data` signal. The sink must be able to receive data as soon as the source interface comes out of reset.

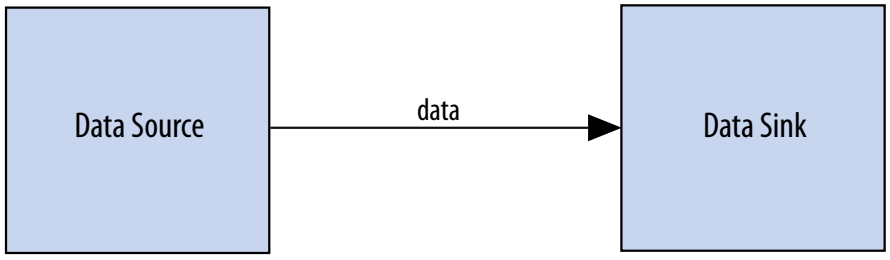
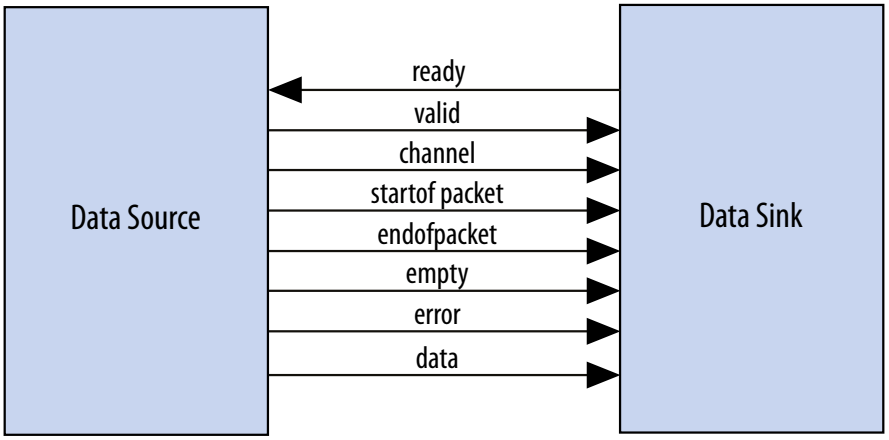


Figure 98. Signals Indicating the Start and End of Packets, Channel Numbers, Error Conditions, and Backpressure

All data transfers using Avalon-ST 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-ST components that you can use to create datapaths, including datapaths whose input and output streams have different properties. Generated systems that include memory-mapped master and slave components may also use these Avalon-ST 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-ST components.

Related Information

Platform Designer Transformations on page 133

3.2.1. Avalon-ST Adapters

Platform Designer automatically adds Avalon-ST adapters between two components during system generation when it detects mismatched interfaces. If you connect mismatched Avalon-ST 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-ST 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 core connections.

3.2.1.1. Avalon-ST Adapter

The Avalon-ST adapter combines the logic of the channel, error, data format, and timing adapters. The Avalon-ST adapter provides adaptations between interfaces that have mismatched Avalon-ST endpoints. Based on the source and sink interface parameterizations for the Avalon-ST adapter, Platform Designer instantiates the necessary adapter logic (channel, error, data format, or timing) as hierarchal sub-components.

3.2.1.1.1. Avalon-ST Adapter Parameters Common to Source and Sink Interfaces

Table 30. Avalon-ST 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.

3.2.1.1.2. Avalon-ST Adapter Upstream Source Interface Parameters

Table 31. Avalon-ST 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.

3.2.1.1.3. Avalon-ST Adapter Downstream Sink Interface Parameters

Table 32. Avalon-ST 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.

3.2.1.2. Channel Adapter

The channel adapter provides adaptations between interfaces that have different channel signal widths.

Table 33. 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.

3.2.1.2.1. Avalon-ST Channel Adapter Input Interface Parameters

Table 34. Avalon-ST 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.

3.2.1.2.2. Avalon-ST Channel Adapter Output Interface Parameters

Table 35. Avalon-ST 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.

3.2.1.2.3. Avalon-ST Channel Adapter Common to Input and Output Interface Parameters

Table 36. Avalon-ST 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-ST 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.

3.2.1.3. 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 37. Data Format Adapter Adaptations

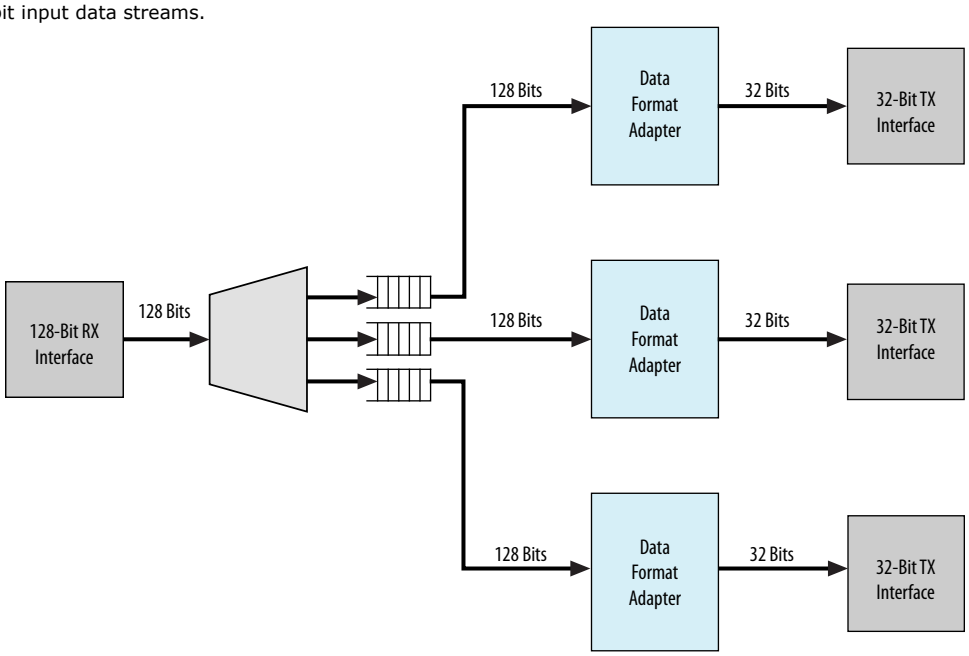
Condition	Description of Adapter Logic
The source and sink's bits per symbol parameters are different.	The connection cannot be made.
The source and sink have a different number of symbols per beat.	The adapter converts the source's width to the sink's width.

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 99. 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.



3.2.1.3.1. Avalon-ST Data Format Adapter Input Interface Parameters

Table 38. Avalon-ST 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.

3.2.1.3.2. Avalon-ST Data Format Adapter Output Interface Parameters

Table 39. Avalon-ST 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.

3.2.1.3.3. Avalon-ST Data Format Adapter Common to Input and Output Interface Parameters

Table 40. Avalon-ST 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-ST 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.

3.2.1.4. 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.

3.2.1.4.1. Avalon-ST Error Adapter Input Interface Parameters

Table 41. Avalon-ST 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.

3.2.1.4.2. Avalon-ST Error Adapter Output Interface Parameters

Table 42. Avalon-ST 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.

3.2.1.4.3. Avalon-ST Error Adapter Common to Input and Output Interface Parameters

Table 43. Avalon-ST 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.

3.2.1.5. 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 44. 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.

3.2.1.5.1. Avalon-ST Timing Adapter Input Interface Parameters

Table 45. Avalon-ST 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.

3.2.1.5.2. Avalon-ST Timing Adapter Output Interface Parameters

Table 46. Avalon-ST 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.

3.2.1.5.3. Avalon-ST Timing Adapter Common to Input and Output Interface Parameters

Table 47. Avalon-ST 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.
Data Symbols Per Beat	Number of symbols per active transfer.

continued...

Parameter Name	Description
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.

3.3. Interrupt Interfaces

Using individual requests, the interrupt logic can process up to 32 IRQ inputs connected to each interrupt receiver. With this logic, the interrupt sender connected to interrupt `receiver_0` 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*.

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 adaption on interrupts, Platform Designer inserts a synchronizer, which 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.

Related Information

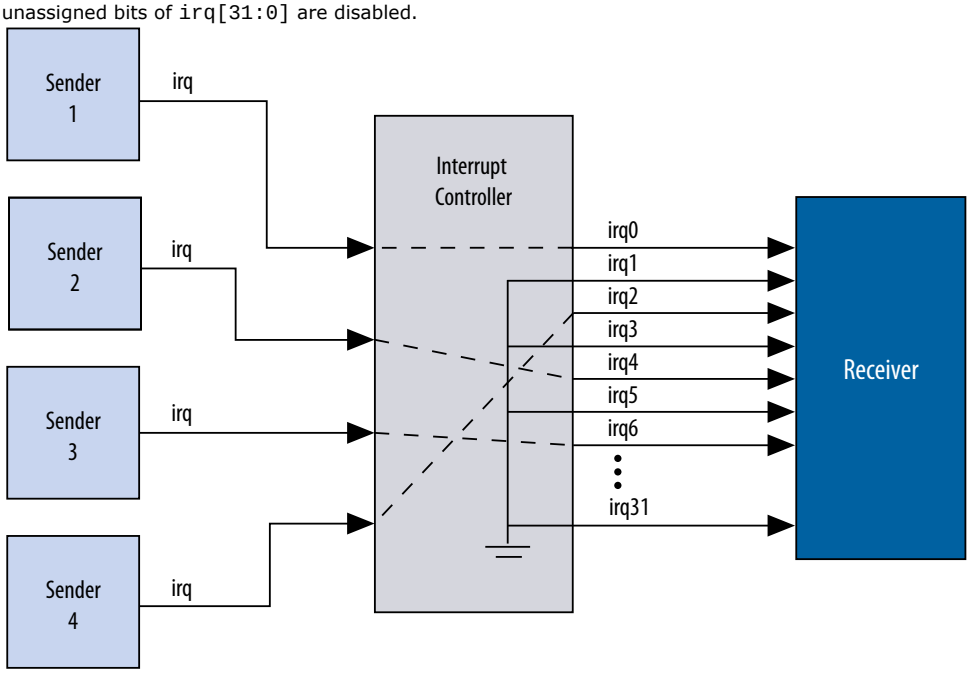
IRQ Mapper on page 164

3.3.1. 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 100. Interrupt Controller Mapping IRQs

Using individual requests, the interrupt controller can process up to 32 IRQ inputs. The interrupt controller generates a 32-bit signal `irq[31:0]` to the receiver, and maps slave IRQ signals to the bits of `irq[31:0]`. Any unassigned bits of `irq[31:0]` are disabled.



3.3.2. Assigning IRQs in Platform Designer

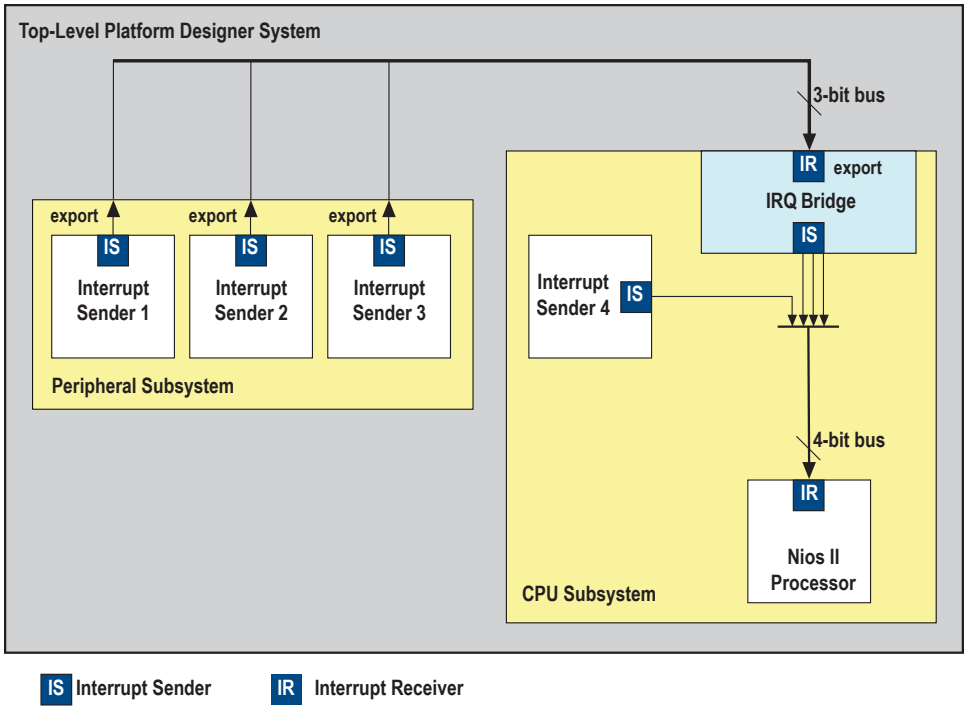
You assign IRQ connections on the **System View** tab of Platform Designer. After adding all components to the system, you connect interrupt senders and receivers. You can use the **IRQ** column to specify an IRQ number with respect to each receiver, or to specify a receiver's IRQ as unconnected. Platform Designer uses the following three components to implement interrupt handling: IRQ Bridge, IRQ Mapper, and IRQ Clock Crosser.

3.3.2.1. IRQ Bridge

The IRQ Bridge allows you to route interrupt wires between Platform Designer subsystems.

Figure 101. Platform Designer IRQ Bridge Application

The peripheral subsystem example below has three interrupt senders that are exported to the to- 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. Receivers 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.

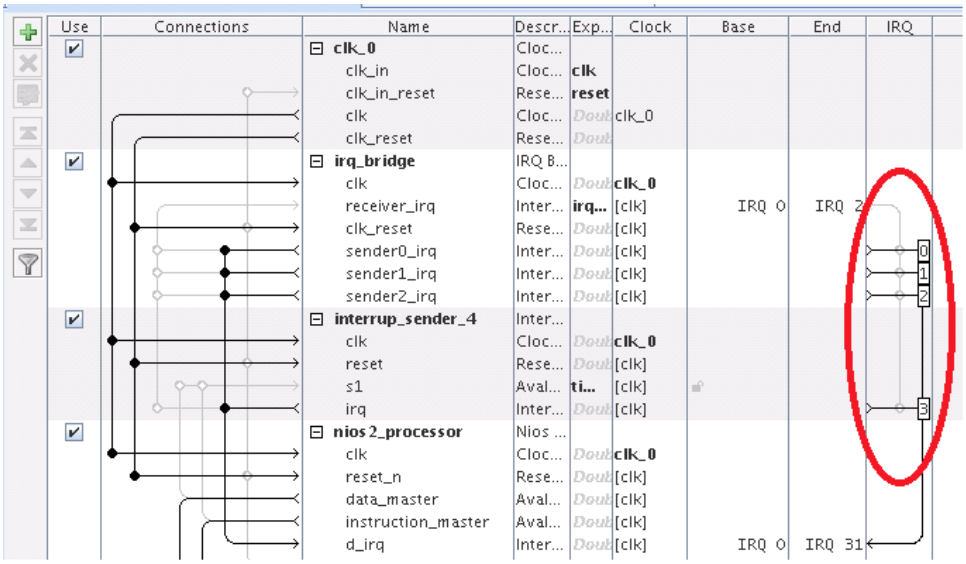
3.3.2.2. IRQ Mapper

Platform Designer inserts the IRQ Mapper automatically during generation. The IRQ Mapper converts individual interrupt wires to a bus, and then maps the appropriate IRQ priority number onto the bus.

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 102. IRQ Column in Platform Designer

Circled in the **IRQ** column are the default interrupt priorities allocated for the CPU subsystem.



Related Information

IRQ Bridge on page 163

3.3.2.3. IRQ Clock Crosser

The IRQ Clock Crosser 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.

3.4. 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.

  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.
- **Reset synchronous edges**
  - **None**—The reset is asserted and deasserted asynchronously. You can use this setting if you have internal synchronization circuitry that matches the reset required for the IP in the system.
  - **Both**—The reset is asserted and deasserted synchronously.
  - **Deassert**—The reset is deasserted synchronously and asserted asynchronously.

For more information about synchronous design practices, refer to *Recommended Design Practices*

Related Information

Recommended Design Practices

3.4.1. (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.

3.4.1.1. 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.

3.4.1.1.1. 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 48. 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 49. 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.

3.4.1.1.2. 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 50. 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 51. 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.

3.4.1.1.3. 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.

3.4.1.1.4. HSSI Serial Clock Example

Example 5. 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 6. 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
```

3.4.1.2. 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.

3.4.1.2.1. 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 52. 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 53. 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.
serializatio n	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.

3.4.1.2.2. 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 **serialzationFactor**. **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 54. 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 55. 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.

3.4.1.2.3. 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.

3.4.1.2.4. HSSI Bonded Clock Example

Example 7. 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 8. 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
```

3.5. 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 selecting **Create Global Reset Network** on the System menu. 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.

Note: If you design your own reset circuitry, you must carefully consider situations which may result in system lockup. For example, if an Avalon-MM slave is reset in the middle of a transaction, the Avalon-MM master may lockup.

Related Information

Specifying Interconnect Requirements on page 39

3.5.1. 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.

3.5.2. 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

3.5.3. 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.

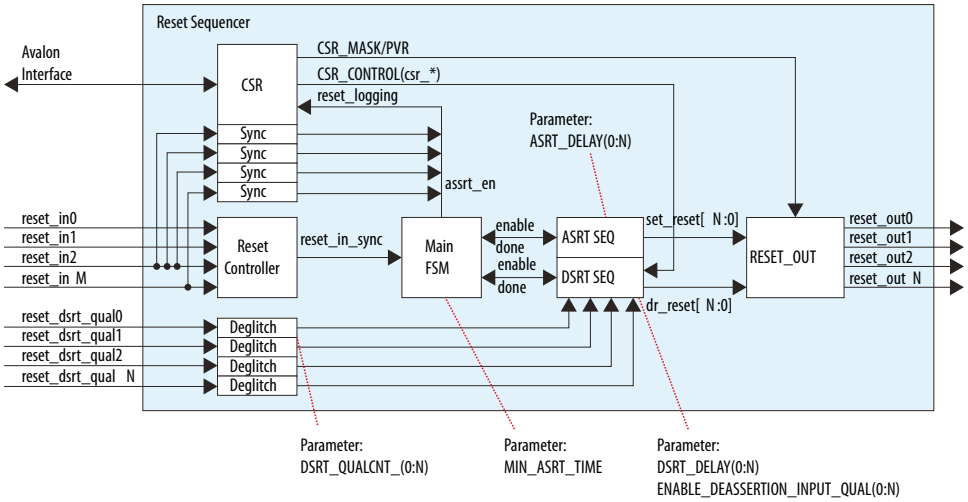
Note: Platform Designer supports multiple reset sink connections to a single reset source interface. However, there are situations in composed systems where an internally generated reset must be exported from the composed system in addition to being used to connect internal components. In this situation, you must declare one reset output interface as an export, and use another reset output to connect internal components.

3.5.4. 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 103. 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).

3.5.4.1. Reset Sequencer Parameters

Table 56. 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.

continued...

Parameter	Description
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.

3.5.4.2. Reset Sequencer Timing Diagrams

Figure 104. Basic Sequencing

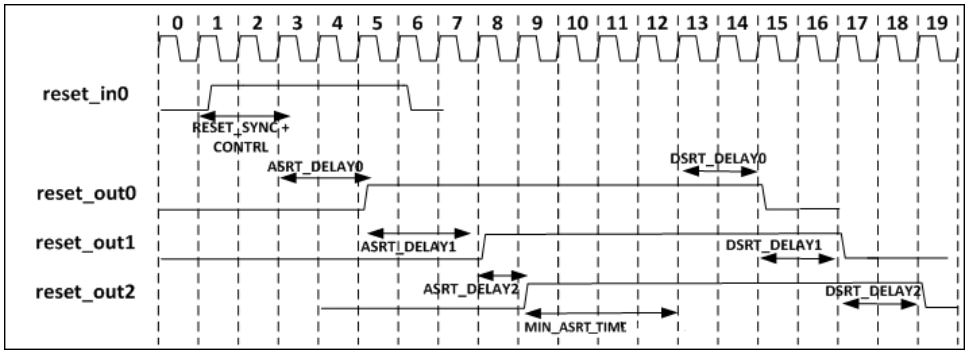
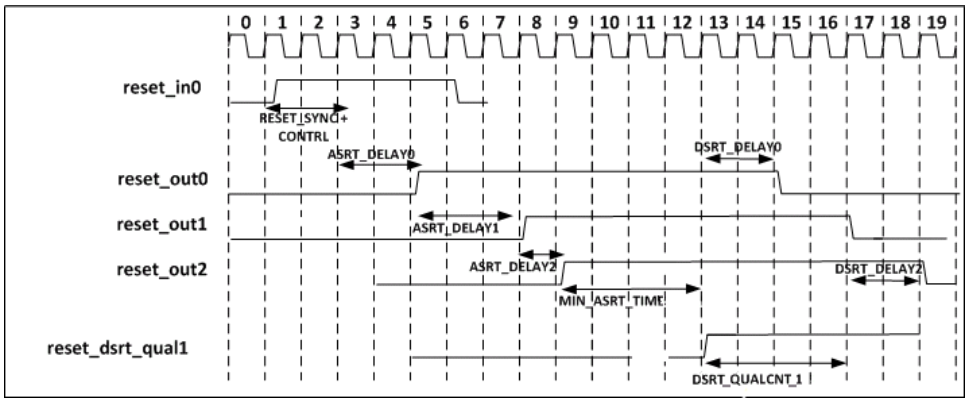


Figure 105. Sequencing with USE_DSRT_QUAL Set



3.5.4.3. 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 57. 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.
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.

3.5.4.3.1. 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 58. 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.
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.

Related Information

Reset Sequencer CSR Registers on page 176

3.5.4.3.2. Reset Sequencer Interrupt Enable Register

The `Interrupt Enable` register bits enable events triggering the IRQ of the reset sequencer.

Table 59. 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.
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.

Related Information

Reset Sequencer CSR Registers on page 176

3.5.4.3.3. Reset Sequencer Control Register

The `Control` register allows you to control the Reset Sequencer.

Table 60. 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.

Related Information

Reset Sequencer CSR Registers on page 176

3.5.4.3.4. 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 61. 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).

Related Information

Reset Sequencer CSR Registers on page 176

3.5.4.3.5. 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 62. 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).

Related Information

Reset Sequencer CSR Registers on page 176

3.5.4.3.6. 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 63. 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.

Related Information

Reset Sequencer CSR Registers on page 176

3.5.4.3.7. 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 64. 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.

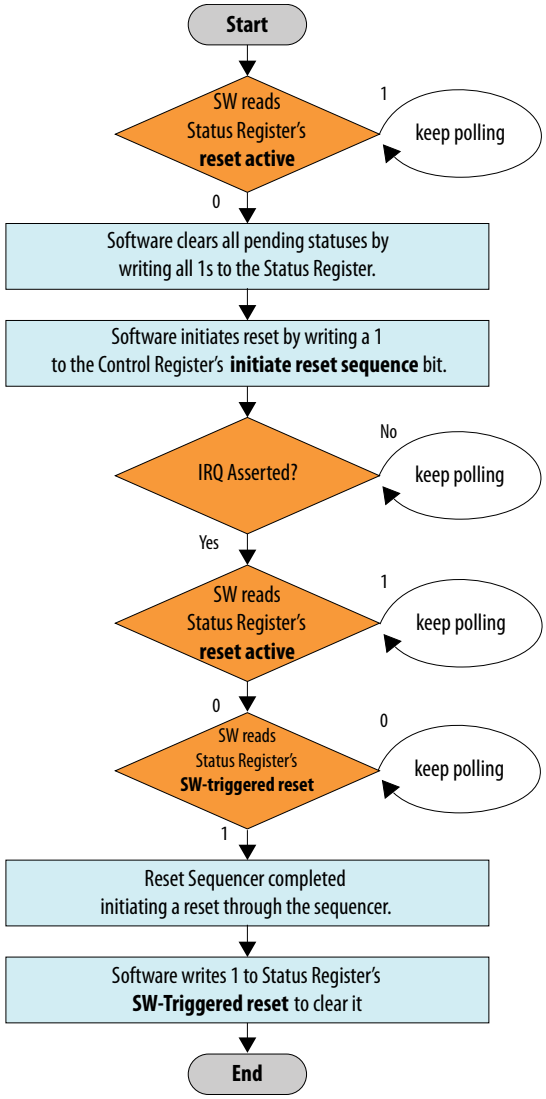
Related Information

Reset Sequencer CSR Registers on page 176

3.5.4.4. Reset Sequencer Software Flows

3.5.4.4.1. Reset Sequencer (Software-Triggered) Flow

Figure 106. Reset Sequencer (Software-Triggered) Flow Diagram



Related Information

- Reset Sequencer Status Register on page 176
- Reset Sequencer Control Register on page 179

3.5.4.4.2. 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.

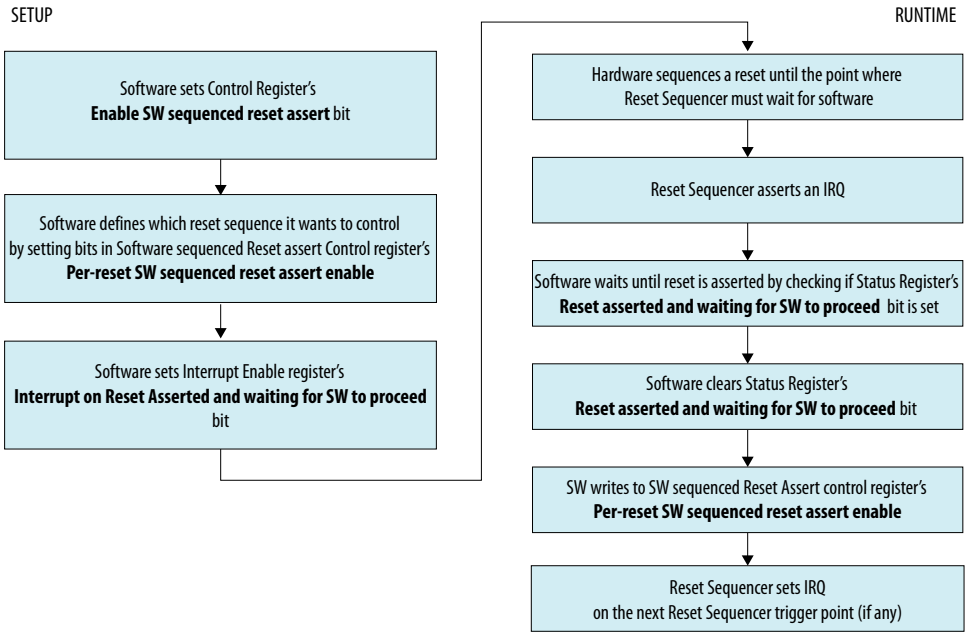
3.5.4.4.3. Reset Deassert Flow

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.

3.5.4.4.4. Reset Assert (Software Sequenced) Flow

Figure 107. Reset Assert (Software Sequenced) Flow



Related Information

- Reset Sequencer Control Register on page 179
- Reset Sequencer Software Sequenced Reset Assert Control Register on page 179
- Reset Sequencer Interrupt Enable Register on page 178
- Reset Sequencer Status Register on page 176

3.5.4.4.5. 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 183

3.6. 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 10

3.7. Interconnect Pipelining

Pipeline stages increase a design's f_{MAX} by reducing the combinational logic depth, at the cost of additional latency and logic.

The **Limit interconnect pipeline stages to** option in the **Interconnect Requirements** tab allows you to define the maximum Avalon-ST 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. Choosing 3 or 4 pipeline stages may significantly increase the logic utilization of the system.

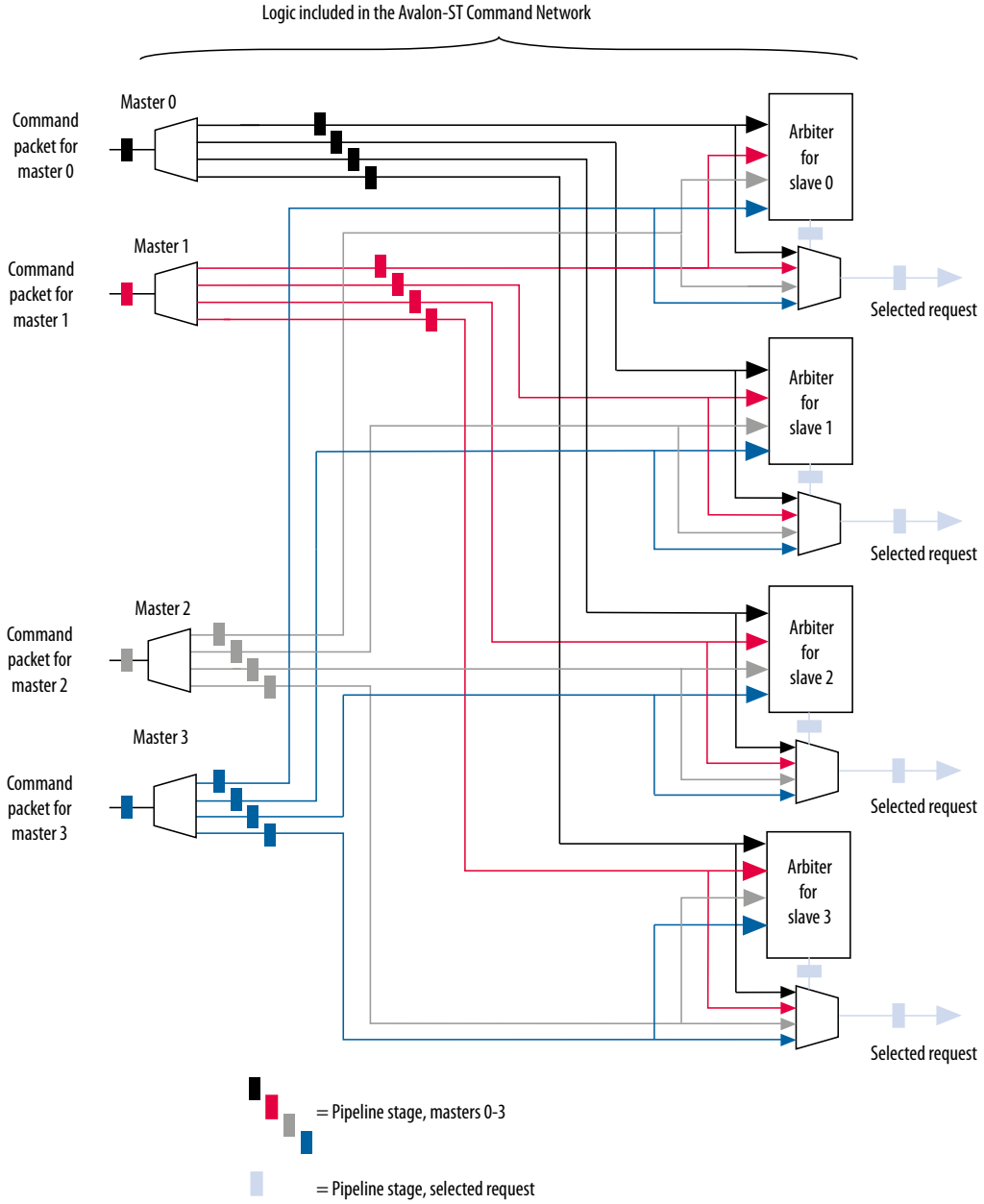
Platform Designer adds additional latency once on the command path, and once on the response path.

This setting is specific for each Platform Designer system or subsystem, so you can specify a unique interconnect pipeline stage value for each subsystem.

The insertion of pipeline stages depends upon the existence of certain interconnect components. For example, single-slave systems do not have multiplexers; therefore, multiplexer pipelining does not occur. In an extreme case, of a single-master to single-slave system, no pipelining occurs, regardless of the value of the **Limit interconnect pipeline stages to** option.

Figure 108. Pipeline Placement in Arbitration Logic

The example shows the possible placement of up to four potential pipeline stages. Platform Designer places these stages before the input to the demultiplexer, at the output of the multiplexer, between the arbiter and the multiplexer, and at the output of the demultiplexer.



You can manually adjust number of pipeline stages in the Platform Designer **Memory-Mapped Interconnect** tab.

Related Information

- Previewing the System Interconnect on page 38
- Inserting Pipeline Stages to Increase System Frequency on page 90

3.7.1. Manually Control Pipelining in the Platform Designer Interconnect

The **Memory-Mapped Interconnect** tab allows you to manipulate pipeline connections in the Platform Designer interconnect.

Consider manually pipelining the interconnect only when changes to the **Limit interconnect pipeline stages to** option do not improve frequency, and exhausted all other options to achieve timing closure, including the use of a bridge. Perform manual pipelining only in complete systems.

Access the **Memory-Mapped Interconnect** tab by clicking **System ➤ Show System With Platform Designer Interconnect**

1. In the Intel Quartus Prime software, compile the design and run timing analysis.
2. From the timing analysis output, identify the critical path through the interconnect and determine the approximate mid-point.
3. In Platform Designer, click **System ➤ Show System With Platform Designer Interconnect**.
4. In the **Memory-Mapped Interconnect** tab, select the interconnect module that contains the critical path.

   You can determine the name of the module from the hierarchical node names in the timing report.
5. Click **Show Pipelinable Locations**. Platform Designer display all possible pipeline locations in the interconnect. Right-click the possible pipeline location to insert or remove a pipeline stage.
6. Locate the possible pipeline location that is closest to the mid-point of the critical path. The names of the blocks in the memory-mapped interconnect tab correspond to the module instance names in the timing report.
7. Right-click the location where you want to insert a pipeline, and then click **Insert Pipeline**.
8. Regenerate the Platform Designer system, recompile the design, and then rerun timing analysis.
9. If necessary, repeat the manual pipelining process again until the design meets the timing requirements.

Manual pipelining has the following limitations:

- 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.
- 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.

3.8. Error Correction Coding (ECC) in Platform Designer Interconnect

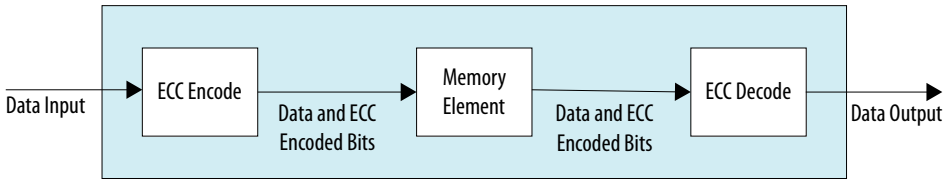
Error Correction Coding (ECC) logic allows the Platform Designer interconnect to detect and correct errors. Enabling ECC improves data integrity in memory blocks. Platform Designer supports ECC protection for Read Data FIFO (`rdata_FIFO`) instances only.

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.

When you enable ECC, Platform Designer interconnect sends uncorrectable errors arising from memory as `DECODEERROR` (`DECERR`) on the Avalon response bus.

Figure 109. 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 151
- Interconnect Requirements on page 40

3.9. 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.

Related Information

- Arm AMBA Protocol Specifications
- Slave Network Interfaces on page 138

3.9.1. Channels

Platform Designer has the following support and restrictions for AMBA 3 AXI channels.

3.9.1.1. 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 master interface has a 12-bit ID.

3.9.1.2. 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

3.9.1.3. Low Power Channel

Low power extensions are not supported in Platform Designer, version 14.0.

3.9.2. Cache Support

AWCACHE and ARCACHE are passed to an AXI slave unmodified.

3.9.2.1. Bufferable

Platform Designer interconnect treats AXI transactions as non-bufferable. All responses must come from the terminal slave.

When connecting to Avalon-MM slaves, since they do not have write responses, the following exceptions apply:

- For Avalon-MM slaves, the write response are generated by the slave agent once the write transaction is accepted by the slave. The following limitation exists for an Avalon bridge:
- For an Avalon bridge, the response is generated before the write reaches the endpoint; users must be aware of this limitation and avoid multiple paths past the bridge to any endpoint slave, or only perform bufferable transactions to an Avalon bridge.

3.9.2.2. Cacheable (Modifiable)

Platform Designer interconnect acknowledges the cacheable (modifiable) attribute of AXI transactions.

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 slave as modifiable because the size requires reduction.
- Platform Designer may consider AXI read and write transactions as modifiable when the destination is an Avalon slave. The AXI transaction may be split into multiple Avalon transactions if the slave 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 master transactions as non-bufferable and non-cacheable, with the allocate bits tied low.

3.9.3. 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 slave 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 50

3.9.4. Atomic Accesses

Exclusive accesses are supported for AXI slaves by passing the lock, transaction ID, and response signals from master to slave, with the limitation that slaves that do not reorder responses. Avalon slaves do not support exclusive accesses, and always return OKAY as a response. Locked accesses are also not supported.

3.9.5. Response Signaling

Full response signaling is supported. Avalon slaves always return OKAY as a response.

3.9.6. Ordering Model

Platform Designer interconnect provides responses in the same order as the commands are issued.

To prevent reordering, for slaves that accept reordering depths greater than 0, Platform Designer does not transfer the transaction ID from the master, but provides a constant transaction ID of 0. For slaves that do not reorder, Platform Designer allows the transaction ID to be transferred to the slave. To avoid cyclic dependencies, Platform Designer supports a single outstanding slave scheme for both reads and writes. Changing the targeted slave before all responses have returned stalls the master, regardless of transaction ID.

3.9.6.1. AXI and Avalon Ordering

There is a potential read-after-write risk when Avalon masters transact to AXI slaves.

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 master to the same slave to be in order.

In response to this potential risk, Avalon interfaces provide a compile-time option to enforce strict order. When turned on, the Avalon interface waits for outstanding write responses before issuing reads.

3.9.7. 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 slaves 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.

3.9.8. Unaligned Address Commands

Unaligned address commands are commands with addresses that do not conform to the data width of a slave. Since Avalon-MM slaves accept only aligned addresses, Platform Designer modifies unaligned commands from AXI masters to the correct data width. Platform Designer must preserve commands issued by AXI masters when passing the commands to AXI slaves.

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.

3.9.9. Avalon and AXI Transaction Support

Platform Designer 14.0 supports transactions between Avalon and interfaces, with some limitations.

3.9.9.1. Transaction Cannot Cross 4KB Boundaries

When an Avalon master issues a transaction to an AXI slave, the transaction cannot cross 4KB boundaries. Non-bursting Avalon masters already follow this boundary restriction.

3.9.9.2. Handling Read Side Effects

Read side effects can occur when more bytes than necessary are read from the slave, 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 master that issues a read command with an unaligned address starting at address `0x01`, and a burstcount of 2 to a 32-bit Avalon slave, the starting address is: `0x00`.
- For a 32-bit AXI master that issues a read command with an unaligned address starting at address `0x01`, with 4-bytes to an 8-bit AXI slave, the starting address is: `0x00`.

3.10. 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.

Related Information

Arm AMBA Protocol Specifications

3.10.1. 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 slave directly.

Intel recommends as an alternative that you instantiate the APB bridge and all the APB slaves in Platform Designer. You should then connect the slave side of the bridge to any high speed interface and connect the master side of the bridge to the APB slaves. 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 Avalon/AXI bridge to export the Avalon/AXI master to the top-level, and then connect this Avalon/AXI interface to the slave side of the APB bridge. Alternatively, instantiate the APB bridge in Platform Designer and export APB master to the top-level, and from there connect to APB bus outside of Platform Designer.

3.10.2. Burst Adaptation

APB is a non-bursting interface. Therefore, for any AXI or Avalon master with bursting support, a burst adapter is inserted before the slave interface and the burst transaction is translated into a series of non-bursting transactions before reaching the APB slave.

3.10.3. Width Adaptation

Platform Designer allows different data width connections with APB. When connecting a wider master to a narrower APB slave, the width adapter converts the wider transactions to a narrower transaction to fit the APB slave data width. APB does not support Write Strobe. Therefore, when you connect a narrower transaction to a wider APB slave, the slave cannot determine which byte lane to write. In this case, the slave data may be overwritten or corrupted.

3.10.4. Error Response

Error responses are returned to the master. Platform Designer performs error mapping if the master is an AMBA 3 AXI or AMBA 4 AXI master, for example, RRESP/BRESP= SLVERR. For the case when the slave does not use SLVERR signal, an OKAY response is sent back to master by default.

3.11. 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.

3.11.1. 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 slaves.

For narrow-sized transfers, bursts with Avalon slaves as destinations are shortened to multiple non-bursting transactions in order to transmit the correct address to the slaves, since Avalon slaves always perform full-sized `datawidth` transactions.

Bursts with AMBA 3 AXI slaves as destinations are shortened to multiple bursts, with each burst length less than or equal to 16. Bursts with AMBA 4 AXI slaves as destinations are not shortened.

3.11.2. QoS

Platform Designer routes 4-bit QoS signals (Quality of Service Signaling) on the read and write address channels directly from the master to the slave.

Transactions from AMBA 3 AXI and Avalon masters 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 slaves that do not support QoS.

For Platform Designer 14.0, there are no programmable QoS registers or compile-time QoS options for a master that overrides its real or default value.

3.11.3. Regions

For Platform Designer 14.0, there is no support for the optional regions feature. AMBA 4 AXI slaves with `AXREGION` signals are allowed. `AXREGION` signals are driven with the default value of `0x0`, and are limited to one entry in a master's address map.

3.11.4. 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

3.11.5. 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].

3.11.6. Width Adaptation and Data Packing in Platform Designer

Data packing applies only to systems where the data width of masters is less than the data width of slaves.

The following rules apply:

- Data packing is supported when masters and slaves are Avalon-MM.
- Data packing is not supported when any master or slave is an AMBA 3 AXI, AMBA 4 AXI, or APB component.

For example, for a read/write command with a 32-bit master connected to a 64-bit slave, and a transaction of 2 burstcounts, Platform Designer sends 2 separate read/write commands to access the 64-bit data width of the slave. Data packing is only supported if the system does not contain AMBA 3 AXI, AMBA 4 AXI, or APB masters or slaves.

3.11.7. Ordering Model

Out of order support is not implemented in Platform Designer, version 14.0. Platform Designer processes AXI slaves 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 slave must remain ordered. The interconnect always provides responses in the same order as the commands issued. Slaves that support reordering provide a constant transaction ID to prevent reordering. AXI slaves that do not reorder are provided with transaction IDs, which allows exclusive accesses to be used for such slaves.

3.11.8. 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.

3.11.9. Locked Transactions

Locked transactions are not supported for Platform Designer, version 14.0.

3.11.10. 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.

3.11.11. Mismatched Attributes

There are rules for how multiple masters issue cache values to a shared memory region. The interconnect meets requirements if signals are not modified.

3.11.12. 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

3.12. AMBA 4 AXI Streaming Interface Support (version 1.0)

3.12.1. 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_master` and `axi4stream_slave`. Connected interfaces must have the same port roles and widths.

Non matching master to slave connections, and multiple masters to multiple slaves connections are not supported.

3.12.1.1. AMBA 4 AXI Streaming Connection Point Parameters

Table 65. AMBA 4 AXI Streaming Connection Point Parameters

Name	Type	Description
`associatedClock`	string	Name of associated clock interface.
`associatedReset`	string	Name of associated reset interface

3.12.1.2. AMBA 4 AXI Streaming Connection Point Signals

Table 66. AMBA 4 AXI-Stream Connection Point Signals

Port Role	Width	Master Direction	Slave Direction	Required
tvalid	1	Output	Input	Yes
tready	1	Input	Output	No
tdata(1)	8:4096	Output	Input	No
tstrb	1:512	Output	Input	No
tkeep	1:512	Output	Input	No
tid(2)	1:8	Output	Input	No
tdest(3)	1:4	Output	Input	No
tuser(4)	1:4096	Output	Input	No
tlast	1	Output	Input	No

3.12.2. Adaptation

AMBA 4 AXI-Stream adaptation support is not available. AMBA 4 AXI-Stream master and slave interface signals and widths must match.

3.13. 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.

Platform Designer 14.0 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.

3.13.1. AMBA 4 AXI-Lite Signals

Platform Designer supports all AMBA 4 AXI-Lite interface signals. All signals are required.

(1) integer in multiple of bytes

(2) maximum 8-bits

(3) maximum 4-bits

(4) number of bits in multiple of the number of bytes of tdata

Table 67. 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

3.13.2. AMBA 4 AXI-Lite Bus Width

AMBA 4 AXI-Lite masters or slaves must have either 32-bit or 64-bit bus widths. Platform Designer interconnect inserts a width adapter if a master and slave pair have different widths.

3.13.3. 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.

3.13.4. AMBA 4 AXI-Lite IDs

AMBA 4 AXI-Lite does not support IDs. Platform Designer performs ID reflection inside the slave agent.

3.13.5. Connections Between AMBA 3 AXI,AMBA 4 AXI and AMBA 4 AXI-Lite

3.13.5.1. AMBA 4 AXI-Lite Slave Requirements

For an AMBA 4 AXI-Lite slave side, the master can be any master 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 masters require a burst adapter to shorten the burst length to 1 before sending a transaction to an AMBA 4 AXI-Lite slave.
- Platform Designer interconnect uses a width adapter for mismatched data widths.
- Platform Designer interconnect performs ID reflection inside the slave agent.
- An AMBA 4 AXI-Lite slave must have an address width of at least 12-bits.
- AMBA 4 AXI-Lite does not have the AXSIZE parameter. Narrow master to a wide AMBA 4 AXI-Lite slave is not supported. For masters that support narrow-sized bursts, for example, AMBA 3 AXI and AMBA 4 AXI, a burst to an AMBA 4 AXI-Lite slave must have a burst size equal to or greater than the slave's burst size.

3.13.5.2. AMBA 4 AXI-Lite Data Packing

Platform Designer interconnect does not support AMBA 4 AXI-Lite data packing.

3.13.6. 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 master 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.

3.14. 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.

3.14.1. AXI Master Interface Signal Types

Table 68. AXI Master 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

3.14.2. AXI Slave Interface Signal Types

Table 69. AXI Slave 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

3.14.3. AMBA 4 AXI Master Interface Signal Types

Table 70. AMBA 4 AXI Master 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

3.14.4. AMBA 4 AXI Slave Interface Signal Types

Table 71. AMBA 4 AXI Slave 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

3.14.5. AMBA 4 AXI-Stream Master and Slave Interface Signal Types

Table 72. AMBA 4 AXI-Stream Master and Slave Interface Signal Types

Name	Width	Master Direction	Slave 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

3.14.6. ACE-Lite Interface Signal Roles

Table 73. ACE-Lite Interface Signal Roles

Name	Width	Master Direction	Slave 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

3.14.7. APB Interface Signal Types

Table 74. APB Interface Signal Types

Name	Width	Direction APB Master	Direction APB Slave	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

3.14.8. Avalon Memory-Mapped Interface Signal Roles

Signal roles define the signal types that Avalon-MM master and slave ports allow.

This specification does not require all signals to exist in an Avalon-MM interface. There is no one signal that is always required. The minimum requirements for an Avalon-MM 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-MM interface:

Table 75. Avalon-MM Signal Roles

Some Avalon-MM 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	Master → Slave	No	Masters: 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 master must use the byteenable signal. Refer to the addressUnits interface property for word addressing. Slaves: By default, the interconnect translates the byte address into a word address in the slave's address space. From the perspective of the slave, each slave access is for a word of data. For example, address = 0 selects the first word of the slave. address = 1 selects the second word of the slave. Refer to the addressUnits interface property for byte addressing.
byteenable byteenable_n	2, 4, 8, 16, 32, 64, 128	Master → Slave	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 master 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 slave. During reads, byteenables indicate which bytes the master is reading. Slaves 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	Master → Slave	No	When asserted, allows the Nios II processor to write on-chip memories configured as ROMs.
read read_n	1	Master → Slave	No	Asserted to indicate a read transfer. If present, readdata is required.
readdata	8, 16, 32, 64, 128, 256, 512, 1024	Slave → Master	No	The readdata driven from the slave to the master in response to a read transfer. Required for interfaces that support reads.
response [1:0]	2	Slave → Master	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. • 00: OKAY—Successful response for a transaction. • 01: RESERVED—Encoding is reserved. • 10: SLAVEERROR—Error from an endpoint slave. Indicates an unsuccessful transaction. • 11: DECODEERROR—Indicates attempted access to an undefined location.

continued...

Signal Role	Width	Direction	Required	Description
				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	Master → Slave	No	Asserted to indicate a `write` transfer. If present, `writedata` is required.
`writedata`	8, 16, 32, 64, 128, 256, 512, 1024	Master → Slave	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	Master → Slave	No	`lock` ensures that once a master wins arbitration, the winning master maintains access to the slave 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 master has been granted, that master retains grant until `lock` is deasserted. A master equipped with `lock` cannot be a burst master. Arbitration priority values for lock-equipped masters are ignored. `lock` is particularly useful for read-modify-write (RMW) operations. The typical read-modify-write operation includes the following steps: 1. Master A asserts lock and reads 32-bit data that has multiple bit fields. 2. Master A deasserts lock, changes one bit field, and writes the 32-bit data back. `lock` prevents master B from performing a write between Master A's read and write.
`waitrequest` `waitrequest_n`	1	Slave → Master	No	A slave asserts `waitrequest` when unable to respond to a `read` or `write` request. Forces the master to wait until the interconnect is ready to proceed with the transfer. At the start of all transfers, a master initiates the transfer and waits until `waitrequest` is deasserted. A master must make no assumption about the assertion state of `waitrequest` when the master is idle: `waitrequest` may be high or low, depending on system properties. When `waitrequest` is asserted, master control signals to the slave must remain constant except for `beginbursttransfer`. For a timing diagram illustrating the `beginbursttransfer` signal, refer to the figure in *Read Bursts*.

continued...

Signal Role	Width	Direction	Required	Description
				An Avalon-MM slave may assert `waitrequest` during idle cycles. An Avalon-MM master may initiate a transaction when `waitrequest` is asserted and wait for that signal to be deasserted. To avoid system lockup, a slave device should assert `waitrequest` when in reset.
Pipeline Signals				
`readdatavalid` `readdatavalid_n`	1	Slave → Master	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*. A slave may assert `readdatavalid` to transfer data to the master independently of whether the slave is stalling a new command with `waitrequest`. Required if the master supports pipelined reads. Bursting masters with read functionality must include the `readdatavalid` signal.
`writeresponsevalid`	1	Slave → Master	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`.
Burst Signals				
`burstcount`	1 – 11	Master → Slave	No	Used by bursting masters 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 masters with read functionality must include the `readdatavalid` signal. For bursting masters and slaves using byte addresses, the following restriction applies to the width of the address: $<address_w> >= <burstcount_w> + \log_2(<symbols_per_word_of_interface>)$ For bursting masters and slaves using word addresses, the $\log_2$ term above is omitted.
`beginbursttransfer`	1	Interconnect → Slave	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. A slave 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.

3.14.9. Avalon Streaming Interface Signal Roles

Each signal in an Avalon-ST source or sink interface corresponds to one Avalon-ST signal role. An Avalon-ST interface may contain only one instance of each signal role. All Avalon-ST signal roles apply to both sources and sinks and have the same meaning for both.

Table 76. Avalon-ST 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 – 4,096	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.
`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 – 5	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.

3.14.10. Avalon Clock Source Signal Roles

An Avalon Clock source interface drives a clock signal out of a component.

Table 77. Clock Source Signal Roles

Signal Role	Width	Direction	Required	Description
`clk`	1	Output	Yes	An output clock signal.

3.14.11. Avalon Clock Sink Signal Roles

A clock sink provides a timing reference for other interfaces and internal logic.

Table 78. 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.

3.14.12. Avalon Conduit Signal Roles

Table 79. 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.

3.14.13. 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 masters and slaves and have the same meaning for both

Table 80. Tristate Conduit Interface Signal Roles

Signal Role	Width	Direction	Required	Description
`request`	1	Master → Slave	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. This protocol makes both rearbitration and continuous bus access possible if no other masters are requesting access.

continued...

Signal Role	Width	Direction	Required	Description
				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	Slave → Master	Yes	When asserted, indicates that a tristate conduit master 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	Slave → Master	No	The input signal of a logical tristate signal.
<*name*>_out	1 – 1024	Master → Slave	No	The output signal of a logical tristate signal.
<*name*>_outen	1	Master → Slave	No	The output enable for a logical tristate signal.

3.14.14. Avalon Tri-State Slave Interface Signal Types

Table 81. Tri-state Slave Interface Signal Types

Name	Width	Direction	Required	Description
`address`	1 - 32	input	No	Address lines to the slave port. Specifies a byte offset into the slave's address space.
`read` `read_n`	1	input	No	Read-request signal. Not required if the slave port never outputs data. If present, data must also be used.
`write` `write_n`	1	input	No	Write-request signal. Not required if the slave port never receives data from a master. If present, data must also be present, and `writebyteenable` cannot be present.
`chipselect` `chipselect_n`	1	input	No	When present, the slave port ignores all Avalon-MM 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 slave 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 slave device drives the data lines, and the FPGA captures the data signals and provides them to the master.

`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 master is writing to the slave. During reads, `byteenables` indicates which bytes the master is reading. Slaves 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 slave:
				`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`

continued...

Name	Width	Direction	Required	Description
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.				

3.14.15. Avalon Interrupt Sender Signal Roles

Table 82. Interrupt Sender Signal Roles

Signal Role	Width	Direction	Required	Description
irq irq_n	1-32	Output	Yes	Interrupt Request. An interrupt sender drives an interrupt signal to an interrupt receiver.

3.14.16. Avalon Interrupt Receiver Signal Roles

Table 83. Interrupt Receiver Signal Roles

Signal Role	Width	Direction	Required	Description
irq	1–32	Input	Yes	irq is an <*n*>-bit vector, where each bit corresponds directly to one IRQ sender with no inherent assumption of priority.

3.15. Platform Designer Interconnect Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • 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.
2017.11.06	17.1.0	• Changed instances of *Qsys* to *Platform Designer (Standard)* • Updated information about the Reset Sequencer.
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	Intel 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 Master and Slave 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-ST 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-ST 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-ST 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.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

4. 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.

Note: Intel now refers to Qsys as Platform Designer (Standard).

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 10
- Platform Designer Interconnect on page 128
- AMBA Protocol Specifications
- Embedded Peripherals IP User Guide
- Avalon Interface Specifications

4.1. Bridges

Bridges affect the way Platform Designer transports data between components. You can insert bridges between master and slave 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.

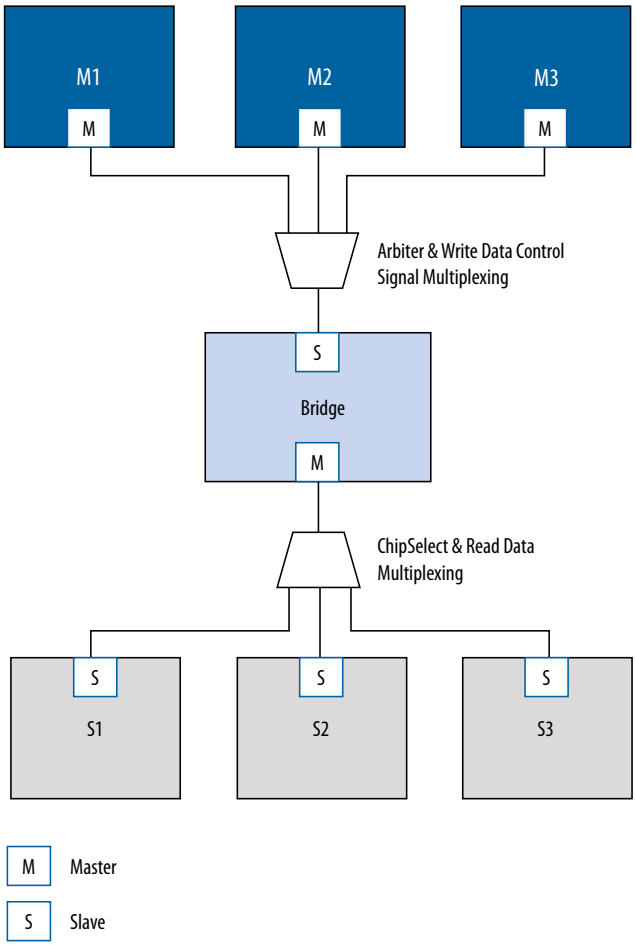
A bridge has one slave interface and one master interface. In Platform Designer, one or more master interfaces from other components connect to the bridge slave. The bridge master connects to one or more slave interfaces on other components.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 110. Using a Bridge in a Platform Designer System

In this example, three masters have logical connections to three slaves, although physically each master connects only to the bridge. Transfers initiated to the slave propagate to the master in the same order in which the transfers are initiated on the slave.

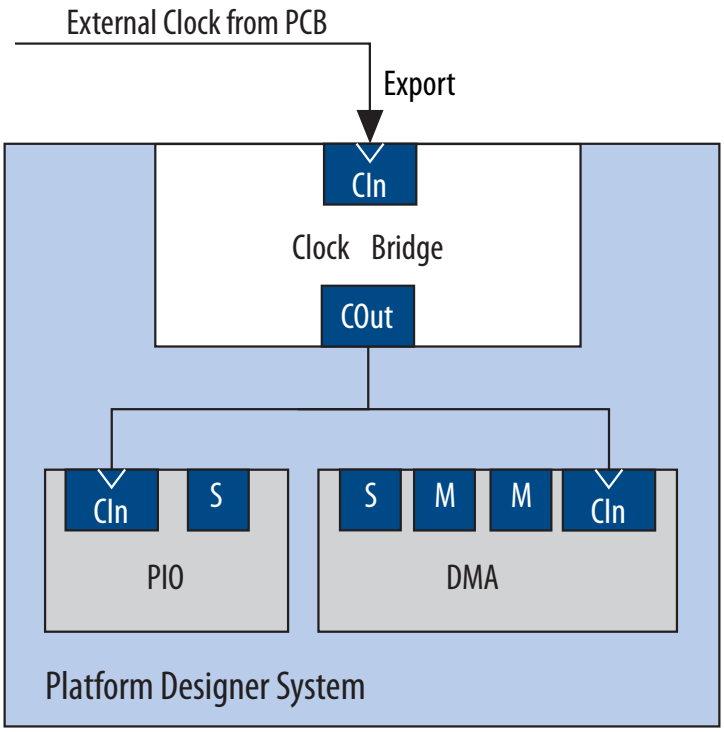


4.1.1. Clock Bridge

The Clock Bridge 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 111. Clock Bridge



4.1.2. Avalon-MM Clock Crossing Bridge

The Avalon-MM Clock Crossing Bridge transfers Avalon-MM commands and responses between different clock domains. You can also use the Avalon-MM Clock Crossing Bridge between AXI masters and slaves of different clock domains.

The Avalon-MM 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 master and slave 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 master and slave components on either side of the DC FIFO. For this configuration, you must assert both the resets on the master and the slave 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.

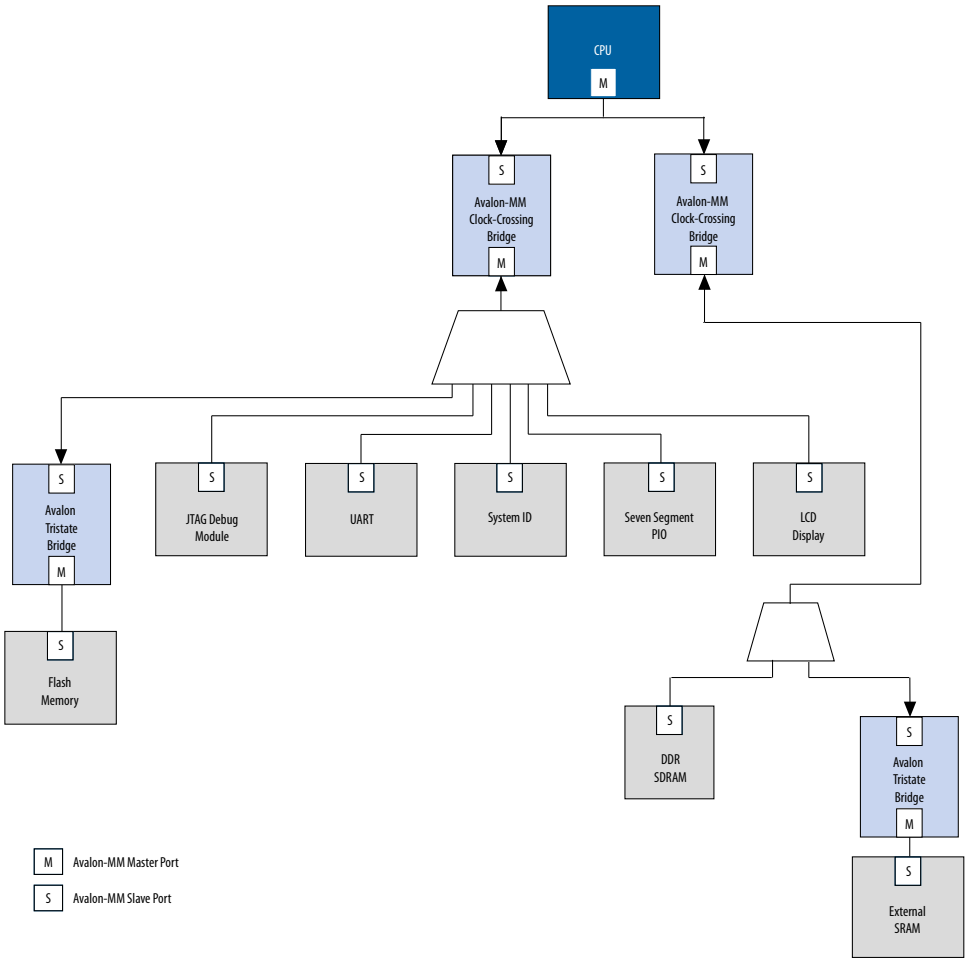
Note: The clock crossing bridge includes appropriate SDC constraints for its internal asynchronous FIFOs. For these SDC constraints to work correctly, do not set false paths on the pointer crossings in the FIFOs. Do not split the bridge's clocks into separate clock groups when you declare SDC constraints; the split has the same effect as setting false paths.

4.1.2.1. Avalon-MM Clock Crossing Bridge Example

In the example shown below, the Avalon-MM Clock Crossing bridges separate slave components into two groups. The Avalon-MM Clock Crossing Bridge places the low performance slave 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 Intel Quartus Prime Fitter to optimize paths that require minimal propagation delay.

Figure 112. Avalon-MM Clock Crossing Bridge



4.1.2.2. Avalon-MM Clock Crossing Bridge Parameters

Table 84. Avalon-MM 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 master 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 slaves.
Use automatically-determined address width	-	The minimum bridge address width that is required to address the downstream slaves.
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 (master-to-slave) FIFO depth.
Respond FIFO depth	2, 4, 8,16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384 bits	Slave-to-master 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 master to target slave 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 slave to master 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.

4.1.3. Avalon-MM Pipeline Bridge

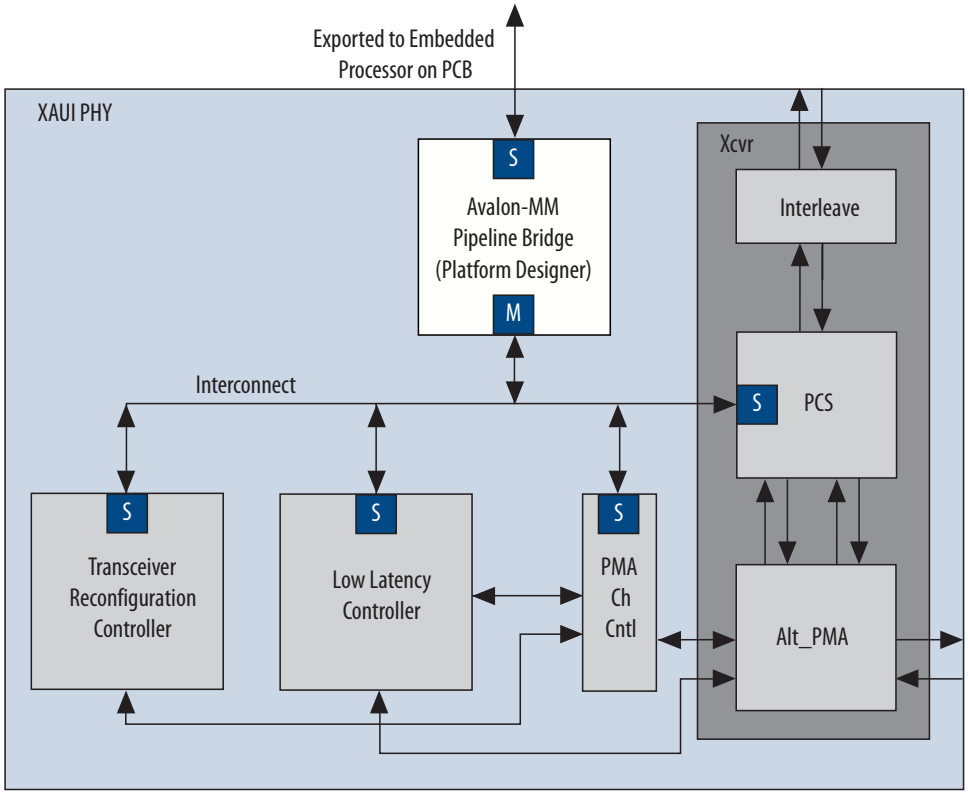
The Avalon-MM Pipeline Bridge inserts a register stage in the Avalon-MM command and response paths. The bridge accepts commands on its slave port and propagates the commands to its master port. The pipeline bridge provides separate parameters to turn on pipelining for command and response signals.

The **Maximum pending read transactions** parameter is the maximum number of pending reads that the Avalon-MM bridge can queue up. To determine the best value for this parameter, review this same option for the bridge's connected slaves and identify the highest value of the parameter, and then add the internal buffering requirements of the Avalon-MM bridge. In general, the value is between 4 and 32. The limit for maximum queued transactions is 64.

You can use the Avalon-MM bridge to export a single Avalon-MM slave interface to control multiple Avalon-MM slave devices. The pipelining feature is optional.

Figure 113. Avalon-MM Pipeline Bridge in a XAUI PHY Transceiver IP Core

In this example, the bridge transfers commands received on its slave interface to its master port.

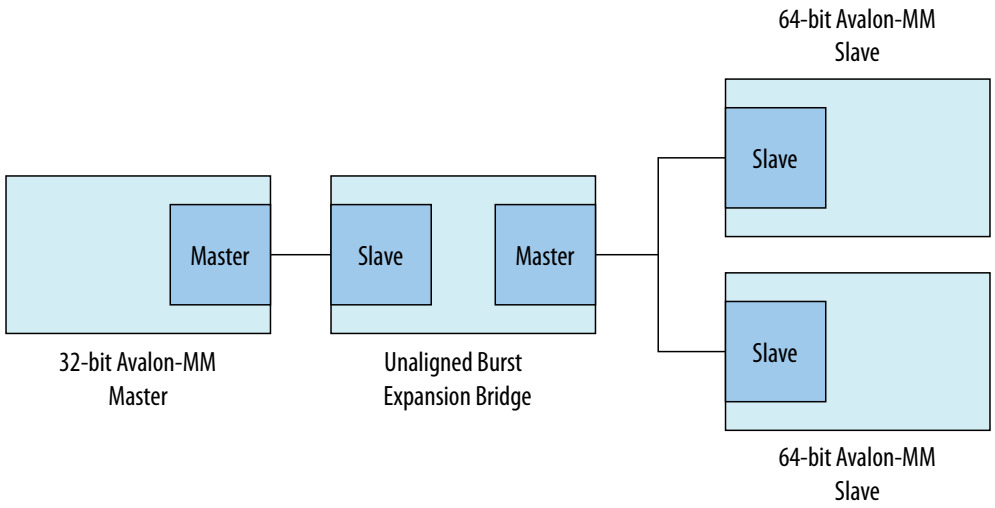


Because the slave interface is exported to the pins of the device, having a single slave port, rather than separate ports for each slave device, reduces the pin count of the FPGA.

4.1.4. Avalon-MM Unaligned Burst Expansion Bridge

The Avalon-MM Unaligned Burst Expansion Bridge aligns read burst transactions from masters connected to its slave interface, to the address space of slaves connected to its master interface. This alignment ensures that all read burst transactions are delivered to the slave as a single transaction.

Figure 114. Avalon-MM Unaligned Burst Expansion Bridge



You can use the Avalon Unaligned Burst Expansion Bridge to align read burst transactions from masters that have narrower data widths than the target slaves. Using the bridge for this purpose improves bandwidth utilization for the master-slave pair, and ensures that unaligned bursts are processed as single transactions rather than multiple transactions.

Note: Do not use the Avalon-MM Unaligned Burst Expansion Bridge if any connected slave has read side effects from reading addresses that are exposed to any connected master's address map. This bridge can cause read side effects due to alignment modification to read burst transaction addresses.

Note: The Avalon-MM Unaligned Burst Expansion Bridge does not support VHDL simulation.

4.1.4.1. Using the Avalon-MM Unaligned Burst Expansion Bridge

When a master sends a read burst transaction to a slave, the Avalon-MM Unaligned Burst Expansion Bridge initially determines whether the start address of the read burst transaction is aligned to the slave'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-MM Unaligned Burst Expansion Bridge then determines whether the final word requested by the master is the last word at the slave read burst address. If a single slave 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 master is the last word at the slave read burst address, the bridge does not modify the burst length of the read burst command to the slave.
- If the final word requested by the master is not the last word at the slave read burst address, the bridge increases the burst length of the read burst command to the slave. The final word requested by the modified read burst command is then the last word at the slave read burst address.

The bridge stores information about each aligned read burst command that it sends to slaves connected to a master interface. When a read response is received on the master 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 master 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 master.

4.1.4.2. Avalon-MM Unaligned Burst Expansion Bridge Parameters

Figure 115. Avalon-MM Unaligned Burst Expansion Bridge Parameter Editor

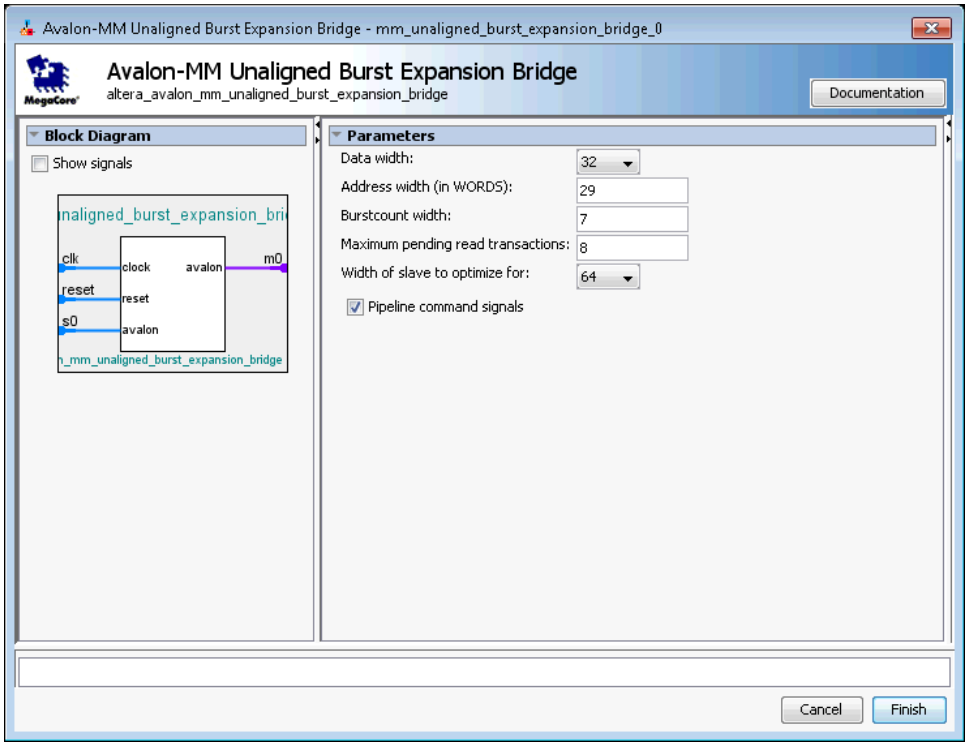


Table 85. Avalon-MM Unaligned Burst Expansion Bridge Parameters

Parameter	Description
Data width	Data width of the master connected to the bridge.
Address width (in WORDS)	The address width of the master connected to the bridge.
Burstcount width	The burstcount signal width of the master connected to the bridge.
Maximum pending read transactions	The **Maximum pending read transactions** parameter is the maximum number of pending reads that the Avalon-MM bridge can queue up. To determine the best value for this parameter, review this same option for the bridge's connected slaves and identify the highest value of the parameter, and then add the internal buffering requirements of the Avalon-MM bridge. In general, the value is between 4 and 32. The limit for maximum queued transactions is 64.
Width of slave to optimize for	The data width of the connected slave. Supported values are: 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096 bits.

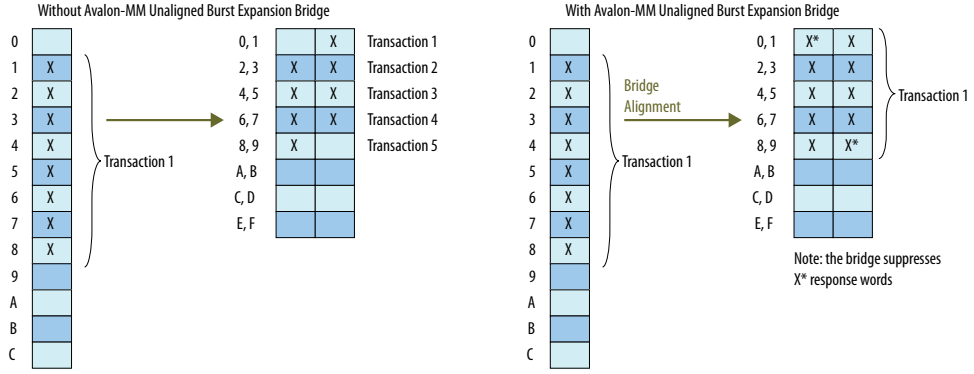
continued...

Parameter	Description
	Note: If you connect multiple slaves, all slaves 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.

4.1.4.3. Avalon-MM Unaligned Burst Expansion Bridge Example

Figure 116. Unaligned Burst Expansion Bridge

The example below shows an unaligned read burst command from a master that the Avalon-MM Unaligned Burst Expansion Bridge converts to an aligned request for a connected slave, and the suppression of words due to the aligned read burst command. In this example, a 32-bit master requests an 8-beat burst of 32-bit words from a 64-bit slave with a start address that is not 64-bit aligned.



Because the target slave has a 64-bit data width, address 1 is unaligned in the slave's address space. As a result, several smaller burst transactions are needed to request the data associated with the master's read burst command.

With an Avalon-MM Unaligned Burst Expansion Bridge in place, the bridge issues a new read burst command to the target slave beginning at address 0 with burst length 10, which requests data up to the word stored at address 9.

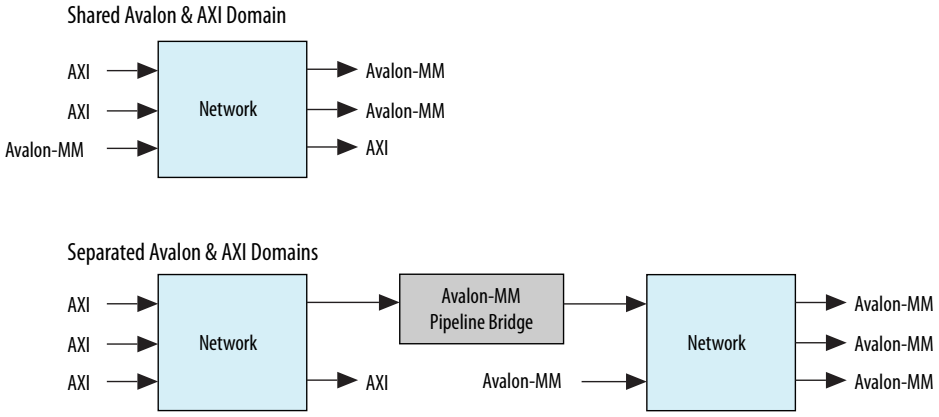
When the bridge receives the word corresponding to address 0, it suppresses it from the master, and then delivers the words corresponding to addresses 1 through 8 to the master. When the bridge receives the word corresponding to address 9, it suppresses that word from the master.

4.1.5. 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.

Figure 117. Avalon-MM Pipeline Bridge Between Avalon-MM and AXI Domains

Using an explicit Avalon-MM bridge to separate the AXI and Avalon domains reduces the amount of bridging logic in the interconnect at the expense of concurrency.



4.1.6. AXI Bridge

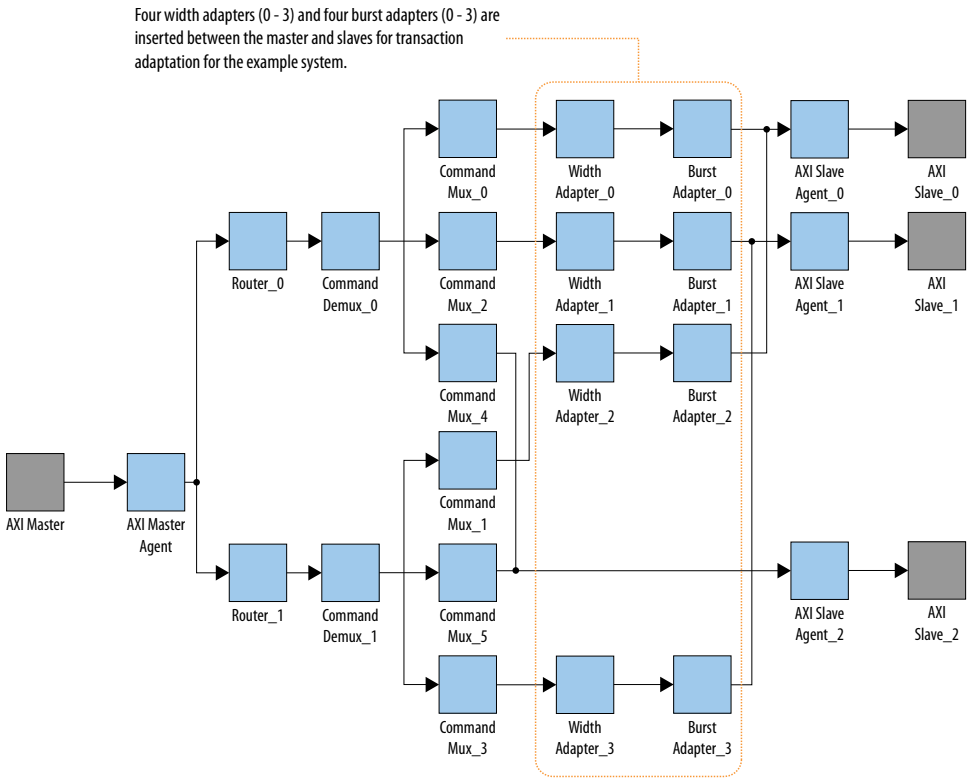
With an AXI bridge, you can influence the placement of resource-intensive components, such as the width and burst adapters. Depending on its use, an AXI bridge may reduce throughput and concurrency, in return for higher f_{MAX} and less logic.

You can use an AXI bridge 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 master or slave interfaces. You can also use an AXI bridge to export AXI interfaces from Platform Designer systems.

Example 9. Reducing the Number of Adapters by Adding a Bridge

The figure shows a system with a single AXI master and three AXI slaves. It also has various interconnect components, such as routers, demultiplexers, and multiplexers. Two of the slaves have a narrower data width than the master; 16-bit slaves versus a 32-bit master.

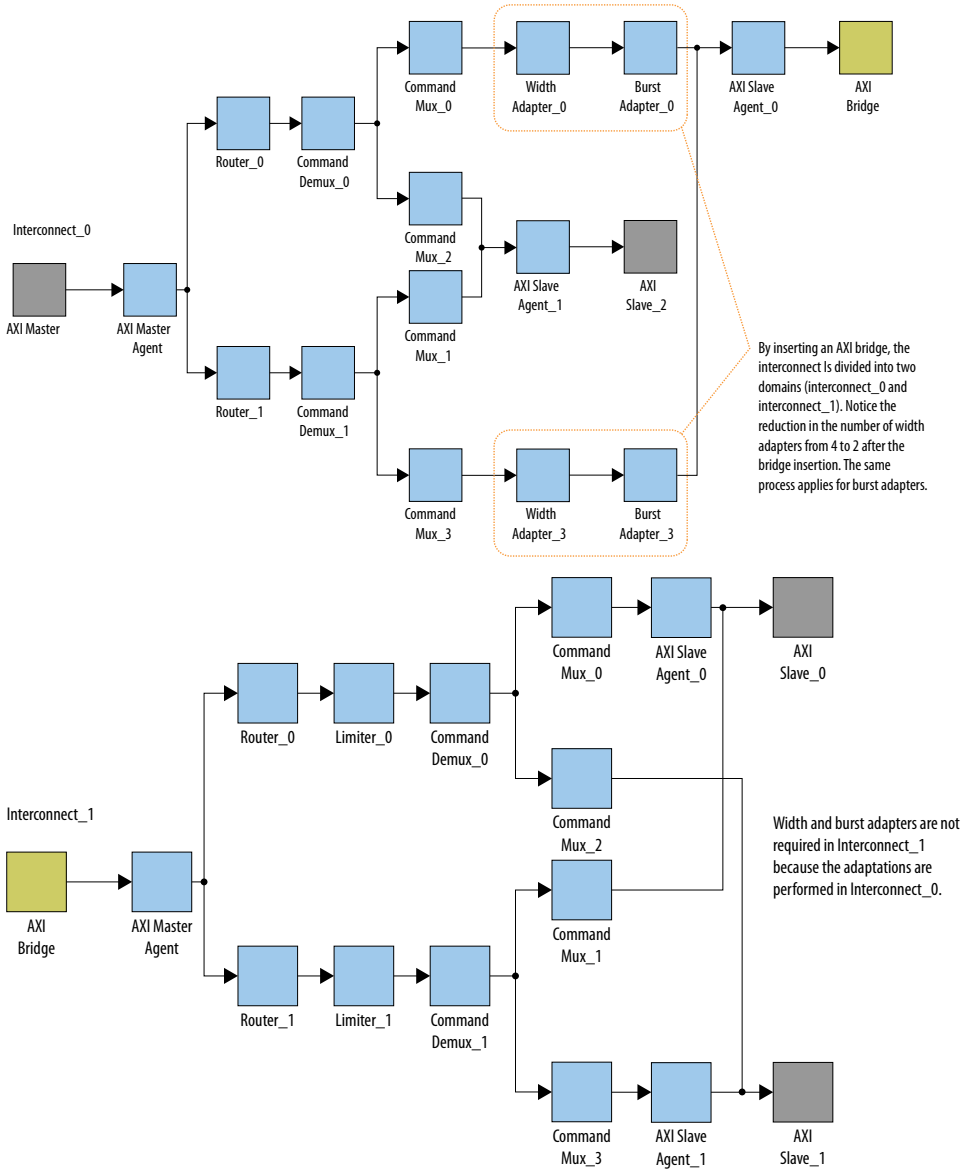
Figure 118. AXI System Without a Bridge



In this system, Platform Designer interconnect creates four width adapters and four burst adapters to access the two slaves.

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 119. 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.

4.1.6.1. 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 3 AXI master and slave interfaces into the component.

Note: In AMBA 3 AXI, aw/aruser accommodates sideband signal usage by hard processor systems (HPS).

Table 86. Sets of Signals for the AXI Bridge Based on the Protocol

Signal Name	AMBA 3 AXI	AMBA 3 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

4.1.6.2. AXI Bridge Parameters

In the parameter editor, you can customize the parameters for the AXI bridge according to the requirements of your design.

Figure 120. AXI Bridge Parameter Editor

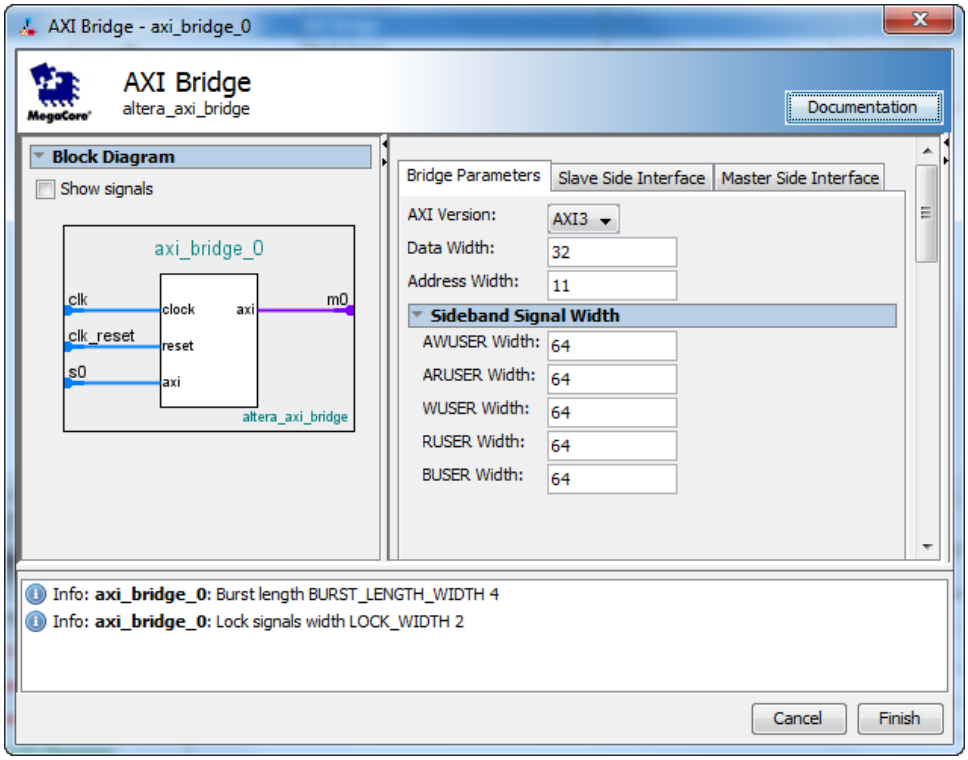


Table 87. AXI Bridge Parameters

Parameter	Type	Range	Description
AXI Version	string	AMBA 3 AXI or AMBA 3 AXI	Specifies the AXI version and signals that Platform Designer generates for the slave and master interfaces of the bridge.
Data Width	integer	8:1024	Controls the width of the data for the master and slave interfaces.
Address Width	integer	1-64 bits	Controls the width of the address for the master and slave interfaces.
AWUSER Width	integer	1-64 bits	Controls the width of the write address channel sideband signals of the master and slave interfaces.
ARUSER Width	integer	1-64 bits	Controls the width of the read address channel sideband signals of the master and slave interfaces.
WUSER Width	integer	1-64 bits	Controls the width of the write data channel sideband signals of the master and slave interfaces.
RUSER Width	integer	1-16 bits	Controls the width of the read data channel sideband signals of the master and slave interfaces.
BUSER Width	integer	1-16 bits	Controls the width of the write response channel sideband signals of the master and slave interfaces.

4.1.6.3. AXI Bridge Slave and Master Interface Parameters

Table 88. AXI Bridge Slave and Master Interface Parameters

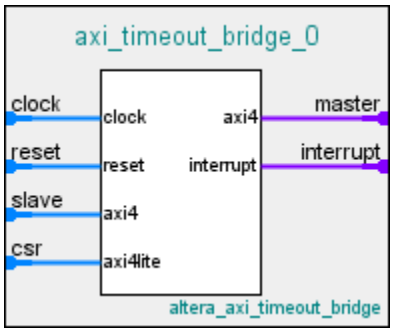
Parameter	Description
ID Width	Controls the width of the thread ID of the master and slave interfaces.
Write/Read/Combined Acceptance Capability	Controls the depth of the FIFO that Platform Designer needs in the interconnect agents based on the maximum pending commands that the slave interface accepts.
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 master 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.

4.1.7. AXI Timeout Bridge

The AXI Timeout Bridge allows your system to recover when it freezes, and facilitates debugging. You can place an AXI Timeout Bridge between a single master and a single slave if you know that the slave may time out and cause your system to freeze. If a slave does not accept a command or respond to a command it accepted, its master can wait indefinitely.

Figure 121. AXI Timeout Bridge

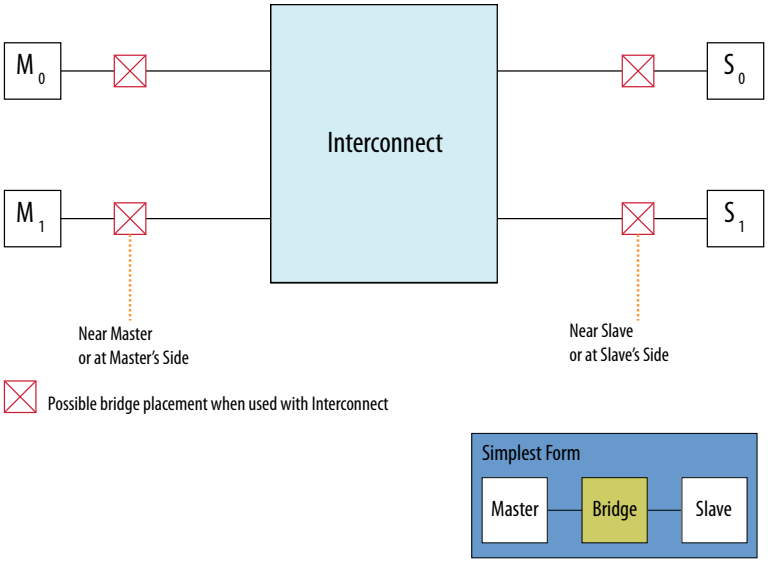


For a domain with multiple masters and slaves, placement of an AXI Timeout Bridge in your design may be beneficial in the following scenarios:

- To recover from a freeze, place the bridge near the slave. If the master attempts to communicate with a slave that freezes, the AXI Timeout Bridge frees the master by generating error responses. The master is then able to communicate with another slave.
- When debugging your system, place the AXI Timeout Bridge near the master. This placement enables you to identify the origin of the burst, and to obtain the full address from the master. Additionally, placing an AXI Timeout Bridge near the master enables you to identify the target slave for the burst.

  Note: If you place the bridge at the slave's side and you have multiple slaves connected to the same master, you do not get the full address.

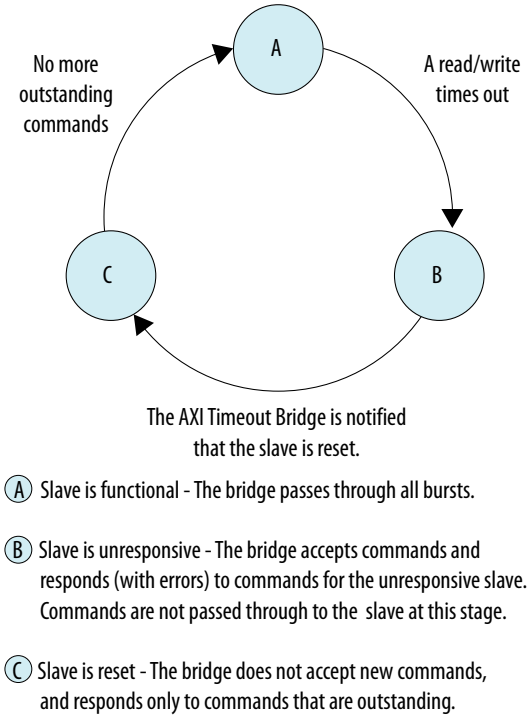
Figure 122. AXI Timeout Bridge Placement



4.1.7.1. 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 123. 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 master on behalf of the unresponsive slave. This stage frees the master and certifies the unresponsive slave as dysfunctional.
- The AXI Timeout Bridge accepts subsequent write addresses, write data, and read addresses to the dysfunctional slave. The bridge does not accept outstanding write responses, and read data from the dysfunctional slave is not passed through to the master.
- The `awvalid`, `wvalid`, `bready`, `arvalid`, and `rready` ports are held low at the master interface of the bridge.

Note: After a timeout, `awvalid`, `wvalid`, and `arvalid` may be dropped before they are accepted by `awready` at the master interface. While the behavior violates the AXI specification, it occurs only on an interface connected to the slave 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 89. 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: Master, Slave, 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 3 AXI, or Avalon master or slave interface. Avalon masters must utilize the bridge's interrupt output to detect a timeout.

The bridge slave 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 slave must be reset and the bridge notified of the change in status via the CSR. Once the CSR notifies the bridge that the slave 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 90. CSR Interrupt Status Information for the AXI Timeout Bridge

Address	Attribute	Name
`0x0`	write-only	Slave is reset
`0x4`	read-only	Timed out operation
`0x8 through 0xF`	read-only	Timed out address

4.1.7.2. AXI Timeout Bridge Parameters

Table 91. AXI Timeout Bridge Parameters

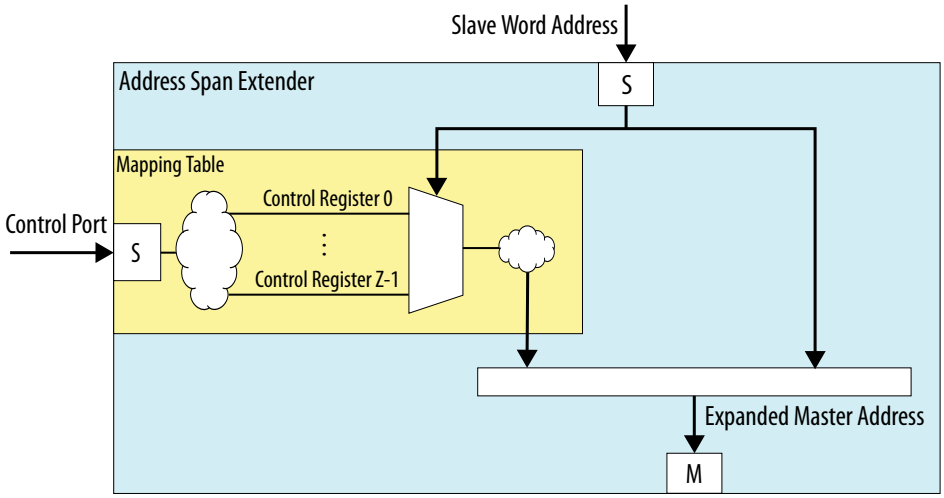
Parameter	Description
ID width	The width of `awid`, `bid`, `arid`, or `rid`.
Address width	The width of `awaddr` or `araddr`.
Data width	The width of `wdata` or `rdata`.
User width	The width of `awuser`, `wuser`, `buser`, `aruser`, or `ruser`.
Maximum number of outstanding writes	The expected maximum 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.

4.1.8. Address Span Extender

The **Address Span Extender** allows memory-mapped master interfaces to access a larger or smaller address map than the width of their address signals allows. The address span extender splits the addressable space into multiple separate windows, so that the master can access the appropriate part of the memory through the window.

The address span extender does not limit master and slave widths to a 32-bit and 64-bit configuration. You can use the address span extender with 1-64 bit address windows.

Figure 124. Address Span Extender



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.

4.1.8.1. 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).

4.1.8.2. Address Span Extender Parameters

Table 92. Address Span Extender Parameters

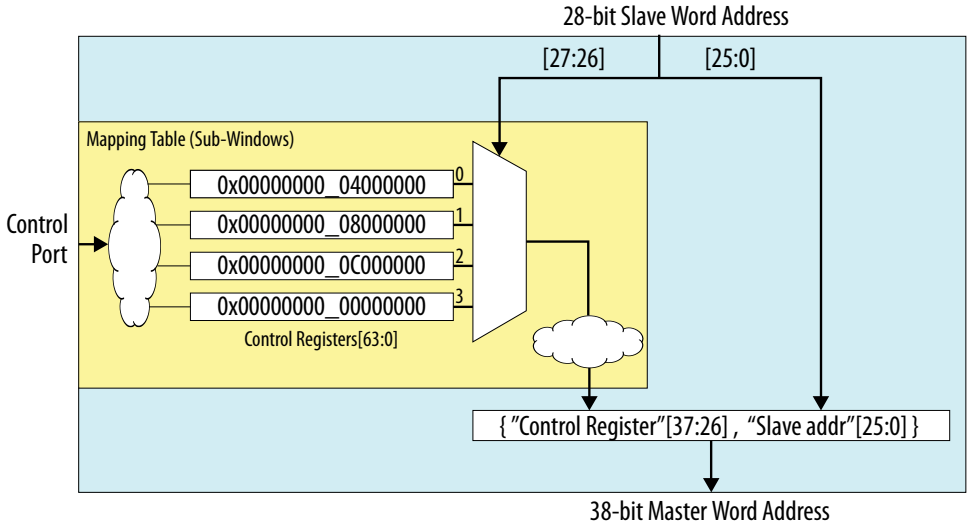
Parameter	Description
Datapath Width	Width of write data and read data signals.
Expanded Master Byte Address Width	Width of the master byte address port. That is, the address span size of all the downstream slaves that attach to the address span extender.
Slave Word Address Width	Width of the slave word address port. That is, the address span size of the downstream slaves that the upstream master accesses.
Burstcount Width	Burst count port width of the downstream slave and the upstream master that attach to the address span extender.
Number of sub-windows	The slave port can represent one or more windows in the master address span. You can subdivide the slave address span into *N* equal spans in *N* sub-windows. A remapping register in the CSR slave 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 master 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 slave.
Maximum Pending Reads	Sets the bridge slave'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 slaves that you attach to this bridge.

4.1.8.3. Calculating the Address Span Extender Slave Address

The diagram describes how Platform Designer calculates the slave address. In this example, the address span extender is configured with a 28-bit address space for slaves. 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.

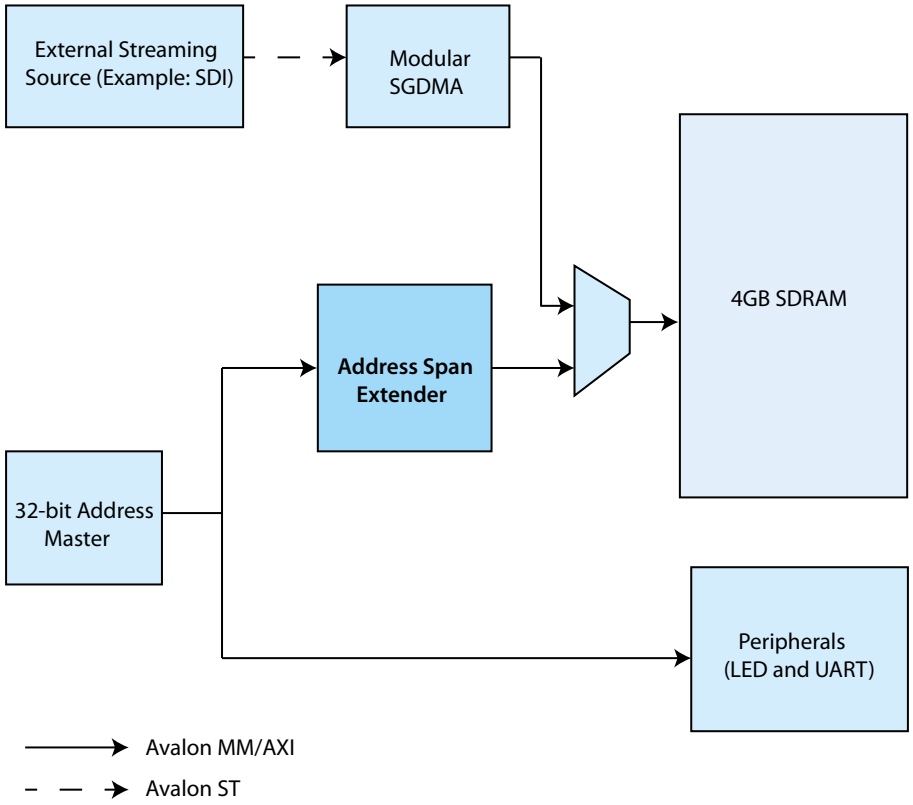
Figure 125. Address Span Extender



4.1.8.4. Using the Address Span Extender

This example shows when and how to use address span extender component in your Platform Designer design.

Figure 126. Block Diagram with Address Span Extender



In the above design, a 32-bit master shares 4 GB SDRAM with an external streaming interface. The master has the path to access streaming data from the SDRAM DDR memory. However, if you connect the whole 32-bit address bus of the master to the SDRAM DDR memory, you cannot connect the master to peripherals such as LED or UART. To avoid this situation, you can implement the address span extender between the master and DDR memory. The address span extender allows the master to access the SDRAM DDR memory and the peripherals at the same time.

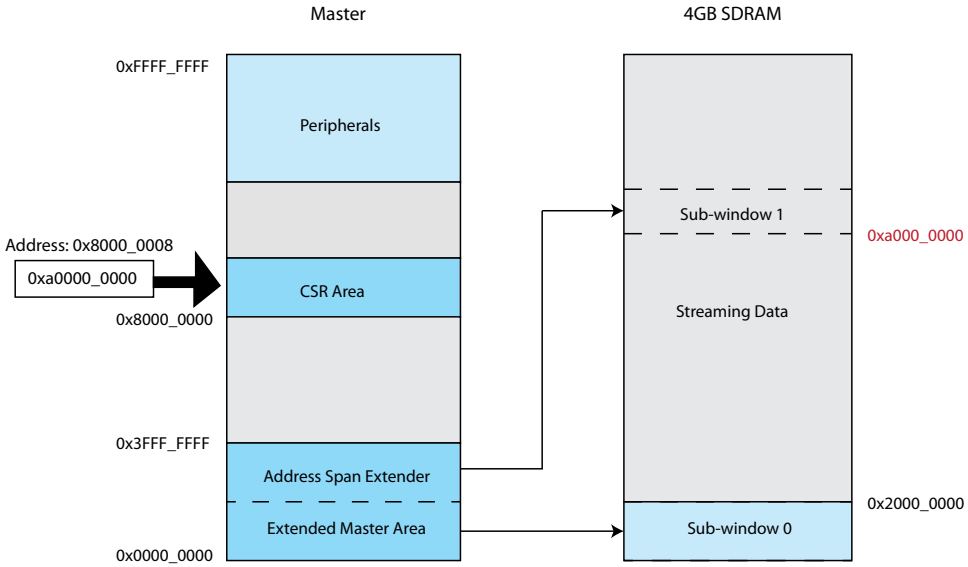
To implement address span extender for the above example, you can divide the address window of the address span extender into two sub-windows of 512 MB each. The sub-window 0 is for the master program area. You can dynamically map the sub-window 1 to any area other than the program area.

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 93. CSR Mapping Table

Address	Data
0x8000_0000	0x0000_0000
0x8000_0008	0xa000_0000

Figure 127. Memory mapping for Address Span Extender

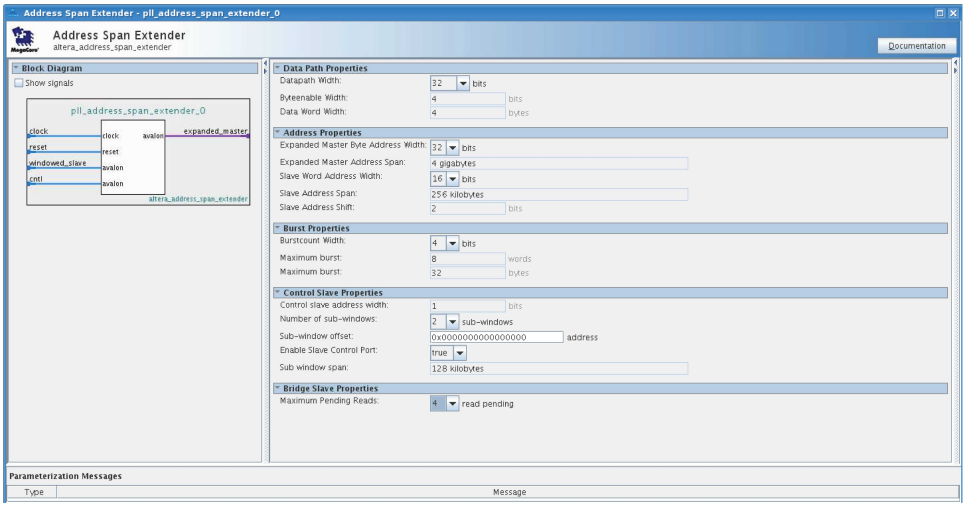


The table below indicates the Platform Designer parameter settings for this address span extender example.

Table 94. 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 master and SDRAM DDR memory is minimal, set the datapath width to align with the upstream master.
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 master. The number of bytes over the data word width in the **Datapath Properties** (4 bytes for this example) accounts for the slave 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 128. Address Span Extender Parameter Editor



Note: You can view the address span extender connections in the **System View** tab. The windowed slave port and control port connect to the master. The expanded master port connects to the SDRAM DDR memory.

4.1.8.5. 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 slave 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 slave 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.

4.1.8.6. Nios II Support

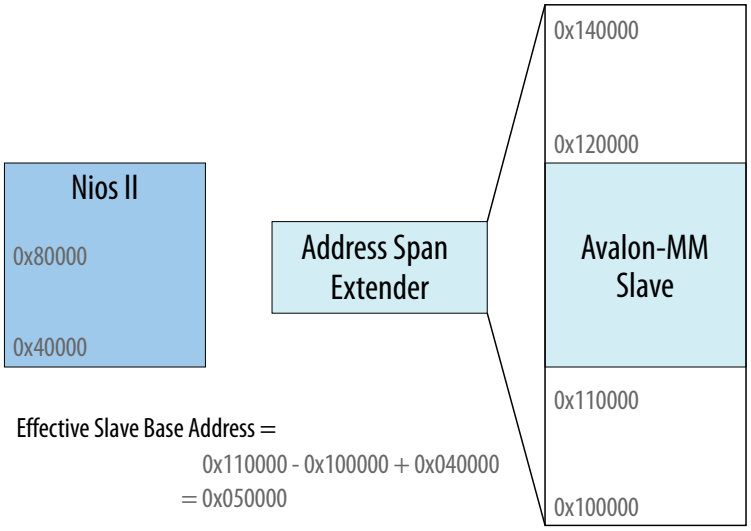
If the address span extender window is fixed, for example, the **Disable Slave Control Port** option is turned on, then the address span extender performs as a bridge. Components on the slave side of the address span extender that are within the window are visible to the Nios II processor. Components 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 a slave at base address `0x1100000`. The slave appears to the Nios II processor as being at address:

```
0x110000 - 0x100000 + 0x40000 = 0x050000
```

Figure 129. 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 slave side of the address span extender.

4.2. Error Response Slave

The Error Response Slave provides a predictable error response service for master 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 slave access. Therefore, an AXI system with address space not fully decoded to slave interfaces requires the Error Response Slave.

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 masters of different data widths, including Avalon or AXI-Lite masters, 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 master interfaces without instantiating a bridge. When you connect an Error Response Slave to a master, the Error Response Slave accepts cycles sent from the master, 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.

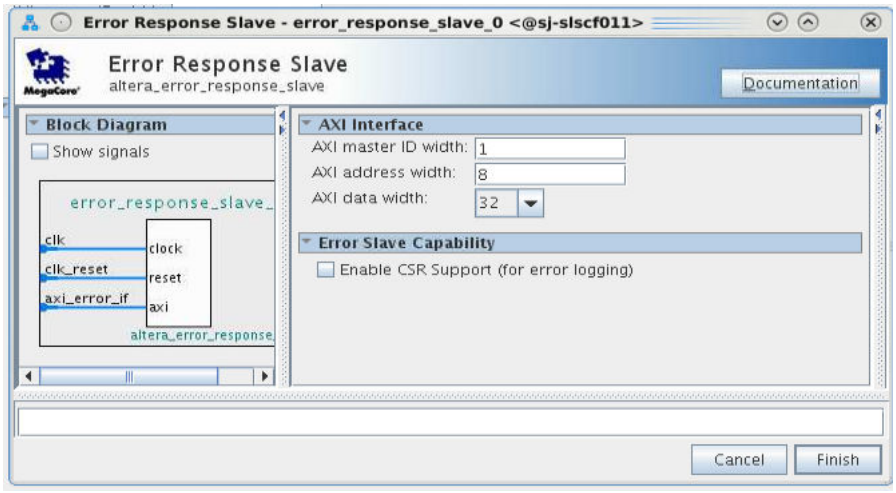
- To set the Error Response Slave as the default slave for a master interface in your system, connect the slave to the master in your Platform Designer system.
- A system can contain more than one Error Response Slave.
- As a best practice, instantiate separate Error Response Slave components for each AXI master in your system.

Related Information

- AMBA 3 AXI Protocol Specification Support (version 1.0) on page 188
- Designating a Default Slave on page 242

4.2.1. Error Response Slave Parameters

Figure 130. Error Response Slave Parameter Editor



If you turn on **Enable CSR Support (for error logging)** more parameters become available.

Figure 131. Error Response Slave Parameter Editor with Enabled CSR Support

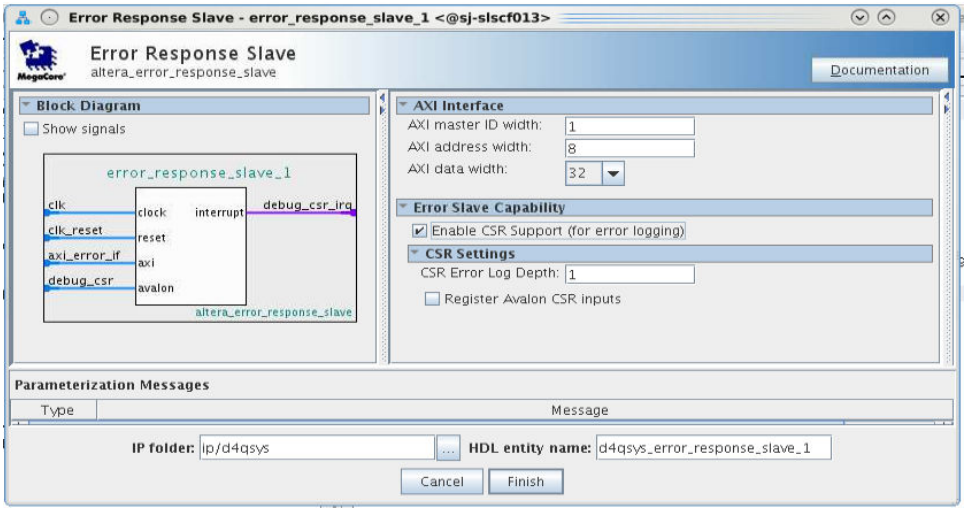


Table 95. Error Response Slave Parameters

Parameter	Value	Description
AXI master ID width	1-8 bits	Specifies the master 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.
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.

4.2.2. 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.

4.2.2.1. 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.

4.2.2.2. CSR Interrupt Status Registers

Table 96. 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.

continued...

Offset	Bits	Attribute	Default	Description
				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.

4.2.2.3. 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 97. 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.
	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`: Master 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.

4.2.2.4. 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 98. 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.

continued...

Offset	Bits	Attribute	Default	Description
	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`: Master 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.
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.

4.2.3. Designating a Default Slave

You can designate any slave in your Platform Designer system as the error response default slave. The default slave you designate provides an error response service for masters 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 slave that you want to designate as the default slave, and then click the checkbox for the slave in the **Default Slave** column.
3. In the **System View** tab, in the **Connections** column, connect the designated default slave to one or more masters.

Related Information

Specifying a Default Slave on page 52

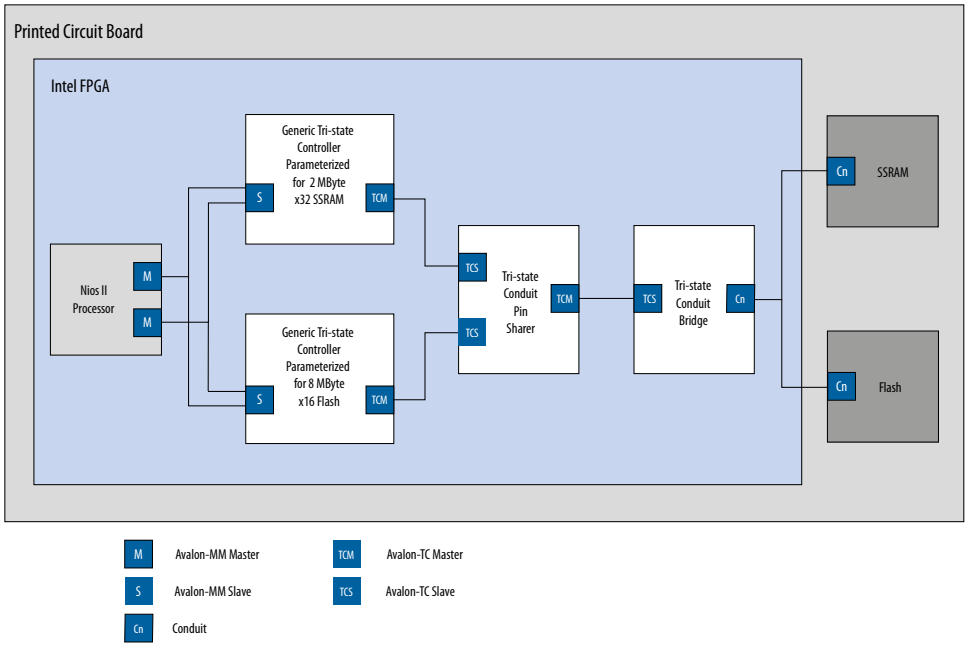
4.3. Tri-State Components

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 10. 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 132. Tri-State Conduit System to Control Off-Chip SRAM and Flash Devices

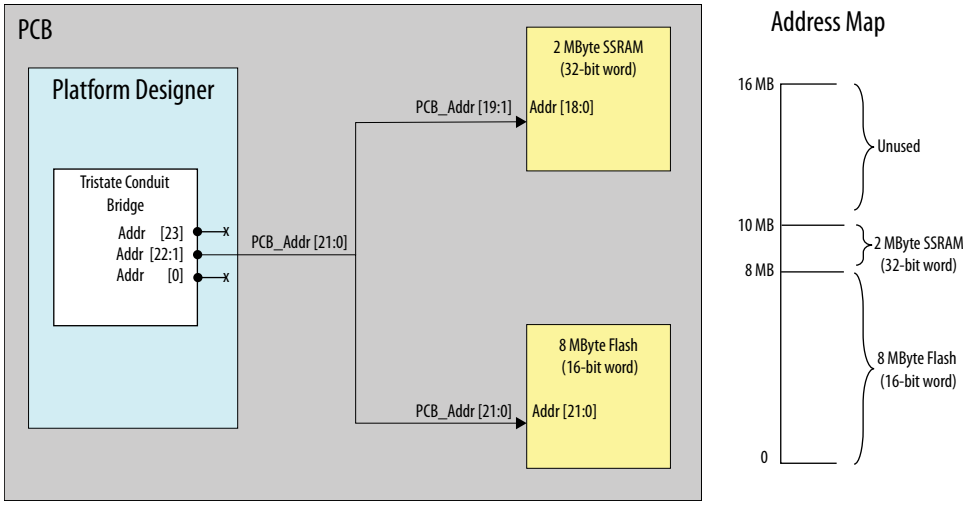


Address Connections from Platform Designer System to PCB

The flash device operates on 16-bit words and must ignore the least-significant bit of the Avalon-MM 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.

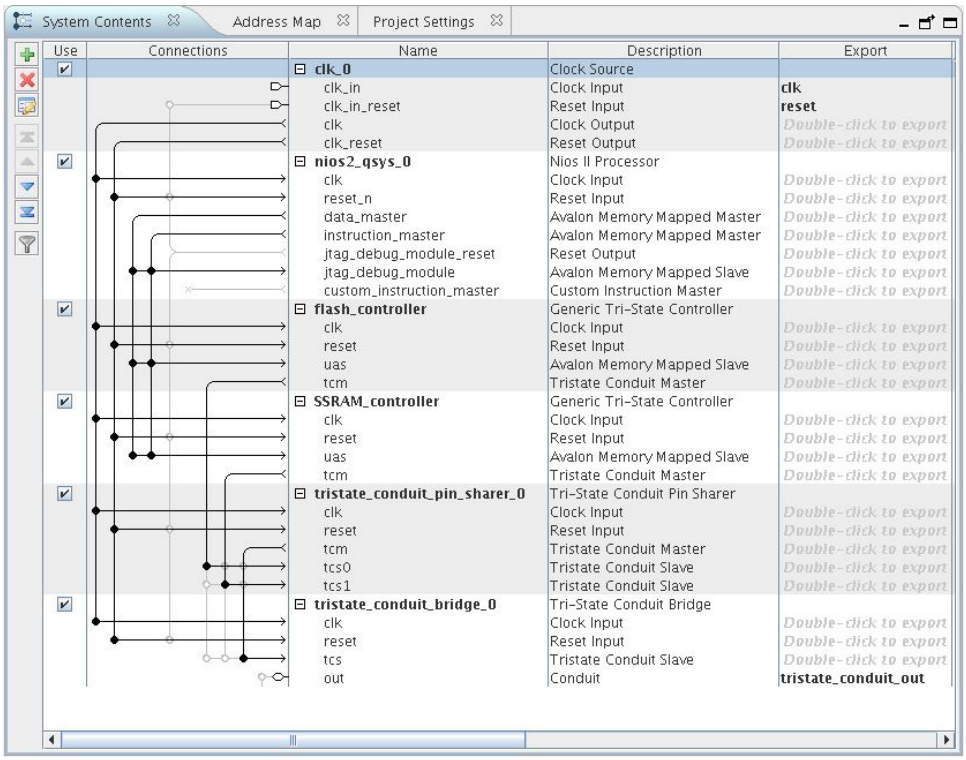
The flash device responds to address range 0 MB to 8 MB-1. The SSRAM responds to address range 8 MB to 10 MB-1. The PCB schematic for the PCB connects `addr[21:0]` to `addr[18:0]` of the SSRAM device because the SSRAM responds to 32-bit word address. The 8 MB flash device accesses 16-bit words; consequently, the schematic does not connect `addr[0]`. The `chipselect` signals select between the two devices.

Figure 133. Address Connections from Platform Designer System to PCB



Note: If you create a custom tri-state conduit master with word aligned addresses, the Tri-state Conduit Pin Sharer does not change or align the address signals.

Figure 134. Tri-State Conduit System in Platform Designer



Related Information

- Avalon Tri-State Conduit Components User Guide
- Avalon Interface Specifications

4.3.1. Generic Tri-State Controller

The Generic Tri-State Controller provides a template for a controller. You can customize the tri-state controller with various parameters to reflect the behavior of an off-chip device. The following types of parameters are available for the 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:

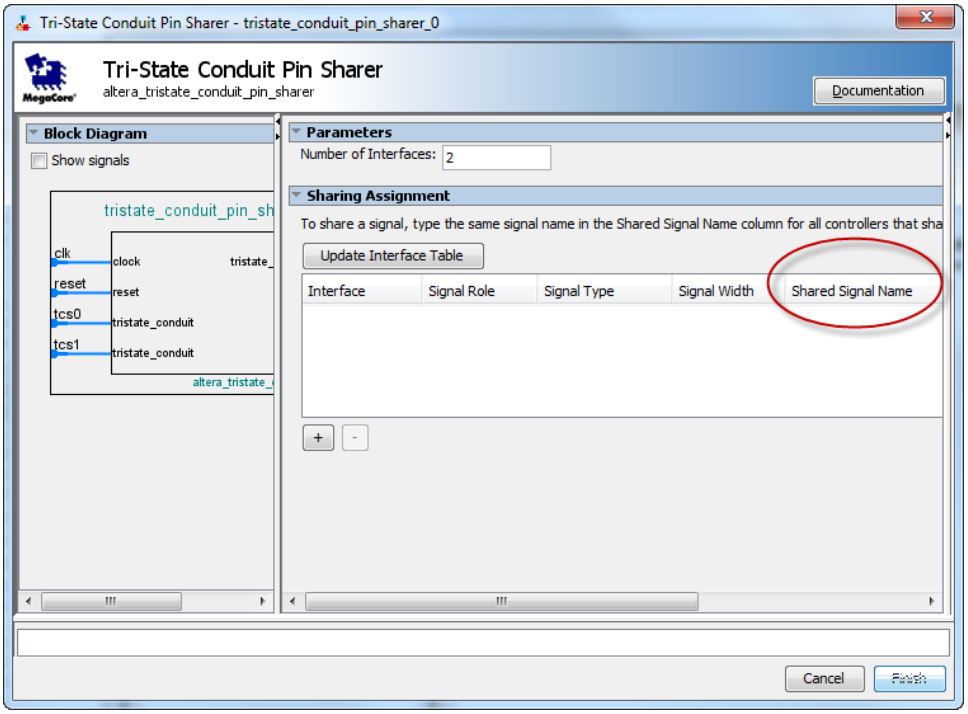
- **Memory-mapped slave interface**—This interface connects to a memory-mapped master, such as a processor.
- **Tristate Conduit Master interface**—The tri-state master interface usually connects to the tri-state conduit slave interface of the tri-state conduit pin sharer.
- **Clock sink**—The component's clock reference. You must connect this interface to a clock source.
- **Reset sink**—This interface connects to a reset source interface.

4.3.2. Tri-State Conduit Pin Sharer

The Tri-state Conduit Pin Sharer 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 and use the parameter editor to specify the signals that are shared.

Figure 135. Tri-State Conduit Pin Sharer Parameter Editor

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.



Note: All tri-state conduit components connected to a pin sharer must be in the same clock domain.

Related Information

Avalon-ST Round Robin Scheduler on page 270

4.3.3. Tri-State Conduit Bridge

The Tri-State Conduit Bridge 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.

4.4. Test Pattern Generator and Checker Cores

The test pattern generator inserts different error conditions, and the test pattern checker reports these error conditions to the control interface, each via an Avalon Memory-Mapped (Avalon-MM) slave.

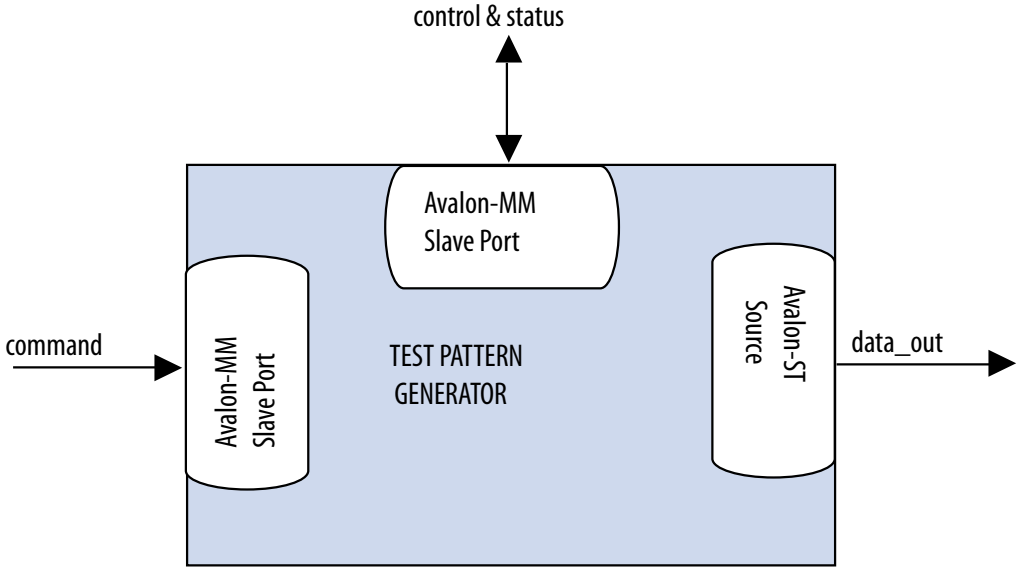
The data generation and monitoring solution for Avalon-ST consists of two components: a test pattern generator core that generates data, and sends it out on an Avalon-ST data interface, and a test pattern checker core that receives the same data and verifies it. Optionally, the data can be formatted 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 test pattern generator and checker cores in a system.

4.4.1. Test Pattern Generator

Figure 136. Test Pattern Generator Core

The test pattern generator core accepts commands to generate data via an Avalon-MM command interface, and drives the generated data to an Avalon-ST data interface. You can parameterize most aspects of the Avalon-ST data interface, such as the number of error bits and data signal width, thus allowing you to test components with different interfaces.



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 test pattern generator has a throttle register that is set via the Avalon-MM control interface. The test pattern generator uses the value of the throttle register in conjunction with a pseudo-random number generator to throttle the data generation rate.

4.4.1.1. Test Pattern Generator Command Interface

The command interface for the Test Pattern Generator is a 32-bit Avalon-MM write slave that accepts data generation commands. It is connected to a 16-element deep FIFO, thus allowing a master peripheral to drive commands into the test pattern generator.

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.

4.4.1.2. Test Pattern Generator Control and Status Interface

The control and status interface of the Test Pattern Generator is a 32-bit Avalon-MM slave 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.

4.4.1.3. Test Pattern Generator Output Interface

The output interface of the Test Pattern Generator is an Avalon-ST 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 test pattern generator maintains an internal state for each channel.

You can configure the output interface of the test pattern generator with the following parameters:

- **Number of Channels**—Number of channels that the test pattern generator 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.

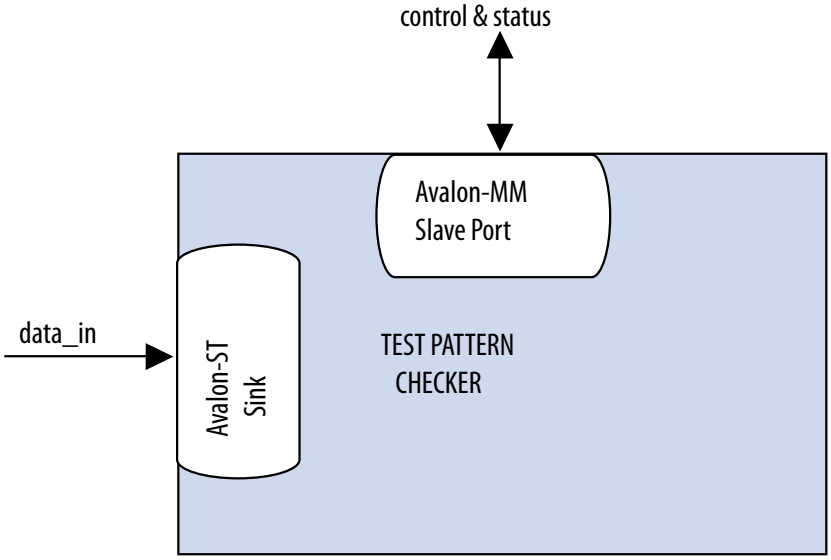
4.4.1.4. Test Pattern Generator Functional Parameter

The Test Pattern Generator functional parameter allows you to configure the test pattern generator as a whole system.

4.4.2. Test Pattern Checker

Figure 137. Test Pattern Checker

The test pattern checker core accepts data via an Avalon-ST interface and verifies it against the same predetermined pattern that the test pattern generator uses to produce the data. The test pattern checker core reports any exceptions to the control interface. You can parameterize most aspects of the test pattern checker's Avalon-ST interface such as the number of error bits and the data signal width. This enables the ability to test components with different interfaces. The test pattern checker has a throttle register that is set via the Avalon-MM control interface. The value of the throttle register controls the rate at which data is accepted.



The test pattern checker 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.

4.4.2.1. Test Pattern Checker Input Interface

The Test Pattern Checker input interface is an Avalon-ST 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.

4.4.2.2. Test Pattern Checker Control and Status Interface

The Test Pattern Checker control and status interface is a 32-bit Avalon-MM slave 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 test pattern checker supports data packets. The control and status interface also provides information on the exceptions detected by the test pattern checker. The interface obtains this information by reading from the exception FIFO.

4.4.2.3. Test Pattern Checker Functional Parameter

The Test Pattern Checker functional parameter allows you to configure the test pattern checker as a whole system.

4.4.2.4. Test Pattern Checker Input Parameters

You can configure the input interface of the test pattern checker 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.

4.4.3. Software Programming Model for the Test Pattern Generator and Checker Cores

The HAL system library support, software files, and register maps describe the software programming model for the test pattern generator and checker cores.

4.4.3.1. HAL System Library Support

For Nios II processor users, Intel provides HAL system library drivers that allow you to initialize and access the test pattern generator and checker cores. Intel recommends you use the provided drivers to access the cores 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.

4.4.3.2. Test Pattern Generator and Test Pattern Checker Core Files

The following files define the low-level access to the hardware, and provide the routines for the HAL device drivers.

Note: Do not modify the test pattern generator or test pattern checker core files.

- Test pattern generator core files:
  - **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.
- Test pattern checker core files:
  - **data_sink_regs.h**—Header file that defines the core's 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.

4.4.3.3. Register Maps for the Test Pattern Generator and Test Pattern Checker Cores

4.4.3.3.1. Test Pattern Generator Control and Status Registers

Table 99. Test Pattern Generator Control and Status Register Map

Shows the offset for the test pattern generator control and status registers. Each register is 32-bits wide.

Offset	Register Name
base + 0	`status`
base + 1	`control`
base + 2	`fill`

Table 100. Test Pattern Generator 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 101. Test Pattern Generator Control Register Bits

Bits	Name	Access	Description
[0]	`ENABLE`	RW	Setting this bit to 1 enables the test pattern generator core.
[7:1]	Reserved		
[16:8]	`THROTTLE`	RW	Specifies the throttle value which can be between 0–256, inclusively. The test pattern generator 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 102. Test Pattern Generator 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		

4.4.3.3.2. Test Pattern Generator Command Registers

Table 103. Test Pattern Generator 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 104. 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 test pattern generator core 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 test pattern generator 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 105. 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.

4.4.3.3.3. Test Pattern Checker Control and Status Registers

Table 106. Test Pattern Checker 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 107. Test Pattern Checker Status Register Bits

Bit(s)	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 108. Test Pattern Checker Control Register Bits

Bit(s)	Name	Access	Description
[0]	`ENABLE`	RW	Setting this bit to 1 enables the test pattern checker.
[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 test pattern generator core. Setting it to 256 causes the test pattern generator core 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 109. `exception_descriptor` Register Bits

Bit(s)	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.

continued...

Bit(s)	Name	Access	Description
[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 110. 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 111. 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.

.

4.4.4. Test Pattern Generator API

The following subsections describe application programming interface (API) for the test pattern generator.

Note: API functions are currently not available from the interrupt service routine (ISR).

data_source_reset() on page 255
data_source_init() on page 255
data_source_get_id() on page 255
data_source_get_supports_packets() on page 256
data_source_get_num_channels() on page 256
data_source_get_symbols_per_cycle() on page 256
data_source_get_enable() on page 256
data_source_set_enable() on page 257
data_source_get_throttle() on page 257
data_source_set_throttle() on page 257

data_source_is_busy() on page 258
data_source_fill_level() on page 258
data_source_send_data() on page 258

4.4.4.1. data_source_reset()

Table 112. 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 slave.
Returns	`void`
Description	Resets the test pattern generator core including all internal counters and FIFOs. The control and status registers are not reset by this function.

4.4.4.2. data_source_init()

Table 113. 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 slave. `command_base`—Base address of the command slave.
Returns	`1`—Initialization is successful. `0`—Initialization is unsuccessful.
Description	Performs the following operations to initialize the test pattern generator core: • Resets and disables the test pattern generator core. • Sets the maximum throttle. • Clears all inserted errors.

4.4.4.3. data_source_get_id()

Table 114. 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 slave.
Returns	Test pattern generator core identifier.
Description	Retrieves the test pattern generator core's identifier.

4.4.4.4. data_source_get_supports_packets()

Table 115. 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 slave.
Returns	`1`—Data packets are supported. `0`—Data packets are not supported.
Description	Checks if the test pattern generator core supports data packets.

4.4.4.5. data_source_get_num_channels()

Table 116. 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 slave.
Returns	Number of channels supported.
Description	Retrieves the number of channels supported by the test pattern generator core.

4.4.4.6. data_source_get_symbols_per_cycle()

Table 117. 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 slave.
Returns	Number of symbols transferred in a beat.
Description	Retrieves the number of symbols transferred by the test pattern generator core in each beat.

4.4.4.7. data_source_get_enable()

Table 118. data_source_get_enable()

Information Type	Description
Prototype	`int data_source_get_enable(alt_u32 base);`
Thread-safe	Yes

continued...

Information Type	Description
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	Value of the ENABLE bit.
Description	Retrieves the value of the ENABLE bit.

4.4.4.8. data_source_set_enable()

Table 119. data_source_set_enable()

Information Type	Description
Prototype	`void data_source_set_enable(alt_u32 base, alt_u32 value);`
Thread-safe	No
Include	*<data_source_util.h >*
Parameters	base—Base address of the control and status slave. value— ENABLE bit set to the value of this parameter.
Returns	void
Description	Enables or disables the test pattern generator core. When disabled, the test pattern generator core stops data transmission but continues to accept commands and stores them in the FIFO

4.4.4.9. data_source_get_throttle()

Table 120. 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 slave.
Returns	Throttle value.
Description	Retrieves the current throttle value.

4.4.4.10. data_source_set_throttle()

Table 121. 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 slave.

continued...

Information Type	Description
	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 test pattern generator sends data.

4.4.4.11. data_source_is_busy()

Table 122. data_source_is_busy()

Information Type	Description
Prototype	int data_source_is_busy(alt_u32 base);
Thread-safe	Yes
Include	<data_source_util.h>
Parameters	base—Base address of the control and status slave.
Returns	1—Test pattern generator core is busy. 0—Test pattern generator core is not busy.
Description	Checks if the test pattern generator is busy. The test pattern generator core is busy when it is sending data or has data in the command FIFO to be sent.

4.4.4.12. data_source_fill_level()

Table 123. 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 slave.
Returns	Number of commands in the command FIFO.
Description	Retrieves the number of commands currently in the command FIFO.

4.4.4.13. data_source_send_data()

Table 124. 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 slave. channel—Channel to send the data.

continued...

Information Type	Description
	`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.

4.4.5. Test Pattern Checker API

The following subsections describe API for the test pattern checker core. The API functions are currently not available from the ISR.

data_sink_reset() on page 260
data_sink_init() on page 260
data_sink_get_id() on page 260
data_sink_get_supports_packets() on page 261
data_sink_get_num_channels() on page 261
data_sink_get_symbols_per_cycle() on page 261
data_sink_get_enable() on page 261
data_sink_set enable() on page 262
data_sink_get_throttle() on page 262
data_sink_set_throttle() on page 262
data_sink_get_packet_count() on page 263
data_sink_get_error_count() on page 263
data_sink_get_symbol_count() on page 263
data_sink_get_exception() on page 264
data_sink_exception_is_exception() on page 264
data_sink_exception_has_data_error() on page 264
data_sink_exception_has_missing_sop() on page 265
data_sink_exception_has_missing_eop() on page 265
data_sink_exception_signalled_error() on page 265
data_sink_exception_channel() on page 266

4.4.5.1. data_sink_reset()

Table 125. 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 slave.
Returns	`void`
Description	Resets the test pattern checker core including all internal counters.

4.4.5.2. data_sink_init()

Table 126. 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 slave.
Returns	`1`—Initialization is successful. `0`—Initialization is unsuccessful.
Description	Performs the following operations to initialize the test pattern checker core: • Resets and disables the test pattern checker core. • Sets the throttle to the maximum value.

4.4.5.3. data_sink_get_id()

Table 127. 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 slave.
Returns	Test pattern checker core identifier.
Description	Retrieves the test pattern checker core's identifier.

4.4.5.4. data_sink_get_supports_packets()

Table 128. 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 slave.
Returns	`1`—Data packets are supported. `0`—Data packets are not supported.
Description	Checks if the test pattern checker core supports data packets.

4.4.5.5. data_sink_get_num_channels()

Table 129. 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 slave.
Returns	Number of channels supported.
Description	Retrieves the number of channels supported by the test pattern checker core.

4.4.5.6. data_sink_get_symbols_per_cycle()

Table 130. 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 slave.
Returns	Number of symbols received in a beat.
Description	Retrieves the number of symbols received by the test pattern checker core in each beat.

4.4.5.7. data_sink_get_enable()

Table 131. data_sink_get_enable()

Information Type	Description
Prototype	`int data_sink_get_enable(alt_u32 base);`
Thread-safe	Yes

continued...

Information Type	Description
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave.
Returns	Value of the ENABLE bit.
Description	Retrieves the value of the ENABLE bit.

4.4.5.8. data_sink_set enable()

Table 132. data_sink_set enable()

Information Type	Description
Prototype	`void data_sink_set_enable(alt_u32 base, alt_u32 value);`
Thread-safe	No
Include	*<data_sink_util.h >*
Parameters	base—Base address of the control and status slave. value—ENABLE bit is set to the value of the parameter.
Returns	void
Description	Enables the test pattern checker core.

4.4.5.9. data_sink_get_throttle()

Table 133. 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 slave.
Returns	Throttle value.
Description	Retrieves the throttle value.

4.4.5.10. data_sink_set_throttle()

Table 134. 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 slave.

continued...

Information Type	Description
	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 test pattern checker receives data.

4.4.5.11. data_sink_get_packet_count()

Table 135. 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 slave. channel—Channel number.
Returns	Number of data packets received on the channel.
Description	Retrieves the number of data packets received on a channel.

4.4.5.12. data_sink_get_error_count()

Table 136. 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 slave. channel—Channel number.
Returns	Number of errors received on the channel.
Description	Retrieves the number of errors received on a channel.

4.4.5.13. data_sink_get_symbol_count()

Table 137. 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 slave.

continued...

Information Type	Description
	channel—Channel number.
Returns	Number of symbols received on the channel.
Description	Retrieves the number of symbols received on a channel.

4.4.5.14. data_sink_get_exception()

Table 138. 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 slave.
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.

4.4.5.15. data_sink_exception_is_exception()

Table 139. 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.

4.4.5.16. data_sink_exception_has_data_error()

Table 140. 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.

4.4.5.17. data_sink_exception_has_missing_sop()

Table 141. 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.

4.4.5.18. data_sink_exception_has_missing_eop()

Table 142. 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.

4.4.5.19. data_sink_exception_signalled_error()

Table 143. 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.

4.4.5.20. data_sink_exception_channel()

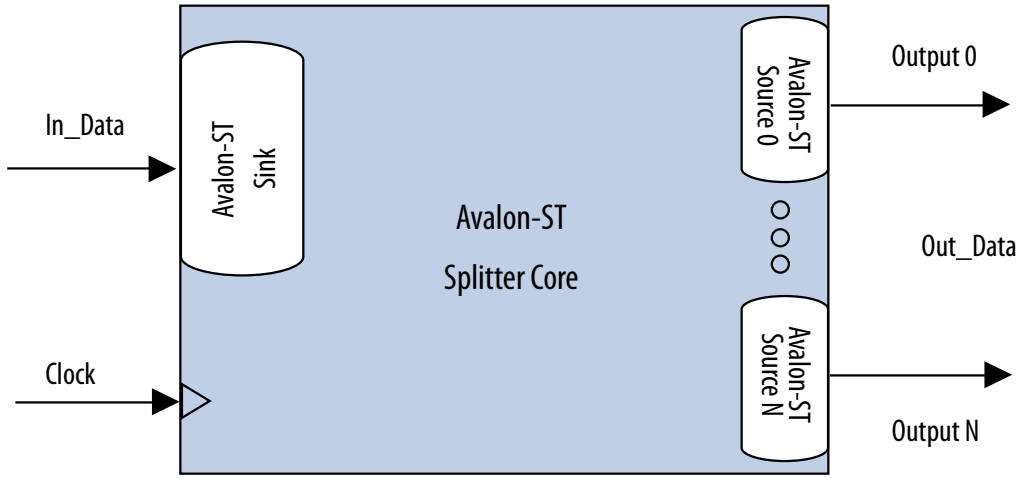
Table 144. data_sink_exception_channel()

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.

4.5. Avalon-ST Splitter Core

Figure 138. Avalon-ST Splitter Core

The Avalon-ST Splitter Core allows you to replicate transactions from an Avalon-ST sink interface to multiple Avalon-ST source interfaces. This core supports from 1 to 16 outputs.



The Avalon-ST Splitter core 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 core includes a clock signal to determine the Avalon-ST interface and clock domain where the core resides. Because the splitter core does not use the `clock` signal internally, latency is not introduced when using this core.

4.5.1. Splitter Core Backpressure

The Avalon-ST Splitter core 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 core introduces no latency.

4.5.2. Splitter Core Interfaces

The Avalon-ST Splitter core supports streaming data, with optional packet, channel, and error signals. The core propagates backpressure from any output interface to the input interface.

Table 145. Avalon-ST Splitter Core Support

Feature	Support
Backpressure	Ready latency = 0.
Data Width	Configurable.
Channel	Supported (optional).
Error	Supported (optional).
Packet	Supported (optional).

4.5.3. Splitter Core Parameters

Table 146. Avalon-ST Splitter Core 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-ST 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.

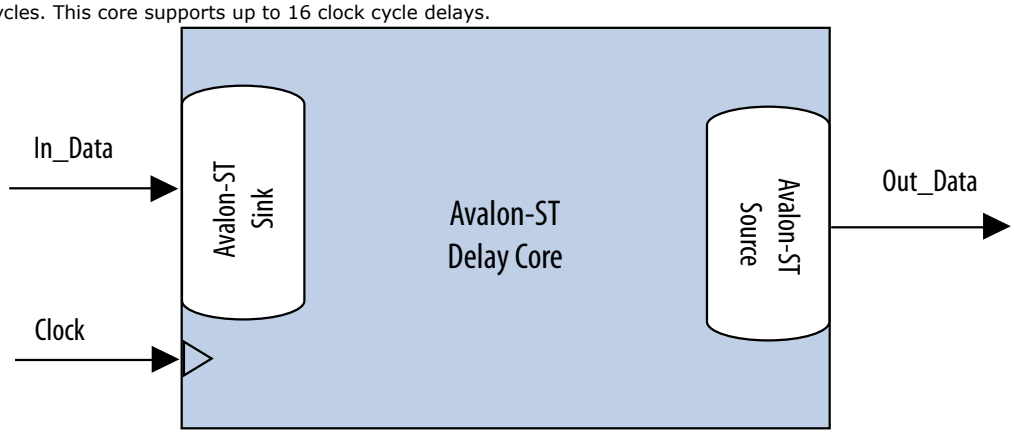
continued...

Parameter	Legal Values	Default Value	Description
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 splitter core is not using the `error` signal. This parameter is disabled when **Use Error** is set to `0`.

4.6. Avalon-ST Delay Core

Figure 139. Avalon-ST Delay Core

The Avalon-ST Delay Core provides a solution to delay Avalon-ST transactions by a constant number of clock cycles. This core supports up to 16 clock cycle delays.



The Avalon-ST Delay core adds a delay between the input and output interfaces. The core 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.

4.6.1. Delay Core Reset Signal

The Avalon-ST Delay core has a `reset` signal that is synchronous to the `clk` signal. When the core 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.

4.6.2. Delay Core Interfaces

The Delay core supports streaming data, with optional packet, channel, and error signals. The delay core does not support backpressure.

Table 147. Avalon-ST Delay Core Support

Feature	Support
Backpressure	Not supported.
Data Width	Configurable.
Channel	Supported (optional).
Error	Supported (optional).
Packet	Supported (optional).

4.6.3. Delay Core Parameters

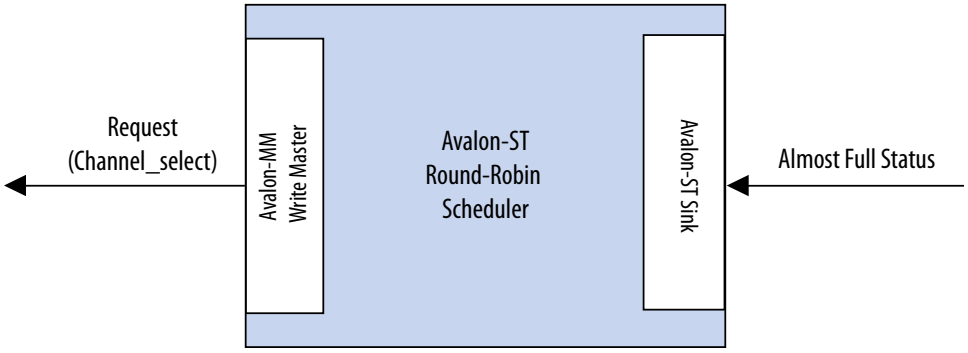
Table 148. Avalon-ST Delay Core Parameters

Parameter	Legal Values	Default Value	Description
Number Of Delay Clocks	0 to 16	1	Specifies the delay the core 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-ST 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.

4.7. Avalon-ST Round Robin Scheduler

Figure 140. Avalon-ST Round Robin Scheduler

The Avalon-ST Round Robin Scheduler core controls the read operations from a multi-channel Avalon-ST component that buffers data by channels. It reads the almost-full threshold values from the multiple channels in the multi-channel component and issues the read request to the Avalon-ST source according to a round-robin scheduling algorithm.



In a multi-channel component, the component 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.

4.7.1. Almost-Full Status Interface (Round Robin Scheduler)

The Almost-Full Status interface is an Avalon-ST sink interface that collects the almost-full status from the sink components for the channels in the sequence provided.

Table 149. Avalon-ST 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

4.7.2. Request Interface (Round Robin Scheduler)

The Request Interface is an Avalon-MM write master interface that requests data from a specific channel. The Avalon-ST Round Robin Scheduler cycles through the channels it supports and schedules data to be read.

4.7.3. Round Robin Scheduler Operation

If a particular channel is almost full, the Avalon-ST 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-ST 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 150. Avalon-ST Round Robin Scheduler Ports

Signal	Direction	Description
Clock and Reset		
`clk`	In	Clock reference.
`reset_n`	In	Asynchronous active low reset.
Avalon-MM Request Interface		
`request_address (log2 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 slave cannot accept a new request.
Avalon-ST 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 (log2 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.

4.7.4. Round Robin Scheduler Parameters

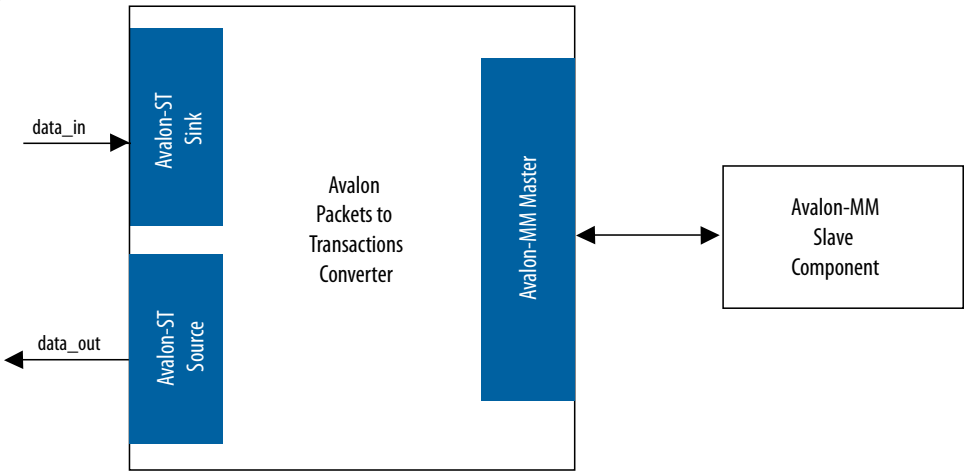
Table 151. Avalon-ST Round Robin Scheduler Parameters

Parameters	Legal Values	Default Value	Description
Number of channels	2–32	2	Specifies the number of channels the Avalon-ST Round Robin Scheduler supports.
Use almost-full status	Enabled, Disabled	Disabled	If enabled, the scheduler uses the almost-full interface. If not, the core requests data from the next channel at the next clock cycle.

4.8. Avalon Packets to Transactions Converter

Figure 141. Avalon Packets to Transactions Converter Core

The Avalon Packets to Transactions Converter core receives streaming data from upstream components and initiates Avalon-MM transactions. The core then returns Avalon-MM transaction responses to the requesting components.



Note: The SPI Slave to Avalon Master Bridge and JTAG to Avalon Master Bridge are examples of the Packets to Transactions Converter core. For more information, refer to the *Avalon Interface Specifications*.

Related Information

Avalon Interface Specifications

4.8.1. Packets to Transactions Converter Interfaces

Table 152. Properties of Avalon-ST 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-MM master interface supports read and write transactions. The data width is set to 32 bits, and burst transactions are not supported.

4.8.2. Packets to Transactions Converter Operation

The Packets to Transactions Converter core receives streams of packets on its Avalon-ST sink interface and initiates Avalon-MM transactions. Upon receiving transaction responses from Avalon-MM slaves, the core transforms the responses to packets and returns them to the requesting components via its Avalon-ST source interface. The core does not report Avalon-ST errors.

4.8.2.1. Packets to Transactions Converter Data Packet Formats

A response packet is returned for every write transaction. The core 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 core returns the data read.

The Packets to Transactions Converter core expects incoming data streams to be in the formats shown in the table below.

Table 153. 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

Packets to Transactions Converter Interfaces on page 272

4.8.2.2. Packets to Transactions Converter Supported Transactions

The Packets to Transactions Converter core supports the following Avalon-MM transactions:

Table 154. Packets to Transactions Converter Supported Transactions

Transaction Code	Avalon-MM 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-MM interface, the core still returns a response packet for this transaction code.

The Packets to Transactions Converter core can process only a single transaction at a time. The `ready` signal on the core's Avalon-ST 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-ST interface is forwarded directly to the Avalon-MM interface and vice-versa. Asserting the `waitrequest` signal on the Avalon-MM interface backpressures the Avalon-ST sink interface. In the opposite direction, if the Avalon-ST source interface is backpressured, the `read` signal on the Avalon-MM interface is not asserted until the backpressure is alleviated. Backpressuring the Avalon-ST source in the middle of a read can result in data loss. In this cases, the core returns the data that is successfully received.

A transaction is considered complete when the core 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 core always uses the end of packet (EOP) to determine the end of data.

4.8.2.3. Packets to Transactions 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 core 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 core processes unsupported transactions as a no transaction.

4.9. Avalon-ST Streaming Pipeline Stage

The Avalon-ST pipeline stage receives data from an Avalon-ST source interface, and outputs the data to an Avalon-ST sink interface. In the absence of back pressure, the Avalon-ST pipeline stage 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, it continues to assert its source interface's current data output. While the pipeline stage is receiving back pressure on its source interface and it 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 142. Pipeline Stage Simple Register

If the ready signal is not pipelined, the pipeline stage becomes a simple register.

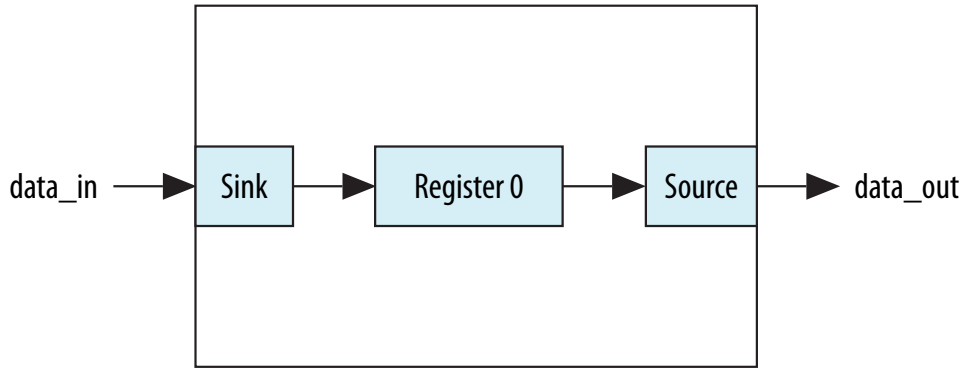
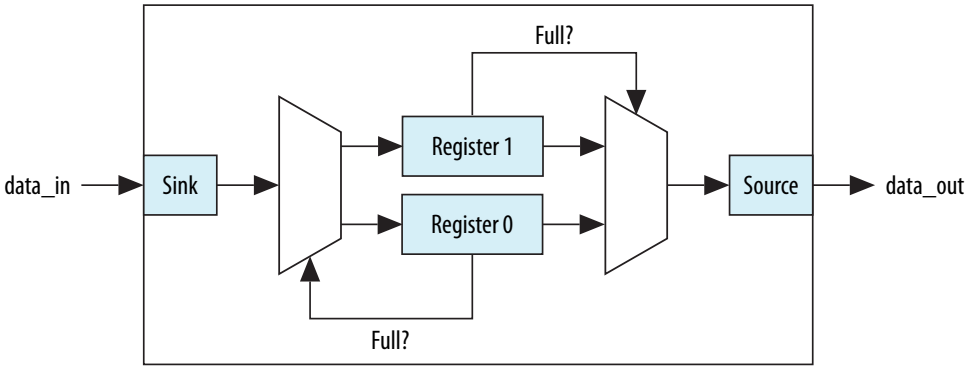


Figure 143. Pipeline Stage Holding Register

If the ready signal is pipelined, the pipeline stage must also include a second "holding" register.



4.10. Streaming Channel Multiplexer and Demultiplexer Cores

The Avalon-ST channel multiplexer core 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-ST channel demultiplexer core 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 cores can transfer data between interfaces on cores that support unidirectional flow of data. The multiplexer and demultiplexer allow you to create multiplexed or demultiplexed datapaths without having to write custom HDL code. The multiplexer includes an Avalon-ST Round Robin Scheduler.

Related Information

Avalon-ST Round Robin Scheduler on page 270

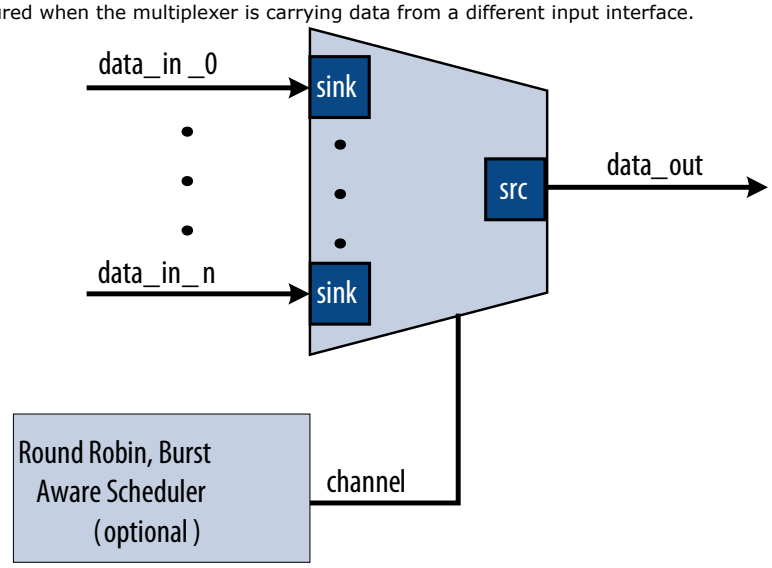
4.10.1. Software Programming Model For the Multiplexer and Demultiplexer Components

The multiplexer and demultiplexer 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. The components cannot generate interrupts.

4.10.2. Avalon-ST Multiplexer

Figure 144. Avalon-ST Multiplexer

The Avalon-ST multiplexer 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.



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.

4.10.2.1. Multiplexer Input Interfaces

Each input interface is an Avalon-ST data interface that optionally supports packets. The input interfaces are identical; they have the same symbol and data widths, error widths, and channel widths.

4.10.2.2. Multiplexer 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.

4.10.2.3. Multiplexer Parameters

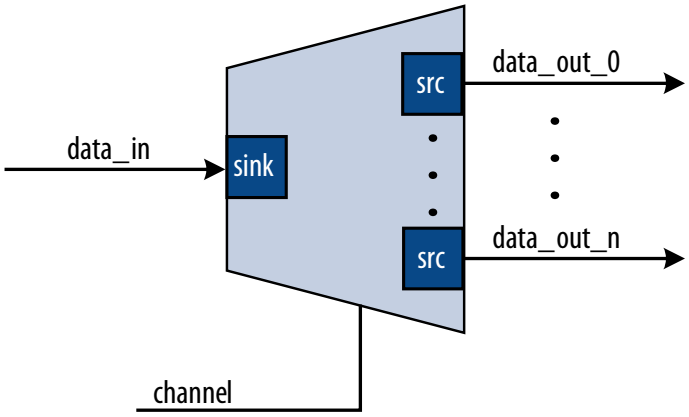
You can configure the following parameters for the multiplexer:

- **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.

4.10.3. Avalon-ST Demultiplexer

Figure 145. Avalon-ST Demultiplexer

That Avalon-ST demultiplexer 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.



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.

4.10.3.1. Demultiplexer Input Interface

Each input interface is an Avalon-ST 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.

4.10.3.2. Demultiplexer 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.

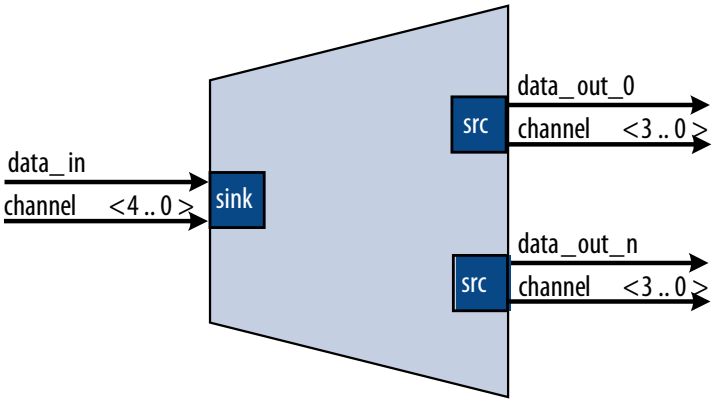
4.10.3.3. Demultiplexer Parameters

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 146. Select Bits for the Demultiplexer



4.11. Single-Clock and Dual-Clock FIFO Cores

The Avalon-ST Single-Clock and Avalon-ST Dual-Clock FIFO cores are FIFO buffers which operate with a common clock and independent clocks for input and output ports respectively.

Figure 147. Avalon-ST Single Clock FIFO Core

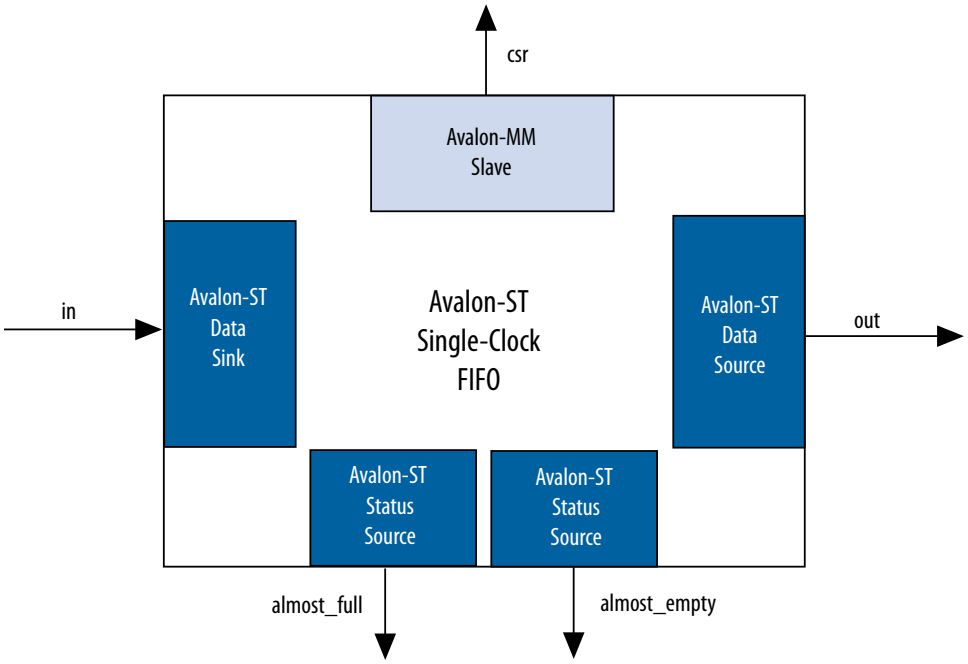
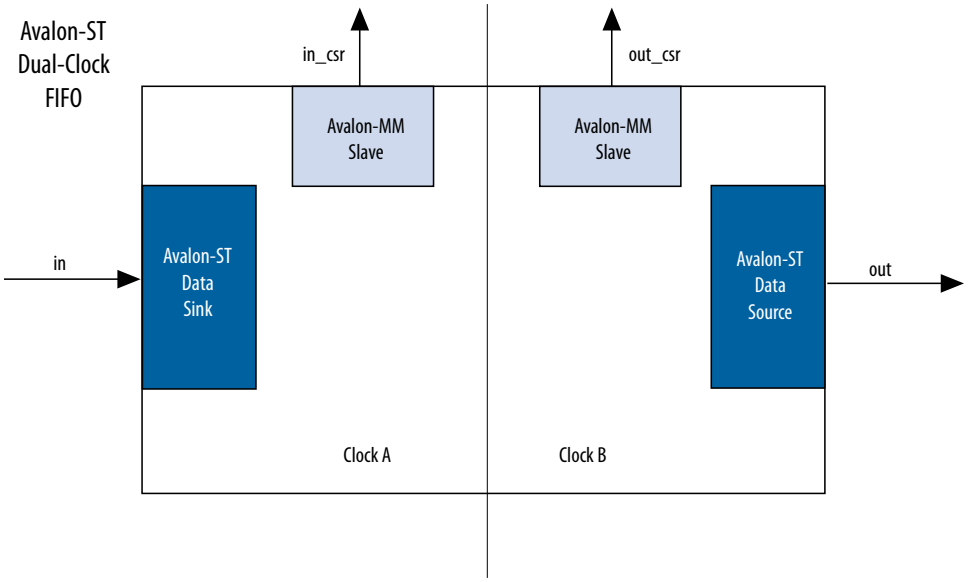


Figure 148. Avalon-ST Dual Clock FIFO Core



4.11.1. Interfaces Implemented in FIFO Cores

The following interfaces are implemented in FIFO cores:

Avalon-ST Data Interface on page 281
Avalon-MM Control and Status Register Interface on page 281

Avalon-ST Status Interface on page 281

4.11.1.1. Avalon-ST Data Interface

Each FIFO core has an Avalon-ST data sink and source interfaces. The data sink and source interfaces in the dual-clock FIFO core are driven by different clocks.

Table 155. Avalon-ST Interfaces Properties

Feature	Property
Backpressure	Ready latency = 0.
Data Width	Configurable.
Channel	Supported, up to 255 channels.
Error	Configurable.
Packet	Configurable.

4.11.1.2. Avalon-MM Control and Status Register Interface

You can configure the single-clock FIFO core to include an optional Avalon-MM interface, and the dual-clock FIFO core to include an Avalon-MM interface in each clock domain. The Avalon-MM 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 core, you can also configure the packet and error handling modes.

4.11.1.3. Avalon-ST Status Interface

The single-clock FIFO core has two optional Avalon-ST status source interfaces from which you can obtain the FIFO buffer almost-full and almost empty statuses.

4.11.2. FIFO Operating Modes

- **Default mode**—The core accepts incoming data on the `in` interface (Avalon-ST data sink) and forwards it to the `out` interface (Avalon-ST data source). The core asserts the `valid` signal on the Avalon-ST source interface to indicate that data is available at the interface.
- **Store and forward mode**—This mode applies only to the single-clock FIFO core. The core 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 core drops all packets received with the `in_error` signal asserted.
- **Cut-through mode**—This mode applies only to the single-clock FIFO core. The core 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.

4.11.3. Fill Level of the FIFO Buffer

You can obtain the fill level of the FIFO buffer via the optional Avalon-MM control and status interface. Turn on the **Use fill level** parameter (**Use sink fill level** and **Use source fill level** in the dual-clock FIFO core) and read the `fill_level` register.

The dual-clock FIFO core 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 dual-clock FIFO 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).

4.11.4. 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 overflow and underflow. This feature is available only in the single-clock FIFO core. 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-ST status source). The core asserts the `almost_full` signal when the fill level is equal to or higher than the almost-full threshold. Likewise, the core asserts the `almost_empty` signal when the fill level is equal to or lower than the almost-empty threshold.

4.11.5. Single-Clock and Dual-Clock FIFO Core Parameters

Table 156. Single-Clock and Dual-Clock FIFO Core Parameters

Parameter	Legal Values	Description
Bits per symbol	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.
Symbols per beat	1–32	
Error width	0–32	The width of the `error` signal.
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.
Use packets	—	Turn on this parameter to enable data packet support on the Avalon-ST data interfaces.
Channel width	1–32	The width of the `channel` signal.
Avalon-ST Single Clock FIFO Only		

continued...

Parameter	Legal Values	Description
Use fill level	—	Turn on this parameter to include the Avalon-MM control and status register interface (CSR). The CSR is enabled when **Use fill level** is set to 1.
Use Store and Forward		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.
Avalon-ST Dual Clock FIFO Only		
Use sink fill level	—	Turn on this parameter to include the Avalon-MM control and status register interface in the input clock domain.
Use source fill level	—	Turn on this parameter to include the Avalon-MM 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 core.
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.
Use Max Channel	—	Turn on this parameter to specify the maximum channel number.
Max Channel	1–255	Maximum channel number.

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*.

Related Information

- Managing Metastability with the Software
- Understanding Metastability in FPGAs

4.11.6. Avalon-ST Single-Clock FIFO Registers

Table 157. Avalon-ST Single-Clock FIFO Registers

The CSR interface in the Avalon-ST Single Clock FIFO core 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.

continued...

32-Bit Word Offset	Name	Access	Reset	Description
				Greater than 0—Enables cut-through mode and specifies the minimum of entries in the FIFO buffer before the `valid` signal on the Avalon-ST source interface is asserted. Once the FIFO core 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 158. Register Description for Avalon-ST Dual-Clock FIFO

The `in_csr` and `out_csr` interfaces in the Avalon-ST Dual Clock FIFO core 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

4.12. Platform Designer System Design Components Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2017.11.06	17.1.0	• Changed instances of *Qsys* 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.
2016.05.03	16.0.0	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	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Avalon-MM Unaligned Burst Expansion Bridge and Avalon-MM Pipeline Bridge, **Maximum pending read transactions** parameter. Extended description.

continued...

Document Version	Intel Quartus Prime Version	Changes
December 2014	14.1.0	• AXI Timeout Bridge. • Added notes to *Avalon-MM Clock Crossing Bridge* pertaining to: — SDC constraints for its internal asynchronous FIFOs. — FIFO-based clock crossing.
June 2014	14.0.0	• AXI Bridge support. • Address Span Extender updates. • Avalon-MM Unaligned Burst Expansion Bridge support.
November 2013	13.1.0	• Address Span Extender
May 2013	13.0.0	• Added Streaming Pipeline Stage support. • Added AMBA APB support.
November 2012	12.1.0	• Moved relevant content from the *Embedded Peripherals IP User Guide*.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

5. Creating Platform Designer Components

You can create a Hardware Component Definition File (`_hw.tcl`) to describe and package IP components for use in a Platform Designer system.

Note: Intel now refers to Qsys as Platform Designer (Standard).

A `_hw.tcl` describes IP components, interfaces and HDL files. Platform Designer provides the Component Editor to help you create a simple `_hw.tcl` file.

Refer to the *Demo AXI Memory* example on the Design Examples page for full code examples that appear in this chapter.

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

- Avalon Interface Specifications
- Protocol Specifications
- Demo AXI Memory Example

5.1. 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.
- Constraint files (both or either Synopsys* Design Constraints File `.sdc` and Intel Quartus Prime IP File `.qip`) that define the component for synthesis and simulation.
- A component's interfaces, including I/O signals.
- The parameters that configure the operation of the component.

5.1.1. 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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 159. IP Component Interface Types

Interface Type	Description
Memory-Mapped	Connects memory-referencing master devices with slave memory devices. Master devices can be processors and DMAs, while slave memory devices can be RAMs, ROMs, and control registers. Data transfers between master and slave may be uni-directional (read only or write only), or bi-directional (read and write).
Streaming	Connects Avalon Streaming (Avalon-ST) 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-ST 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

- Avalon Interface Specifications
- AMBA Protocol Specifications

5.1.2. Component Structure

Intel provides components automatically installed with the Intel Quartus Prime software. You can obtain a list of Platform Designer-compliant components provided by third-party IP developers on Altera's **Intellectual Property & Reference Designs** page by typing: **qsys certified** in the **Search** box, and then selecting **IP Core & Reference Designs**. Components are also provided with Intel development kits, which are listed on the **All Development Kits** page.

Every component is defined with a <*component_name*>_hw.tcl file, 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 three 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**—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.
- **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.

Related Information

- Create a Composed Component or Subsystem on page 316
- Add Component Instances to a Static or Generated Component on page 319

5.1.3. 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 II Software Developer's Handbook* for information about writing a device driver or software package suitable for use with the Nios II processor.

Related Information

Nios II Software Developer's Handbook
Refer to the "Nios II Software Build Tools" and "Overview of the Hardware Abstraction Layer" chapters.

5.1.4. 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>*.

5.1.4.1. 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 Intel Quartus Prime software maintains a list of all IP components associated with your design on the **Components** tab in the Project Navigator.

Related Information

Upgrade IP Components Dialog Box
In *Intel Quartus Prime Help*

5.2. 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

5.3. Create IP Components in the Platform Designer Component Editor

The Platform Designer Component Editor allows you to create and package an IP component. When you use the Component Editor to define a component, Platform Designer writes the information to an `_hw.tcl` file.

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 interfaces, signals, and parameters.
- Set parameters on interfaces and signals that can alter the component's structure or functionality.

If you add the top-level HDL file that defines the component on **Files** tab in the Platform Designer Component Editor, you must define the component's parameters and signals in the HDL file. You cannot add or remove them in the Component Editor.

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 11. 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

#
# connection point slave
#
add_interface slave axi end
set_interface_property slave associatedClock clock
set_interface_property slave associatedReset reset
set_interface_property slave readAcceptanceCapability 1
set_interface_property slave writeAcceptanceCapability 1
set_interface_property slave combinedAcceptanceCapability 1
set_interface_property slave readDataReorderingDepth 1
set_interface_property slave ENABLED true

add_interface_port slave axs_awid awid Input AXI_ID_W
...
add_interface_port slave axs_rresp rresp Output 2
```

Related Information

Component Interface Tcl Reference on page 467

5.3.1. 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.

Intel recommends that you move `_hw.tcl` files and their associated files to an `ip/` directory within your Intel Quartus Prime project directory. You can use IP components with other applications, such as the C compiler and a board support package (BSP) generator.

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 10
- Publishing Component Information to Embedded Software
  In *Nios II Gen 2 Software Developer's Handbook*

- Publishing Component Information to Embedded Software (Nios II Software Developer's Handbook)
- Creating a System with Platform Designer on page 10

5.3.2. 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 Intel 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.

5.4. Specify IP Component Type Information

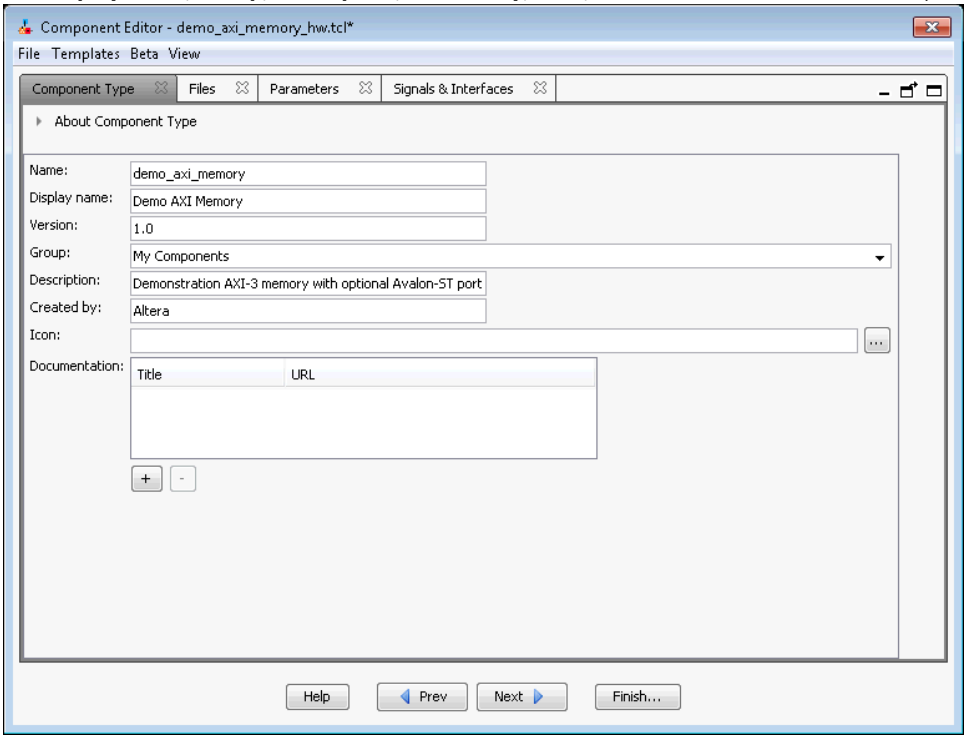
The **Component Type** tab in the Platform Designer Component Editor allows you to specify the following information about the component:

- **Name**—Specifies the name used in 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 and instance of the component, and also appears in the IP Catalog under **Project** and on the **System View** tab.
- **Version**—Specifies the version number of the component.
- **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 Intel 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**—Allows you to specify the author of the component.
- **Icon**—Allows you to enter 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**—Allows you to add 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 149. 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 Intel 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 ACDS releases.

Example 12. _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 Intel 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-ST 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 467

5.5. 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

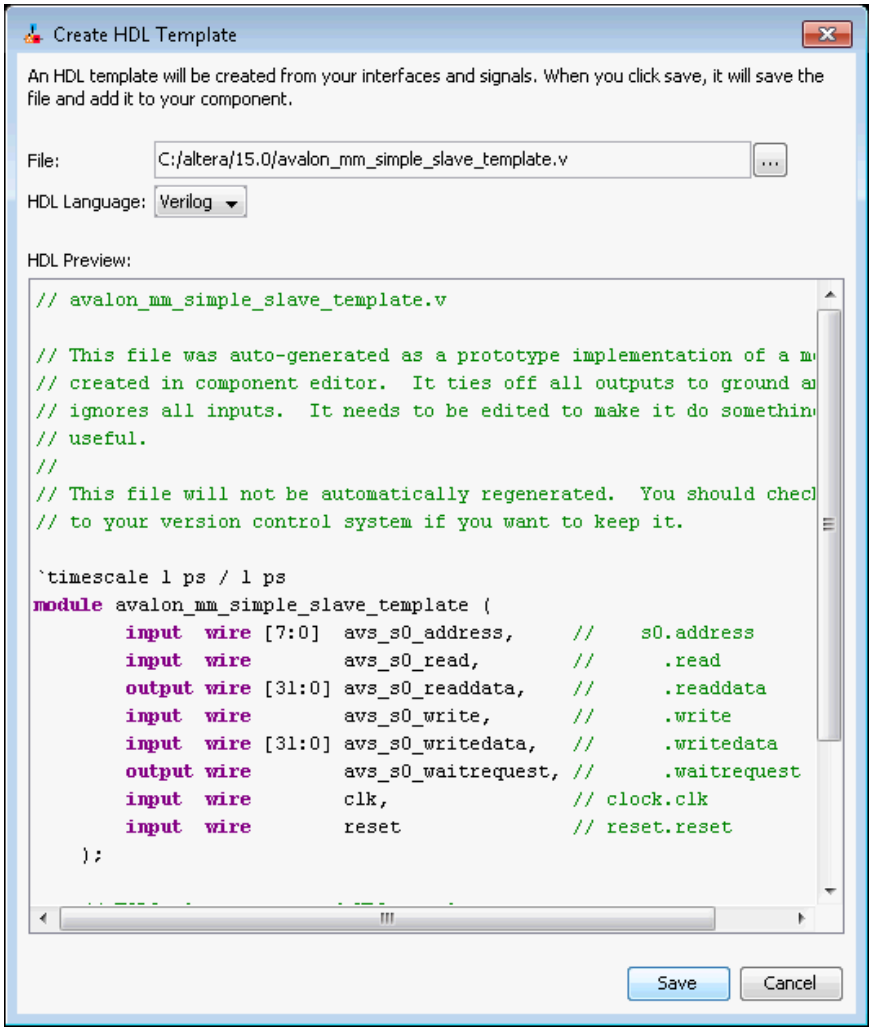
Specify Synthesis and Simulation Files in the Platform Designer Component Editor on page 297

5.6. 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 297

5.7. 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 Intel 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 295
- Create an HDL File Using a Template in the Platform Designer Component Editor on page 295

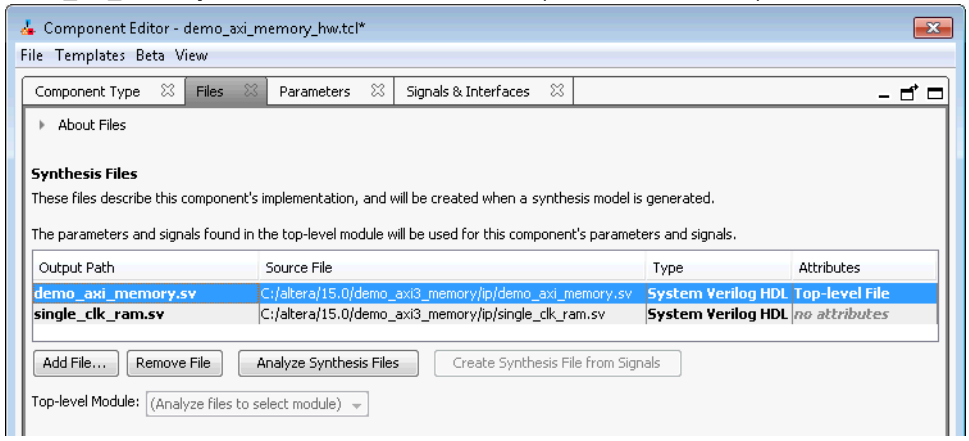
5.7.1. Specify HDL Files for Synthesis in the Platform Designer Component Editor

In the Platform Designer Component Editor, you can add HDL files and other support files with options on the `Files` tab.

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 Intel Quartus Prime software. The synthesis files for a component are copied to the generation output directory during Platform Designer system generation.

Figure 150. Using HDL Files to Define a Component

In the **Synthesis Files** section on the **Files** tab in the Platform Designer Component Editor, the **demo_axi_memory.sv** file should be selected as the top-level file for the component.



5.7.2. 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 13. _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 298
- Component Interface Tcl Reference on page 467

5.7.3. Name HDL Signals for Automatic Interface and Type Recognition in the Platform Designer Component Editor

If you create the component's top-level HDL file before using the Component Editor, the Component Editor recognizes the interface and signal types based on the signal names in the source HDL file. This auto-recognition feature eliminates the task of manually assigning each interface and signal type in the Component Editor.

To enable auto-recognition, you must create signal names using the following naming convention:

<interface type prefix>_<interface name>_<signal type>

Specifying an interface name with *<interface name>* is optional if you have only one interface of each type in the component definition. For interfaces with only one signal, such as clock and reset inputs, the *<interface type prefix>* is also optional.

Table 160. Interface Type Prefixes for Automatic Signal Recognition

When the Component Editor recognizes a valid prefix and signal type for a signal, it automatically assigns an interface and signal type to the signal based on the naming convention. If no interface name is specified for a signal, you can choose an interface name on the **Signals & Interfaces** tab in the Component Editor.

Interface Prefix	Interface Type
asi	Avalon-ST sink (input)
aso	Avalon-ST source (output)
avm	Avalon-MM master
avs	Avalon-MM slave
axm	AXI master
axs	AXI slave
apm	APB master
aps	APB slave
coe	Conduit
csi	Clock Sink (input)
cso	Clock Source (output)
inr	Interrupt receiver
ins	Interrupt sender
ncm	Nios II custom instruction master
ncs	Nios II custom instruction slave
rsi	Reset sink (input)

continued...

Interface Prefix	Interface Type
rso	Reset source (output)
tcm	Avalon-TC master
tcs	Avalon-TC slave

Refer to the *Avalon Interface Specifications* or the *AMBA Protocol Specification* for the signal types available for each interface type.

Related Information

- Avalon Interface Specifications
- Protocol Specifications

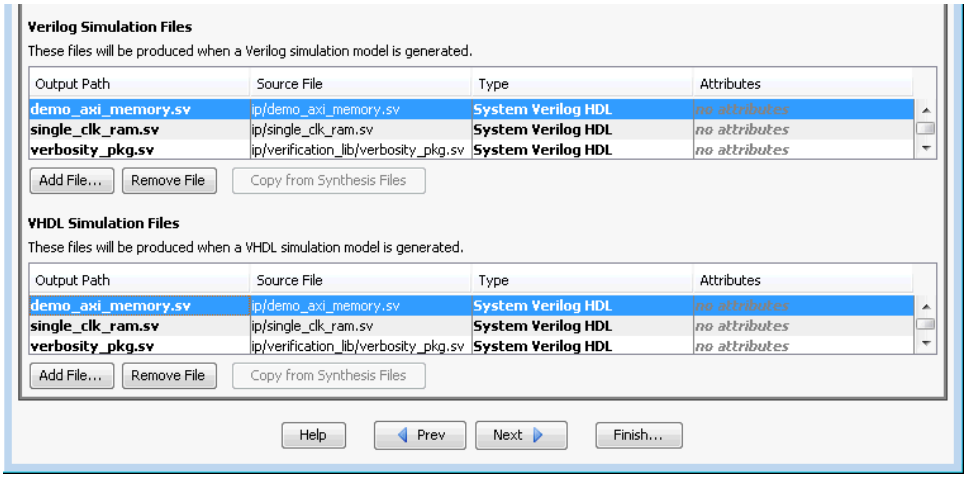
5.7.4. Specify Files for Simulation in the Component Editor

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 151. 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 14. _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 467

5.7.5. Include an Internal Register Map Description in the .svd for Slave Interfaces Connected to an HPS Component

Platform Designer supports the ability for IP component designers to specify register map information on their slave interfaces. This allows components with slave 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 slave interface using the following command:

`set_interface_property` <*slave 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 15. 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 467
- CMSIS - Cortex Microcontroller Software

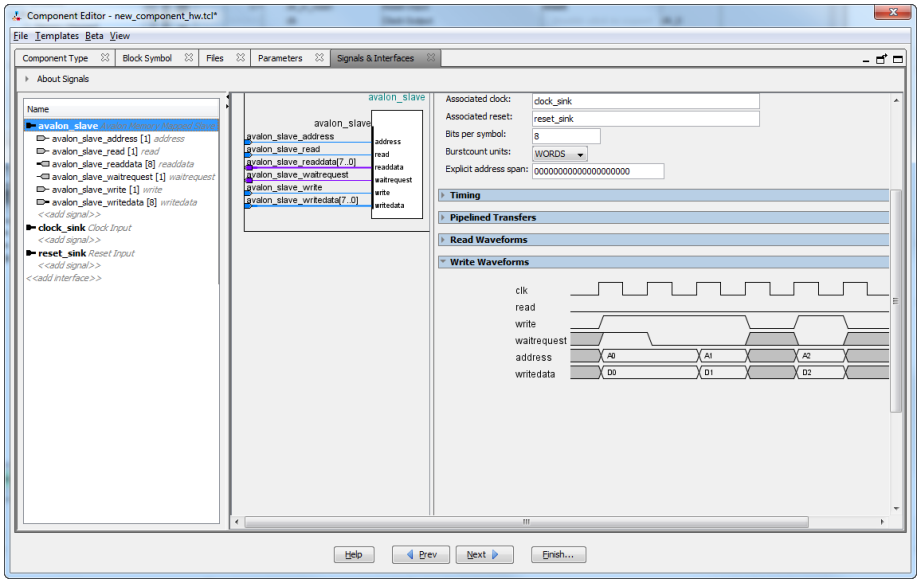
5.8. Add Signals and Interfaces in the Platform Designer Component Editor

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 nsignal 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 152. Platform Designer Signals & Interfaces tab



5.9. 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.

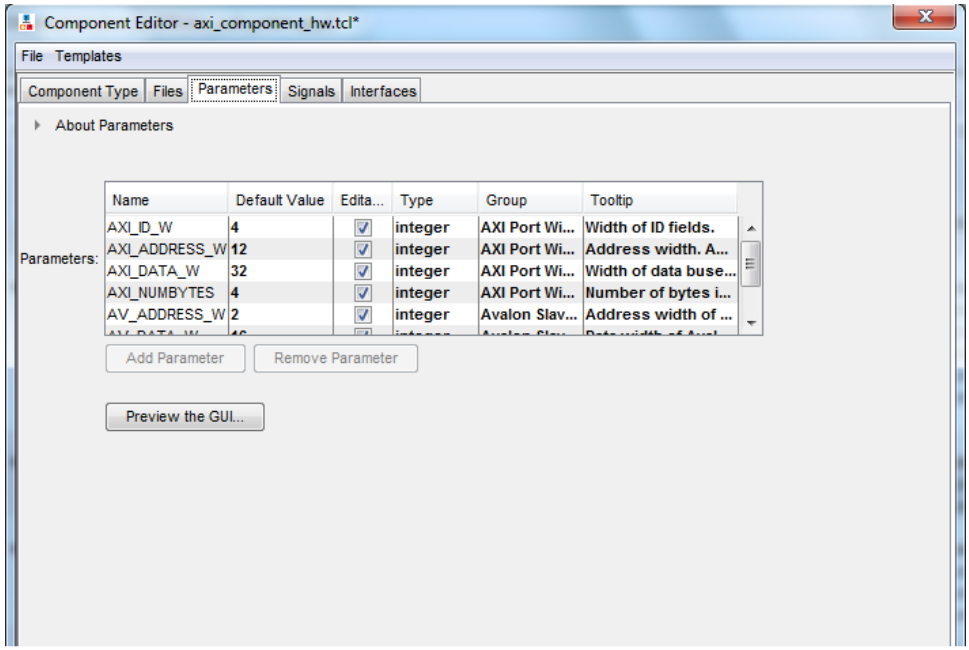
If you create a top-level template HDL file for synthesis with the Component Editor, you can remove the newly-created file from the **Synthesis Files** list on the **Files** tab, make your parameter changes, and then re-analyze the top-level synthesis file.

You can use the **Parameters** table to specify the following information about each parameter:

- **Name**—Specifies the name of the parameter.
- **Default Value**—Sets the default value for new instances of the component.
- **Editable**—Specifies whether or not the user can edit the parameter value.

- **Type**—Defines the parameter type as string, integer, boolean, std_logic, logic vector, natural, or positive.
- **Group**—Allows you to group parameters in parameter editor.
- **Tooltip**—Allows you to add a description of the parameter that appears when the user of the component points to the parameter in the editor.

Figure 153. 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 using radio buttons for parameter selections, or displaying an image.

Example 16. _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.

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 slave's ID bit width is smaller than required for your system, the AXI slave response may not reach all AXI masters. The formula of an AXI slave ID bit width is calculated as follows:

`maximum_master_id_width_in_the_interconnect` + log2 (number_of_masters_in_the_same_interconnect)

For example, if an AXI slave connects to three AXI masters and the maximum AXI master ID length of the three masters is 5 bits, then the AXI slave ID is 7 bits, and is calculated as follows:

```
5 bits + 2 bits (log2(3 masters)) = 7
```

Table 161. AXI Master and Slave Parameters

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.

AXI Master Parameters	AXI Slave Parameters
readIssuingCapability	readAcceptanceCapability
writeIssuingCapability	writeAcceptanceCapability
combinedIssuingCapability	combinedAcceptanceCapability
	readDataReorderingDepth

Related Information

Component Interface Tcl Reference on page 467

5.9.1. 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 a drop-down list or radio button 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 162. 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 308

5.9.2. 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 307
Platform Designer System Information Parameters on page 307
Platform Designer Derived Parameters on page 308

Related Information

Declare Parameters with Custom _hw.tcl Commands on page 308

5.9.2.1. 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*.

5.9.2.2. 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>
```

5.9.2.3. 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 308

5.9.2.3.1. 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.

5.9.3. 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-ST 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.

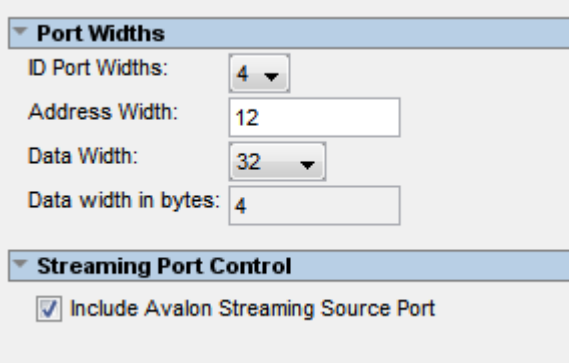
Example 17. Parameter Declaration

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 Slave 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-ST source (default),\
or hide the interface"

set_parameter_property ENABLE_STREAM_OUTPUT GROUP \
"Streaming Port Control"

...
```

Figure 154. Resulting Parameter Editor GUI from Parameter Declarations



Related Information

- Control Interfaces Dynamically with an Elaboration Callback on page 314
- Component Interface Tcl Reference on page 467

5.9.4. 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 18. 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 467
- Demo AXI Memory Example

5.10. 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 19. Example Part 1: Parameters, Platform Designer Interface, and Ports in _hw.tcl

```
# request TCL package from ACDS 17.1
#
package require -exact qsys 17.1


#
# 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
# Adding ports to avalon_slave interface
add_interface_port avalon_slave address address Input 10
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
set_interface_property avalon_slave associatedClock clk
set_interface_property avalon_slave associatedReset reset

#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
```

Part 2 defines the interface name, ports, and parameters of the SystemVerilog interface.

Example 20. 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

# 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
add_fileset_file mem_ifc.sv SYSTEM_VERILOG PATH mem_ifc.sv
SYSTEMVERILOG_INTERFACE
```

Related Information

SystemVerilog Interface Commands on page 553

5.11. User Alterable 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>
     ```

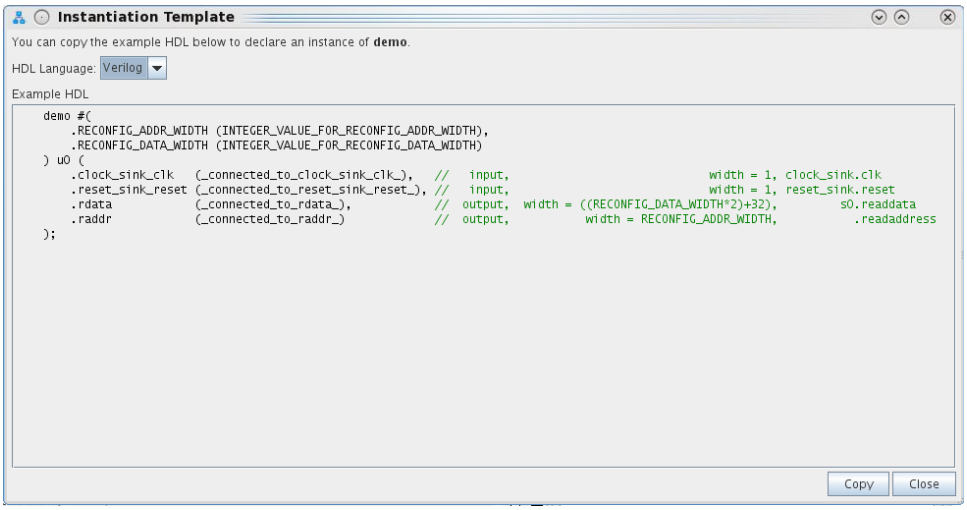
   - To set the width expression in the port property:

     ```
     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 155. Instantiation Template Dialog Box



The following sample contains _hw.tcl to set exportable width values:

Example 21. 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"
}
```

5.12. 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.

5.13. Control Interfaces Dynamically with an Elaboration Callback

You can allow user parameters to dynamically control your component's behavior with a 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 22. Avalon-ST Source Interface Optionally Included in a Component Specified with an Elaboration Callback

```
set_module_property ELABORATION_CALLBACK elaborate

proc elaborate {} {

   # Optionally disable the Avalon- ST 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 308
- Validate Parameter Values with a Validation Callback on page 310
- Component Interface Tcl Reference on page 467

5.14. 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 23. 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 297
- Component Interface Tcl Reference on page 467

5.15. 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.

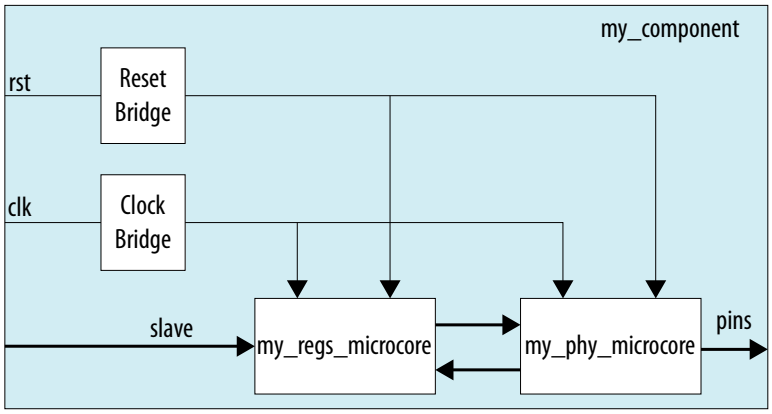
To connect instances of your component, you must define the component's interfaces. Unlike an HDL-based component, a composed component does not directly specify the signals that are exported. Instead, interfaces of submodules are chosen as the external interface, and each internal interface's ports 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 master 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 156. Top-Level of a Composed Component



Example 24. 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 slave avalon slave
  add_interface pins conduit end

  set_interface_property clk EXPORT_OF clk.in_clk
  set_instance_property_value reset synchronous_edges deassert
  set_interface_property reset EXPORT_OF reset.in_reset
  set_interface_property slave EXPORT_OF regs.slave
  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 467

5.16. Create an IP Component with Platform Designer a System View Different from the Generated Synthesis Output Files

There are cases where it may be beneficial to have the structural Platform Designer system view of a component differ from the generated synthesis output files. The structural composition callback allows you to define a structural hierarchy for a component separately from the generated output files.

One application of this feature is for IP designers who want to send out a placed-and-routed component that represents a Platform Designer system in order to ensure timing closure for their client or team-mate. In this case, the designer creates a design partition for the Platform Designer system, and then exports a post-fit Intel Quartus Prime Exported Partition File (**.qxp**) when satisfied with the placement and routing results.

The designer specifies a **.qxp** file as the generated synthesis output file for the new component. The designer can specify whether to use a simulation output fileset for the custom simulation model file, or to use simulation output files generated from the original Platform Designer system.

When the client or team-mate adds this component to their Platform Designer system, the designer wants the client or team-mate to see a structural representation of the component, including lower-level components and the address map of the original Platform Designer system. This structural view is a logical representation of the component that is used during the elaboration and validation phases in Platform Designer.

Example 25. Structural Composition Callback and .qxp File as the Generated Output

To specify a structural representation of the component for Platform Designer, connect components or generate a hardware Tcl description of the Platform Designer system, and then insert the Tcl commands into a structural composition callback. To invoke the structural composition callback use the command:

```
set_module_property STRUCTURAL_COMPOSITION_CALLBACK
structural_hierarchy
```

```
package require -exact qsys 14.0
set_module_property name example_structural_composition

set_module_property STRUCTURAL_COMPOSITION_CALLBACK \
structural_hierarchy

add_fileset synthesis_fileset QUARTUS_SYNTH \
synth_callback_procedure

add_fileset simulation_fileset SIM_VERILOG \
sim_callback_procedure

set_fileset_property synthesis_fileset TOP_LEVEL \
my_custom_component
```

```
set_fileset_property simulation_fileset TOP_LEVEL \
my_custom_component

proc structural_hierarchy {} {

# called during elaboration and validation phase
# exported ports should be same in structural_hierarchy
# and generated QXP

# These commands could come from the exported hardware Tcl

    add_interface clk clock sink
    add_interface reset reset sink

    add_instance clk_0 clock_source
    set_interface_property clk EXPORT_OF clk_0.clk_in
    set_interface_property reset EXPORT_OF clk_0.clk_in_reset

    add_instance pll_0 altera_pll
    # connections and connection parameters
    add_connection clk_0.clk pll_0.refclk clock
    add_connection clk_0.clk_reset pll_0.reset reset
}

proc synth_callback_procedure { entity_name } {

# the QXP should have the same name for ports
# as exportedin structural_hierarchy

 add_fileset_file my_custom_component.qxp QXP PATH \
 "my_custom_component.qxp"
}

proc sim_callback_procedure { entity_name } {

# the simulation files should have the same name for ports as
# exported in structural_hierarchy

add_fileset_file my_custom_component.v VERILOG PATH \
"my_custom_component.v"
 ….
 ….
}
```

Related Information

Create a Composed Component or Subsystem on page 316

5.17. 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 Components on page 320
- Generated Components on page 321

5.17.1. Static Components

Static 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 26. 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 27. 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
```

5.17.2. 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 28. 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 29. 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 315
- Intellectual Property & Reference Designs

5.17.3. Design Guidelines for Adding Component Instances

In order to promote standard and predictable results when generating static and generated components, Intel recommends the following 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.

5.18. Creating Platform Designer Components Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2018.05.07	18.0	• Added scripting support for wire-level expressions.
2017.11.06	17.1.0	• Changed instances of *Qsys* to *Platform Designer (Standard)* • Replaced mentions of altera_axi_default_slave to altera_error_response_slave
2017.05.08	17.0.0	• Updated Figure: Address Span Extender
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.

continued...

Document Version	Intel Quartus Prime Version	Changes
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.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

6. 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 Intel Quartus Prime installation directory:

```
<Intel Quartus Prime installation directory>\quartus\sopc_builder
\bin
```

For command-line help listing of all the options for any executable, type the following command:

```
<Intel 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.

6.1. 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 163. 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 "$", for example: `/extra/dir,$` .
`--family[=<value>]`	Optional	Sets the device family.
`--part[=<value>]`	Optional	Sets the device part number. If set, this option overrides the `--family` option.

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Option	Usage	Description
`--project-directory[=<directory>]`	Optional	Specifies the component locations relative to the project, if any. Default option is '.' (current directory). To exclude any project directory, use ''.
`--new-component-type[=<value>]`	Optional	Specifies the instance type for parameterization in a variation.
`--system-info[=<DEVICE_FAMILY|DEVICE_FEATURES|CLOCK_RATE|\CLOCK_DOMAIN|RESET_DOMAIN|CLOCK_RESET_INFO|ADDRESS_WIDTH|ADDRESS_MAP|MAX_SLAVE_DATA_WIDTH|\INTERRUPTS_USED|TRISTATECONDUIT_MASTERS|TRISTATECONDUIT_INFO|DEVICE|PART_TRAIT|\DEVICE_SPEEDGRADE|CUSTOM_INSTRUCTION_SLAVES|GENERATION_ID|UNIQUE_ID|\AVALON_SPEC|QUARTUS_INI|DESIGN_ENVIRONMENT>]`	Optional/ Repeatable	Specifies the value of a system info setting.
`--require-generation`	Optional	Marks the loading system as requiring generation.
`--debug`	Optional	Enables debugging features and output.
`--host-controller`	Optional	Specifies the instance type that you want to parameterize in a variation.
`--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`.
`--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 157. Internal Menu Options

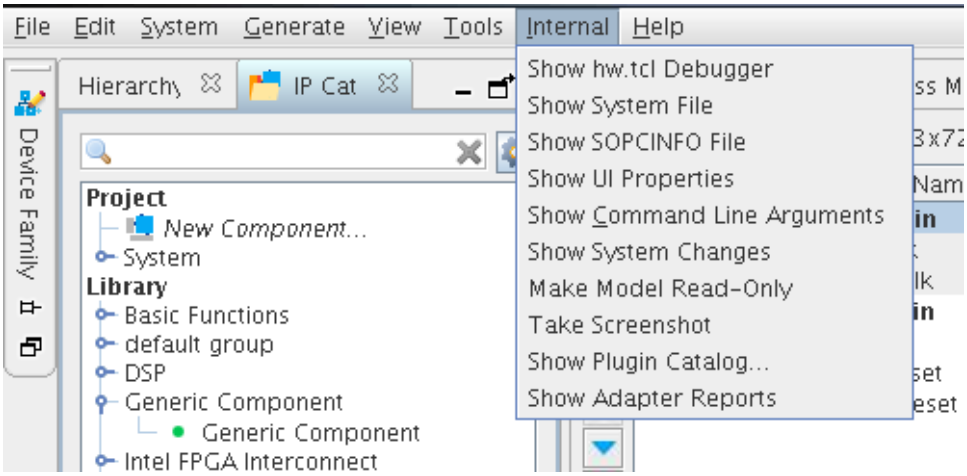


Table 164. Debug Options on the Internal Menu

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.

6.2. Scripting IP Core Generation

Use the `qsys-script` and `qsys-generate` utilities to define and generate an IP core variation outside of the Intel 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 333

6.2.1. qsys-generate Command-Line Options

Table 165. 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.
`--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.
`--export-qsys-script`	Optional	If you set this option to true, Platform Designer exports the post-generation system as a Platform Designer script file with the extension `.tcl`.
`--family=<value>`	Optional	Sets the device family name.
`--help`	Optional	Displays help for `--qsys-generate`.
`--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.
`--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.

continued...

Option	Usage	Description
`--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.
`--output-directory=<value>`	Optional	Sets the output directory. Platform Designer creates each generation target in a sub-directory of the output directory. If you do not specify the output directory, Platform Designer uses a sub-directory of the current working directory matching the name of the system.
`--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 Intel Quartus Prime project. Specify the generation language for the top-level RTL file for the Platform Designer system. The default value is *VERILOG*.
`--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*.
`--upgrade-ip-cores`	Optional	Enables upgrading all the IP cores that support upgrade in the Platform Designer system.
`--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.

6.3. Display Available IP Components with ip-catalog

The `ip-catalog` command displays a list of available IP components relative to the current Intel Quartus Prime project directory, as either text or XML.

You can use the following options with the `ip-catalog` utility:

Table 166. ip-catalog Command-Line Options

Option	Usage	Description
`--project-dir=<directory>`	Optional	Finds IP components relative to the Intel 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.

6.4. 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 167. 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.

6.5. 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 168. 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.
`--nativelink-mode`	Optional	Generates files for Intel Quartus Prime NativeLink RTL simulation.
`--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`.

6.6. 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 30. 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 169. 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 Intel 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` Intel Quartus Prime project file. Utilizes the specified Intel 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 Intel Quartus Prime project. Creates a new Intel 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 Intel Quartus Prime project revision as the new Intel Quartus Prime project name.

continued...

Option	Usage	Description
`--rev=<value>`	Optional	Allows you to specify the name of the Intel 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.
`--help`	Optional	Displays help for the `qsys-script` utility.

Related Information

Intel FPGA Wiki: Platform Designer Scripts

6.7. 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 Intel Quartus Prime software, include:

```
package require -exact qsys 18.0
```

The Platform Designer scripting commands fall under the following categories:

System on page 336
Subsystems on page 349
Instances on page 358
Connections on page 391
Top-level Exports on page 403
Validation on page 416
Miscellaneous on page 422
Wire-Level Connection Commands on page 435

6.7.1. System

This section lists the commands that allow you to manipulate a Platform Designer system.

create_system on page 337
export_hw_tcl on page 338
get_device_families on page 339
get_devices on page 340
get_module_properties on page 341
get_module_property on page 342
get_project_properties on page 343
get_project_property on page 344
load_system on page 345
save_system on page 346
set_module_property on page 49
set_project_property on page 348

6.7.1.1. 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 345
- save_system on page 346

6.7.1.2. 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 345
- save_system on page 346

6.7.1.3. 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 340

6.7.1.4. 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 339

6.7.1.5. 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 342
- set_module_property on page 49

6.7.1.6. 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 341
- set_module_property on page 49

6.7.1.7. get_project_properties

Description

Returns the list of properties that you can query for properties pertaining to the Intel 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 344
- set_project_property on page 348

6.7.1.8. get_project_property

Description

Returns the value of an Intel 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 341
- get_module_property on page 342
- set_module_property on page 49
- Project Properties on page 453

6.7.1.9. 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 337
- save_system on page 346

6.7.1.10. 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 345
- create_system on page 337

6.7.1.11. 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 341
- get_module_property on page 342
- Module Properties on page 447

6.7.1.12. 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 343
- get_project_property on page 344
- Project Properties on page 453

6.7.2. Subsystems

This section lists the commands that allow you to obtain the connection and parameter information of instances in your Platform Designer subsystem.

get_composed_connections on page 350
get_composed_connection_parameter_value on page 351
get_composed_connection_parameters on page 352
get_composed_instance_assignment on page 353
get_composed_instance_assignments on page 354
get_composed_instance_parameter_value on page 355
get_composed_instance_parameters on page 356
get_composed_instances on page 357

6.7.2.1. 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 351
- get_composed_connection_parameters on page 352

6.7.2.2. 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_master/memory.s0
baseAddress
```

Related Information

- get_composed_connection_parameters on page 352
- get_composed_connections on page 350

6.7.2.3. 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_master/memory.s0
```

Related Information

- get_composed_connection_parameter_value on page 351
- get_composed_connections on page 350

6.7.2.4. 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 354
- get_composed_instances on page 357

6.7.2.5. 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 353
- get_composed_instances on page 357

6.7.2.6. 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 356
- get_composed_instances on page 357

6.7.2.7. 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 355
- get_composed_instances on page 357

6.7.2.8. 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 353
- get_composed_instance_assignments on page 354
- get_composed_instance_parameter_value on page 355
- get_composed_instance_parameters on page 356

6.7.3. Instances

This section lists the commands that allow you to manipulate the instances of IP components in your Platform Designer system.

add_instance on page 359
apply_instance_preset on page 360
create_ip on page 361
add_component on page 362
duplicate_instance on page 363
enable_instance_parameter_update_callback on page 364
get_instance_assignment on page 365
get_instance_assignments on page 366
get_instance_documentation_links on page 367
get_instance_interface_assignment on page 368
get_instance_interface_assignments on page 369
get_instance_interface_parameter_property on page 370
get_instance_interface_parameter_value on page 371
get_instance_interface_parameters on page 372
get_instance_interface_port_property on page 373
get_instance_interface_ports on page 374
get_instance_interface_properties on page 375
get_instance_interface_property on page 376
get_instance_interfaces on page 377
get_instance_parameter_property on page 378
get_instance_parameter_value on page 43
get_instance_parameter_values on page 380
get_instance_parameters on page 44
get_instance_port_property on page 382
get_instance_properties on page 383
get_instance_property on page 384
get_instances on page 385
is_instance_parameter_update_callback_enabled on page 386
remove_instance on page 387
set_instance_parameter_value on page 388
set_instance_parameter_values on page 389
set_instance_property on page 390

6.7.3.1. 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 (optional) The required version of the specified instance type. If you do not specify any instance, Platform Designer uses the latest version.

Example

```
add_instance uart_0 altera_avalon_uart 16.1
```

Related Information

- get_instance_property on page 384
- get_instances on page 385
- remove_instance on page 387
- set_instance_parameter_value on page 388
- get_instance_parameter_value on page 43

6.7.3.2. 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 388

6.7.3.3. 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 362
- load_system on page 345
- save_system on page 346
- set_instance_parameter_value on page 388

6.7.3.4. 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

- load_component on page 0
- load_instantiation on page 0
- save_system on page 346

6.7.3.5. 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 359
- remove_instance on page 387

6.7.3.6. 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 386
- set_instance_parameter_value on page 388

6.7.3.7. 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 366

6.7.3.8. 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 365

6.7.3.9. 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}
```

6.7.3.10. 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 369

6.7.3.11. 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 368

6.7.3.12. 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_interface_parameters on page 372
- get_instance_interfaces on page 377
- get_parameter_properties on page 428
- Parameter Properties on page 448

6.7.3.13. 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 372
- get_instance_interfaces on page 377

6.7.3.14. 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 371
- get_instance_interfaces on page 377

6.7.3.15. 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 374
- get_port_properties on page 412
- Port Properties on page 452

6.7.3.16. 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 373
- get_instance_interfaces on page 377

6.7.3.17. 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 376
- get_instance_interfaces on page 377

6.7.3.18. 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 375
- get_instance_interfaces on page 377
- Element Properties on page 443

6.7.3.19. 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 374
- get_instance_interface_properties on page 375
- get_instance_interface_property on page 376

6.7.3.20. 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 44
- get_parameter_properties on page 428
- Parameter Properties on page 448

6.7.3.21. 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 44
- set_instance_parameter_value on page 388

6.7.3.22. 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 44
- set_instance_parameter_value on page 388
- set_instance_parameter_values on page 389

6.7.3.23. 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 378
- get_instance_parameter_value on page 43
- set_instance_parameter_value on page 388

6.7.3.24. 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 374
- get_port_properties on page 412
- Port Properties on page 452

6.7.3.25. 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 384

6.7.3.26. 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 383
- Element Properties on page 443

6.7.3.27. 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 359
- remove_instance on page 387

6.7.3.28. 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 364

6.7.3.29. 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 359
- get_instances on page 385

6.7.3.30. 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 43
- get_instance_parameter_property on page 378

6.7.3.31. 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 43
- get_instance_parameter_values on page 380
- get_instance_parameters on page 44

6.7.3.32. 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 44
- get_instance_property on page 384
- Instance Properties on page 444

6.7.4. Connections

This section lists the commands that allow you to manipulate the interface connections in your Platform Designer system.

add_connection on page 392
auto_connect on page 393
get_connection_parameter_property on page 394
get_connection_parameter_value on page 395
get_connection_parameters on page 396
get_connection_properties on page 397
get_connection_property on page 398
get_connections on page 399
remove_connection on page 400
remove_dangling_connections on page 401
set_connection_parameter_value on page 402

6.7.4.1. 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_master sdram.s1
```

Related Information

- get_connection_parameter_value on page 395
- get_connection_property on page 398
- get_connections on page 399
- remove_connection on page 400
- set_connection_parameter_value on page 402

6.7.4.2. auto_connect

Description

Creates connections from an instance or instance interface to matching interfaces of other instances in the system. For example, Avalon-MM slaves connect to Avalon-MM masters.

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 392

6.7.4.3. 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_master/dma0.csr baseAddress UNITS
```

Related Information

- get_connection_parameter_value on page 395
- get_connection_property on page 398
- get_connections on page 399
- get_parameter_properties on page 428
- Parameter Properties on page 448

6.7.4.4. 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-MM 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_master/dma0.csr baseAddress
```

Related Information

- get_connection_parameters on page 396
- get_connections on page 399
- set_connection_parameter_value on page 402

6.7.4.5. 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_master/dma0.csr
```

Related Information

- get_connection_parameter_property on page 394
- get_connection_parameter_value on page 395
- get_connection_property on page 398

6.7.4.6. 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 398
- get_connections on page 399

6.7.4.7. 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_master/dma0.csr TYPE
```

Related Information

- get_connection_properties on page 397
- Connection Properties on page 440

6.7.4.8. 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_master`, 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_master
```

Related Information

- add_connection on page 392
- remove_connection on page 400

6.7.4.9. 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_master/sdram.s0
```

Related Information

- add_connection on page 392
- get_connections on page 399

6.7.4.10. 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 392
- get_connections on page 399
- remove_connection on page 400

6.7.4.11. 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_master/dma0.csr baseAddress "0x000a0000"
```

Related Information

- get_connection_parameter_value on page 395
- get_connection_parameters on page 396

6.7.5. Top-level Exports

This section lists the commands that allow you to manipulate the exported interfaces in your Platform Designer system.

add_interface on page 404
get_exported_interface_sysinfo_parameter_value on page 405
get_exported_interface_sysinfo_parameters on page 406
get_interface_port_property on page 407
get_interface_ports on page 408
get_interface_properties on page 409
get_interface_property on page 410
get_interfaces on page 411
get_port_properties on page 412
remove_interface on page 413
set_interface_port_property on page 414
set_interface_property on page 415

6.7.5.1. 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 408
- get_interface_properties on page 409
- get_interface_property on page 410
- set_interface_property on page 415

6.7.5.2. 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 406
- set_exported_interface_sysinfo_parameter_value on page 0
- System Info Type Properties on page 454

6.7.5.3. 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 405
- set_exported_interface_sysinfo_parameter_value on page 0
- Access Type on page 460

6.7.5.4. 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 408
- get_port_properties on page 412
- Port Properties on page 452

6.7.5.5. 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 407
- get_interfaces on page 411

6.7.5.6. 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 410
- set_interface_property on page 415

6.7.5.7. 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 409
- set_interface_property on page 415
- Interface Properties on page 445

6.7.5.8. 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 404
- get_interface_ports on page 408
- get_interface_property on page 410
- remove_interface on page 413
- set_interface_property on page 415

6.7.5.9. 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 373
- get_instance_interface_ports on page 374
- get_instance_port_property on page 382
- get_interface_port_property on page 407
- get_interface_ports on page 408

6.7.5.10. 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 404
- get_interfaces on page 411

6.7.5.11. 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 465
- get_interface_ports on page 408
- get_interfaces on page 411
- get_port_properties on page 412

6.7.5.12. 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 404
- get_interface_properties on page 409
- get_interface_property on page 410
- Interface Properties on page 445

6.7.6. Validation

This section lists the commands that allow you to validate the components, instances, interfaces and connections in a Platform Designer system.

set_validation_property on page 417
validate_connection on page 418
validate_instance on page 419
validate_instance_interface on page 420
validate_system on page 421

6.7.6.1. 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 421
- Validation Properties on page 457

6.7.6.2. 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_master/sdram.s1
```

Related Information

- validate_instance on page 419
- validate_instance_interface on page 420
- validate_system on page 421

6.7.6.3. 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 418
- validate_instance_interface on page 420
- validate_system on page 421

6.7.6.4. 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_master
```

Related Information

- validate_connection on page 418
- validate_instance on page 419
- validate_system on page 421

6.7.6.5. 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 418
- validate_instance on page 419
- validate_instance_interface on page 420

6.7.7. Miscellaneous

This section lists the miscellaneous commands that you can use for your Platform Designer systems.

auto_assign_base_addresses on page 423
auto_assign_irqs on page 424
auto_assign_system_base_addresses on page 425
get_interconnect_requirement on page 426
get_interconnect_requirements on page 427
get_parameter_properties on page 428
lock_avalon_base_address on page 429
send_message on page 47
set_interconnect_requirement on page 431
set_use_testbench_naming_pattern on page 432
unlock_avalon_base_address on page 433
get_testbench_dutname on page 434
get_use_testbench_naming_pattern on page 435

6.7.7.1. 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 425
- lock_avalon_base_address on page 429
- unlock_avalon_base_address on page 433

6.7.7.2. 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
```

6.7.7.3. 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 423
- lock_avalon_base_address on page 429
- unlock_avalon_base_address on page 433

6.7.7.4. get_interconnect_requirement

Description

Returns the value of an interconnect requirement for a system or interface of a child instance.

Usage

`get_interconnect_requirement` <*element_id*> <*requirement*>

Returns

String The value of the interconnect requirement.

Arguments

element_id `{$system}` for the system, or the qualified name of the interface of an instance, in <*instance*>.<*interface*> format. In Tcl, the system identifier is escaped, for example, `{$system}`.

requirement The name of the requirement.

Example

```
get_interconnect_requirement {$system} qsys_mm.maxAdditionalLatency
```

6.7.7.5. get_interconnect_requirements

Description

Returns the list of all interconnect requirements in the system.

Usage

```
get_interconnect_requirements
```

Returns

String[] A flattened list of interconnect requirements. Every sequence of three elements in the list corresponds to one interconnect requirement. The first element in the sequence is the element identifier. The second element is the requirement name. The third element is the value. You can loop over the returned list with a `foreach loop`, for example:

```
foreach { element_id name value } $requirement_list { loop_body
          }
```

Arguments

No arguments.

Example

```
get_interconnect_requirements
```

6.7.7.6. 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 394
- get_instance_interface_parameter_property on page 370
- get_instance_parameter_property on page 378

6.7.7.7. 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 423
- auto_assign_system_base_addresses on page 425
- unlock_avalon_base_address on page 433

6.7.7.8. 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 Intel 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!"
```

6.7.7.9. set_interconnect_requirement

Description

Sets the value of an interconnect requirement for a system or an interface of a child instance.

Usage

`set_interconnect_requirement` <*element_id*> <*requirement*> <*value*>

Returns

No return value.

Arguments

element_id `{$system}` for the system, or qualified name of the interface of an instance, in <*instance*>.<*interface*> format. In Tcl, the system identifier is escaped, for example, `{$system}`.

requirement The name of the requirement.

value The requirement value.

Example

```
set_interconnect_requirement {$system} qsys_mm.clockCrossingAdapter HANDSHAKE
```

6.7.7.10. 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.

6.7.7.11. 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 423
- auto_assign_system_base_addresses on page 425
- lock_avalon_base_address on page 429

6.7.7.12. 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 435
- set_use_testbench_naming_pattern on page 432

6.7.7.13. 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 434
- set_use_testbench_naming_pattern on page 432

6.7.8. 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

Create a Composed Component or Subsystem on page 316

6.7.8.1. 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"
```

6.7.8.2. 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]}
```

6.7.8.3. 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]}
```

6.8. 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:

Connection Properties on page 440
Design Environment Type Properties on page 441
Direction Properties on page 442
Element Properties on page 443
Instance Properties on page 444
Interface Properties on page 445
Message Levels Properties on page 446
Module Properties on page 447
Parameter Properties on page 448
Parameter Status Properties on page 450
Parameter Type Properties on page 451
Port Properties on page 452
Project Properties on page 453
System Info Type Properties on page 454
Units Properties on page 456
Validation Properties on page 457
Interface Direction on page 458
File Set Kind on page 459
Access Type on page 460
Instantiation HDL File Properties on page 461
Instantiation Interface Duplicate Type on page 462
Instantiation Interface Properties on page 463
Instantiation Properties on page 464
Port Properties on page 465
VHDL Type on page 466

6.8.1. 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.

6.8.2. 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.

6.8.3. 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.

6.8.4. 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` or `avalon_slave`.
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`.

6.8.5. 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.

6.8.6. 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`

6.8.7. 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.

6.8.8. Module Properties

Type	Name	Description
String	GENERATION_ID	The generation ID for the system.
String	NAME	The name of the instance.

6.8.9. 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.
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.
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.

Type	Name	Description
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 454
- Parameter Type Properties on page 451
- Units Properties on page 456

6.8.10. 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.

6.8.11. 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.)

6.8.12. 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 442

6.8.13. Project Properties

Type	Name	Description
String	DEVICE	The device part number in the Intel Quartus Prime project that contains the Platform Designer system.
String	DEVICE_FAMILY	The device family name in the Intel Quartus Prime project that contains the Platform Designer system.

6.8.14. 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 slaves connected to the specified memory-mapped master 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 slave 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 slave connected to the specified memory-mapped master.
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 masters connected to a tri-state conduit slave. The slave is specified as the SYSTEM_INFO parameter property. The value contains information about the slave, connected master 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 slaves.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

Related Information

- Design Environment Type Properties on page 441
- Avalon Interface Specifications
- Platform Designer Interconnect on page 128

6.8.15. 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.

6.8.16. 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.

6.8.17. Interface Direction

Type	Name	Description
String	INPUT	Indicates that the interface is a slave (input, transmitter, sink, or end).
String	OUTPUT	Indicates that the interface is a master (output, receiver, source, or start).

6.8.18. 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 Intel 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.

6.8.19. 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.

6.8.20. 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.

6.8.21. 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.

6.8.22. Instantiation Interface Properties

Name	Type	Description
String	DIRECTION	The direction of the interface.
String	TYPE	The type of the interface.

6.8.23. 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.

6.8.24. 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 442

6.8.25. 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.

6.9. Platform Designer Command-Line Interface Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.12.15	18.1.0	First release as separate chapter.
2016.05.03	16.0.0	• Qsys Command-Line Utilities updated with latest supported command-line options.
June 2012	12.0.0	• Added command-line utilities, and scripts.
December 2010	10.1.0	Initial release of content.

7. 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 as Platform Designer (Standard).

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
- AMBA Protocol Specifications
- Creating Platform Designer Components on page 286
- Platform Designer Interconnect on page 128
- Publishing Component Information to Embedded Software
  In *Nios II Gen2 Software Developer's Handbook*

7.1. Platform Designer _hw.tcl Command Reference

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

7.1.1. Interfaces and Ports

add_interface on page 469
add_interface_port on page 471
get_interfaces on page 473
get_interface_assignment on page 474
get_interface_assignments on page 475
get_interface_ports on page 476
get_interface_properties on page 477
get_interface_property on page 478
get_port_properties on page 479
get_port_property on page 480
set_interface_assignment on page 481
set_interface_property on page 483
set_port_property on page 484
set_interface_upgrade_map on page 485

Related Information

Interface Properties on page 565

7.1.1.1. 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_slave avalon slave

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	master, slave
axi	master, slave
tristate_conduit	master, slave
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 471
- get_interface_assignments on page 475
- get_interface_properties on page 477
- get_interfaces on page 473

7.1.1.2. 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_slave 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 469
- get_port_properties on page 479

- get_port_property on page 480
- get_port_property on page 480
- Direction Properties on page 574
- Avalon Interface Specifications

7.1.1.3. 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 469

7.1.1.4. 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 469
- get_interface_assignments on page 475
- get_interfaces on page 473

7.1.1.5. 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 469
- get_interface_assignment on page 474
- get_interfaces on page 473

7.1.1.6. 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_slave
```

Related Information

- add_interface_port on page 471
- get_port_property on page 480
- set_port_property on page 484

7.1.1.7. 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 478
- set_interface_property on page 483
- Avalon Interface Specifications

7.1.1.8. 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_slave 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 477
- set_interface_property on page 483
- Avalon Interface Specifications

7.1.1.9. 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 471
- get_port_property on page 480
- set_port_property on page 484
- Port Properties on page 572

7.1.1.10. 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 471
- get_port_properties on page 479
- set_port_property on page 484
- Port Properties on page 572

7.1.1.11. 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 469
- get_interface_assignment on page 474
- get_interface_assignments on page 475

7.1.1.12. 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_slave 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 477
- get_interface_property on page 478
- Avalon Interface Specifications

7.1.1.13. 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 471
- get_port_properties on page 479
- set_port_property on page 484

7.1.1.14. 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_master_interface new_avalon_master_interface }
```

7.1.2. Parameters

add_parameter on page 487
get_parameters on page 488
get_parameter_properties on page 489
get_parameter_property on page 490
get_parameter_value on page 491
get_string on page 492
load_strings on page 493
set_parameter_property on page 494
set_parameter_value on page 495
decode_address_map on page 496

7.1.2.1. 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 489
- get_parameter_property on page 490
- get_parameter_value on page 491
- set_parameter_property on page 494
- set_parameter_value on page 495
- Parameter Type Properties on page 570

7.1.2.2. 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 487
- get_parameter_property on page 490
- get_parameter_value on page 491
- get_parameters on page 488
- set_parameter_property on page 494

7.1.2.3. 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 487
- get_parameter_property on page 490
- get_parameter_value on page 491
- get_parameters on page 488
- set_parameter_property on page 494
- Parameter Properties on page 568

7.1.2.4. 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 487
- get_parameter_properties on page 489
- get_parameter_value on page 491
- get_parameters on page 488
- set_parameter_property on page 494
- set_parameter_value on page 495
- Parameter Properties on page 568

7.1.2.5. 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 487
- get_parameter_property on page 490
- get_parameters on page 488
- set_parameter_property on page 494
- set_parameter_value on page 495

7.1.2.6. 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 493

7.1.2.7. 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 492
- Java Properties File

7.1.2.8. 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 487
- get_parameter_properties on page 489
- set_parameter_property on page 494
- Parameter Properties on page 568

7.1.2.9. 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 } ]
```

7.1.2.10. decode_address_map

Description

Converts an XML–formatted address map into a list of Tcl lists. Each inner list is in the correct format for conversion to an array. The XML code that describes each slave includes: its name, start address, and end address.

Availability

Elaboration, Generation, Composition

Usage

`decode_address_map` <*address_map_XML_string*>

Returns

No return value.

Arguments

address_mapXML_string	An XML string that describes the address map of a master.

Example

In this example, the code describes the address map for the master that accesses the `ext_ssram`, `sys_clk_timer` and `sysid` slaves. The format of the string may differ from the example below; it may have different white space between the elements and include additional attributes or elements. Use the `decode_address_map` command to decode the code that represents a master's address map to ensure that your code works with future versions of the address map.

```
<address-map>
  <slave name='ext_ssram' start='0x01000000' end='0x01200000' />
  <slave name='sys_clk_timer' start='0x02120800' end='0x02120820' />
  <slave name='sysid' start='0x021208B8' end='0x021208C0' />
</address-map>
```

Note: Intel recommends that you use the code provided below to enumerate over the IP components within an address map, rather than writing your own parser.

```
set address_map_xml [get_parameter_value my_map_param]
set address_map_dec [decode_address_map $address_map_xml]
foreach i $address_map_dec {
    array set info $i
    send_message info "Connected to slave $info(name)"
}
```

7.1.3. Display Items

add_display_item on page 498
get_display_items on page 500
get_display_item_properties on page 501
get_display_item_property on page 502
set_display_item_property on page 503

7.1.3.1. 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 501
- get_display_item_property on page 502
- get_display_items on page 500
- set_display_item_property on page 503
- Display Item Kind Properties on page 576

7.1.3.2. 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 498
- get_display_item_properties on page 501
- get_display_item_property on page 502
- set_display_item_property on page 503

7.1.3.3. 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 498
- get_display_item_property on page 502
- set_display_item_property on page 503
- Display Item Properties on page 575

7.1.3.4. 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 498
- get_display_item_properties on page 501
- get_display_items on page 500
- set_display_item_property on page 503
- Display Item Properties on page 575

7.1.3.5. 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 498
- get_display_item_properties on page 501
- get_display_item_property on page 502
- Display Item Properties on page 575

7.1.4. Module Definition

add_documentation_link on page 505
get_module_assignment on page 506
get_module_assignments on page 507
get_module_ports on page 508
get_module_properties on page 509
get_module_property on page 510
send_message on page 511
set_module_assignment on page 512
set_module_property on page 513
add_hdl_instance on page 514
package on page 515

7.1.4.1. 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
```

7.1.4.2. 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 507
- set_module_assignment on page 512

7.1.4.3. 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 506
- set_module_assignment on page 512

7.1.4.4. 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 469
- add_interface_port on page 471

7.1.4.5. 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 510
- set_module_property on page 513
- Module Properties on page 578

7.1.4.6. 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 509
- set_module_property on page 513
- Module Properties on page 578

7.1.4.7. 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!"
```

7.1.4.8. 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 506
- get_module_assignments on page 507

7.1.4.9. 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 509
- get_module_property on page 510
- Module Properties on page 578

7.1.4.10. 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_core_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_core_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 532
- get_instance_parameters on page 530
- get_instances on page 522
- set_instance_parameter_value on page 535

7.1.4.11. 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
```

7.1.5. Composition

add_instance on page 517
add_connection on page 518
get_connections on page 519
get_connection_parameters on page 520
get_connection_parameter_value on page 521
get_instances on page 522
get_instance_interfaces on page 523
get_instance_interface_ports on page 524
get_instance_interface_properties on page 525
get_instance_property on page 526
set_instance_property on page 527
get_instance_properties on page 528
get_instance_interface_property on page 529
get_instance_parameters on page 530
get_instance_parameter_property on page 531
get_instance_parameter_value on page 532
get_instance_port_property on page 533
set_connection_parameter_value on page 534
set_instance_parameter_value on page 535

7.1.5.1. 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 (optional) The required version of the specified type. If no version is specified, the highest available version is used.

Example

```
add_instance my_uart altera_avalon_uart
add_instance my_uart altera_avalon_uart 14.1
```

Related Information

- add_connection on page 518
- get_instance_interface_property on page 529
- get_instance_parameter_value on page 532
- get_instance_parameters on page 530
- get_instance_property on page 526
- get_instances on page 522
- set_instance_parameter_value on page 535

7.1.5.2. 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_master sdram.s1 avalon
```

Related Information

- add_instance on page 517
- get_instance_interfaces on page 523

7.1.5.3. 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 518

7.1.5.4. 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_master/dma0.csr
```

Related Information

- add_connection on page 518
- get_connection_parameter_value on page 521

7.1.5.5. 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_master/dma0.csr baseAddress
```

Related Information

- add_connection on page 518
- get_connection_parameters on page 520

7.1.5.6. 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 514
- add_instance on page 517
- get_instance_parameter_value on page 532
- get_instance_parameters on page 530
- set_instance_parameter_value on page 535

7.1.5.7. 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 517
- get_instance_interface_ports on page 524
- get_instance_interfaces on page 523

7.1.5.8. 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_master ]
```

Related Information

- add_instance on page 517
- get_instance_interfaces on page 523
- get_instance_port_property on page 533

7.1.5.9. 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_master ]
```

Related Information

- add_instance on page 517
- get_instance_interface_property on page 529
- get_instance_interfaces on page 523

7.1.5.10. 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 517
- get_instance_properties on page 528
- set_instance_property on page 527
- Instance Properties on page 567

7.1.5.11. 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 517
- get_instance_properties on page 528
- get_instance_property on page 526
- Instance Properties on page 567

7.1.5.12. 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 517
- get_instance_property on page 526
- set_instance_property on page 527
- Instance Properties on page 567

7.1.5.13. 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_master setupTime ]
```

Related Information

- add_instance on page 517
- get_instance_interfaces on page 523

7.1.5.14. 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 514
- add_instance on page 517
- get_instance_parameter_value on page 532
- get_instances on page 522
- set_instance_parameter_value on page 535

7.1.5.15. 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 517
- Parameter Properties on page 568

7.1.5.16. 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 514
- add_instance on page 517
- get_instance_parameters on page 530
- get_instances on page 522
- set_instance_parameter_value on page 535

7.1.5.17. 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 517
- get_instance_interface_ports on page 524
- Port Properties on page 572

7.1.5.18. 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-MM 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_master/dma0.csr baseAddress "0x000a0000"
```

Related Information

- add_connection on page 518
- get_connection_parameter_value on page 521

7.1.5.19. 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 514
- add_instance on page 517
- get_instance_parameter_value on page 532
- get_instances on page 522

7.1.6. Fileset Generation

add_fileset on page 537
add_fileset_file on page 538
set_fileset_property on page 539
get_fileset_file_attribute on page 540
set_fileset_file_attribute on page 541
get_fileset_properties on page 542
get_fileset_property on page 543
get_fileset_sim_properties on page 544
set_fileset_sim_properties on page 545
create_temp_file on page 546

7.1.6.1. 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 538
- get_fileset_property on page 543
- Fileset Properties on page 580

7.1.6.2. 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 Intel 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 537
- get_fileset_file_attribute on page 540
- File Kind Properties on page 584
- File Source Properties on page 585
- File Attribute Properties on page 583

7.1.6.3. 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 537
- Fileset Properties on page 580

7.1.6.4. 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 537
- add_fileset_file on page 538
- get_fileset_file_attribute on page 540
- File Attribute Properties on page 583

- add_fileset on page 537
- add_fileset_file on page 538
- get_fileset_file_attribute on page 540
- File Attribute Properties on page 583

7.1.6.5. 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
```

7.1.6.6. 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 537
- get_fileset_properties on page 542
- set_fileset_property on page 539
- Fileset Properties on page 580

7.1.6.7. 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 580

7.1.6.8. 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 537
- set_fileset_sim_properties on page 545
- Operating System Properties on page 592
- Simulator Properties on page 586

7.1.6.9. 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 544
- Operating System Properties on page 592
- Simulator Properties on page 586

7.1.6.10. 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 537
- add_fileset_file on page 538

7.1.7. Miscellaneous

check_device_family_equivalence on page 548
get_device_family_displayname on page 549
get_qip_strings on page 550
set_qip_strings on page 551
set_interconnect_requirement on page 552

7.1.7.1. 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 549

7.1.7.2. 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 548

7.1.7.3. 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 551

7.1.7.4. set_qip_strings

Description

Places strings in the Intel Quartus Prime IP File (**.qip**) file, which Platform Designer passes to the command as a Tcl list. You add the **.qip** file to your Intel 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 550

7.1.7.5. set_interconnect_requirement

Description

Sets the value of an interconnect requirement for a system or an interface on a child instance.

Availability

Composition

Usage

`set_interconnect_requirement` <*element_id*> <*name*> <*value*>

Returns

No return value

Arguments

element_id `{$system}` for system requirements, or qualified name of the interface of an instance, in `<instance>.<interface>` format. Note that the system identifier has to be escaped in TCL.

name The name of the requirement.

value The new requirement value.

Example

```
set_interconnect_requirement {$system} qsys_mm.maxAdditionalLatency 2
```

7.1.8. SystemVerilog Interface Commands

add_sv_interface on page 554
get_sv_interfaces on page 555
get_sv_interface_property on page 556
get_sv_interface_properties on page 557
set_sv_interface_property on page 558

7.1.8.1. 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
```

7.1.8.2. 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
```

7.1.8.3. 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
```

7.1.8.4. 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
```

7.1.8.5. 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
```

7.1.9. Wire-Level Expression Commands

set_wirelevel_expression on page 435
get_wirelevel_expressions on page 436
remove_wirelevel_expressions on page 437

7.1.9.1. 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"
```

7.1.9.2. 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]}
```

7.1.9.3. 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]}
```

7.2. Platform Designer _hw.tcl Property Reference

Script Language Properties on page 564
Interface Properties on page 565
SystemVerilog Interface Properties on page 565
Instance Properties on page 567
Parameter Properties on page 568
Parameter Type Properties on page 570
Parameter Status Properties on page 571
Port Properties on page 572
Direction Properties on page 574
Display Item Properties on page 575
Display Item Kind Properties on page 576
Display Hint Properties on page 577
Module Properties on page 578
Fileset Properties on page 580
Fileset Kind Properties on page 581
Callback Properties on page 582
File Attribute Properties on page 583
File Kind Properties on page 584
File Source Properties on page 585
Simulator Properties on page 586
Port VHDL Type Properties on page 587
System Info Type Properties on page 588
Design Environment Type Properties on page 590
Units Properties on page 591
Operating System Properties on page 592
Quartus.ini Type Properties on page 593

7.2.1. Script Language Properties

Name	Description
TCL	Implements the script in Tcl.

7.2.2. 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. Masters in the same SVD address group write register data of their connected slaves into the same SVD file
SVD_ADDRESS_OFFSET	Generates a CMSIS SVD file. Slaves connected to this master 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 468

7.2.3. 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.

7.2.4. 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

7.2.5. 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 588
- Parameter Type Properties on page 570
- Units Properties on page 591

7.2.6. 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.

7.2.7. 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.

7.2.8. 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> foo <role> <direction> <width> add_interface_port <interface> bar <role> <direction> <width> set_port_property foo 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`

Type	Name	Description
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`. Has no effect for IP components that implement a generation callback instead of using the default wrapper generation.
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 574
- Port VHDL Type Properties on page 587
- Avalon Interface Specifications

7.2.9. Direction Properties

Name	Description
`Bidir`	Direction for a bidirectional signal.
`Input`	Direction for an input signal.
`Output`	Direction for an output signal.

7.2.10. 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.

7.2.11. 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.

7.2.12. 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

7.2.13. Module Properties

Name	Description
ANALYZE_HDL	When set to false, prevents a call to the Intel 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 Intel 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.
GENERATION_CALLBACK	The name for a custom generation callback.
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 slaves, 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.

Name	Description
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
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.

7.2.14. 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.

7.2.15. Fileset Kind Properties

Name	Description
EXAMPLE_DESIGN	Contains example design files.
QUARTUS_SYNTH	Contains files that Platform Designer uses for the Intel 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`

7.2.16. 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.

7.2.17. 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.

7.2.18. File Kind Properties

Name	Description
DAT	DAT Data
FLI_LIBRARY	FLI Library
HEX	HEX Data
MIF	MIF Data
OTHER	Other
PLI_LIBRARY	PLI Library
QXP	QXP File
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

7.2.19. 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`.

7.2.20. 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

7.2.21. 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.

7.2.22. 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 slaves connected to the specified memory-mapped master 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 slave 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 slave connected to the specified memory-mapped master.
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 masters connected to an Avalon Tri-state Conduit slave. The slave is specified as the system info argument. The value contains information about the slave, the connected master 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 slaves.
String	UNIQUE_ID	A string guaranteed to be unique to this instance.

Related Information

Design Environment Type Properties on page 590

7.2.23. 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.

7.2.24. 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.

7.2.25. 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

7.2.26. 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.

7.3. Component Interface Tcl Reference Revision History

The table below indicates edits made to the *Component Interface Tcl Reference* content since its creation:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide.
2017.11.06	17.1.0	• Changed instances of *Qsys* to *Platform Designer (Standard)* • Added statement clarifying use of brackets.
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.
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.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Design Recommendations

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Online Version
Send Feedback
UG-20175
683323
2018.09.24

Contents

1. Recommended Design Practices 4
1.1. Following Synchronous FPGA Design Practices 4
1.1.1. Implementing Synchronous Designs 4
1.1.2. Asynchronous Design Hazards 5
1.2. HDL Design Guidelines 6
1.2.1. Optimizing Combinational Logic 6
1.2.2. Optimizing Clocking Schemes 9
1.2.3. Optimizing Physical Implementation and Timing Closure 14
1.2.4. Optimizing Power Consumption 17
1.2.5. Managing Design Metastability 17
1.3. Checking Design Violations 17
1.3.1. Validating Against Design Rules 17
1.3.2. Creating Custom Design Rules 20
1.4. Use Clock and Register-Control Architectural Features 24
1.4.1. Use Global Reset Resources 24
1.4.2. Use Global Clock Network Resources 33
1.4.3. Use Clock Region Assignments to Optimize Clock Constraints 34
1.4.4. Avoid Asynchronous Register Control Signals 35
1.5. Implementing Embedded RAM 35
1.6. Recommended Design Practices Revision History 36

2. Recommended HDL Coding Styles 38
2.1. Using Provided HDL Templates 38
2.1.1. Inserting HDL Code from a Provided Template 38
2.2. Instantiating IP Cores in HDL 39
2.3. Inferring Multipliers and DSP Functions 40
2.3.1. Inferring Multipliers 40
2.3.2. Inferring Multiply-Accumulator and Multiply-Adder Functions 41
2.4. Inferring Memory Functions from HDL Code 44
2.4.1. Inferring RAM functions from HDL Code 45
2.4.2. Inferring ROM Functions from HDL Code 62
2.4.3. Inferring Shift Registers in HDL Code 64
2.5. Register and Latch Coding Guidelines 67
2.5.1. Register Power-Up Values 67
2.5.2. Secondary Register Control Signals Such as Clear and Clock Enable 69
2.5.3. Latches 71
2.6. General Coding Guidelines 75
2.6.1. Tri-State Signals 75
2.6.2. Clock Multiplexing 75
2.6.3. Adder Trees 77
2.6.4. State Machine HDL Guidelines 79
2.6.5. Multiplexer HDL Guidelines 85
2.6.6. Cyclic Redundancy Check Functions 88
2.6.7. Comparator HDL Guidelines 90
2.6.8. Counter HDL Guidelines 91
2.7. Designing with Low-Level Primitives 91
2.8. Recommended HDL Coding Styles Revision History 92

3. Managing Metastability with the Intel Quartus Prime Software..... 94
3.1. Metastability Analysis in the Intel Quartus Prime Software.....95
3.1.1. Synchronization Register Chains..... 95
3.1.2. Identify Synchronizers for Metastability Analysis..... 96
3.1.3. How Timing Constraints Affect Synchronizer Identification and Metastability Analysis..... 96
3.2. Metastability and MTBF Reporting.....97
3.2.1. Metastability Reports.....98
3.2.2. Synchronizer Data Toggle Rate in MTBF Calculation.....100
3.3. MTBF Optimization..... 100
3.3.1. Synchronization Register Chain Length.....101
3.4. Reducing Metastability Effects.....102
3.4.1. Apply Complete System-Centric Timing Constraints for the Timing Analyzer... 102
3.4.2. Force the Identification of Synchronization Registers.....102
3.4.3. Set the Synchronizer Data Toggle Rate.....103
3.4.4. Optimize Metastability During Fitting.....103
3.4.5. Increase the Length of Synchronizers to Protect and Optimize.....103
3.4.6. Set Fitter Effort to Standard Fit instead of Auto Fit.....103
3.4.7. Increase the Number of Stages Used in Synchronizers..... 104
3.4.8. Select a Faster Speed Grade Device.....104
3.5. Scripting Support.....104
3.5.1. Identifying Synchronizers for Metastability Analysis..... 105
3.5.2. Synchronizer Data Toggle Rate in MTBF Calculation.....105
3.5.3. report_metastability and Tcl Command.....105
3.5.4. MTBF Optimization..... 106
3.5.5. Synchronization Register Chain Length.....106
3.6. Managing Metastability..... 106
3.7. Managing Metastability with the Intel Quartus Prime Software Revision History.....106

A. Intel Quartus Prime Standard Edition User Guides.....108

1. Recommended Design Practices

This chapter provides design recommendations for Intel® FPGA devices. This chapter also describes the Intel Quartus® Prime Design Assistant. The Design Assistant checks your design for violations of Intel's design recommendations

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

1.1. 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.

1.1.1. 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

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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.

1.1.2. 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.

1.2. 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.

1.2.1. 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).

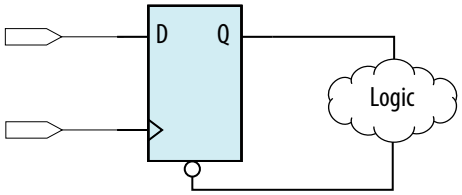
For cases where combinational logic feeds registers, the register control signals can implement part of the logic function to save LUT resources. By following the recommendations in this section, you can improve the reliability of your combinational design.

1.2.1.1. 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 1. 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.

1.2.1.2. 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 71

1.2.1.3. 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.

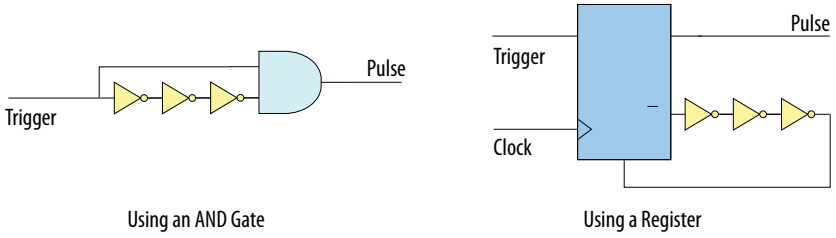
In some ASIC designs, delays are used for buffering signals as they are routed around the device. This functionality is not required in FPGA devices because the routing structure provides buffers throughout the device.

1.2.1.4. Use Synchronous Pulse Generators

Use synchronous techniques to design pulse generators.

Figure 2. 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).

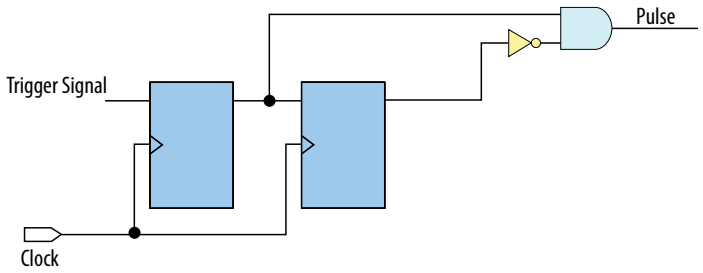


In the first method, a trigger signal feeds both inputs of a 2-input AND gate, and the design adds inverters to one of the inputs to create a delay chain. The width of the pulse depends on the time differences between the path that feeds the gate directly and the path that goes through the delay chain. 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 3. 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.

1.2.2. Optimizing Clocking Schemes

Like combinational logic, clocking schemes have a large effect on the performance and reliability of a design.

Avoid using internally generated clocks (other than PLLs) wherever possible because they can cause functional and timing problems in the design. Clocks generated with combinational logic can introduce glitches that create functional problems, and the delay inherent in combinational logic can lead to timing problems.

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.

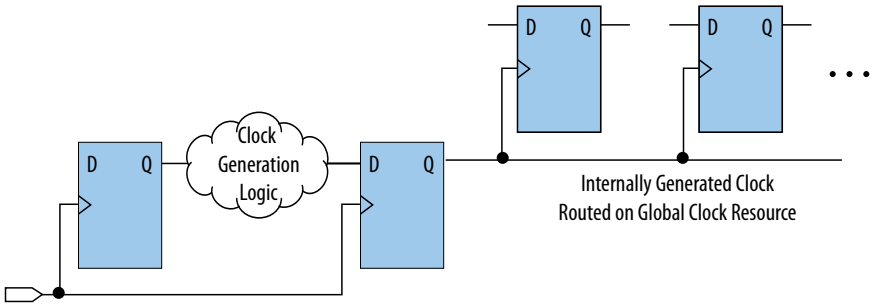
1.2.2.1. Register Combinational Logic Outputs

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 4. 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.

1.2.2.2. 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.

1.2.2.3. 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.

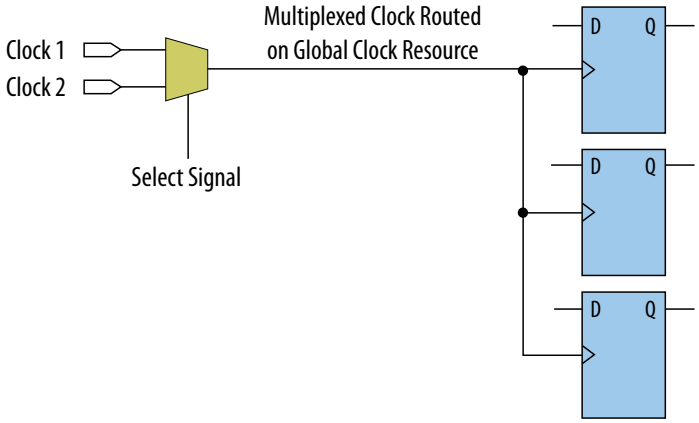
You can often use ripple clock structures to make ripple counters out of the smallest amount of logic possible. However, in all Intel devices supported by the Intel Quartus Prime software, using a ripple clock structure to reduce the amount of logic used for a counter is unnecessary because the device allows you to construct a counter using one logic element per counter bit. You should avoid using ripple counters completely.

1.2.2.4. 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 5. 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.

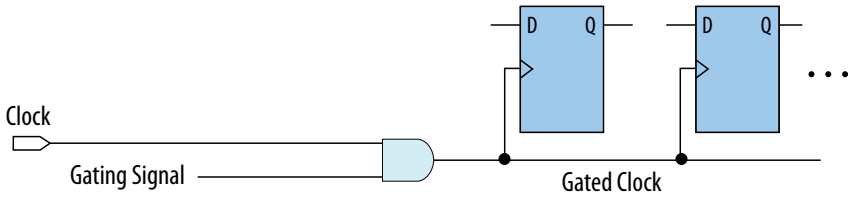
Tip: Use dedicated hardware to perform clock multiplexing when it is available, instead of using multiplexing logic. For example, you can use the clock-switchover feature or clock control block available in certain Intel FPGA devices. These dedicated hardware blocks ensure that you use global low-skew 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 on the Literature page of the Altera website.

1.2.2.5. 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 6. 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.

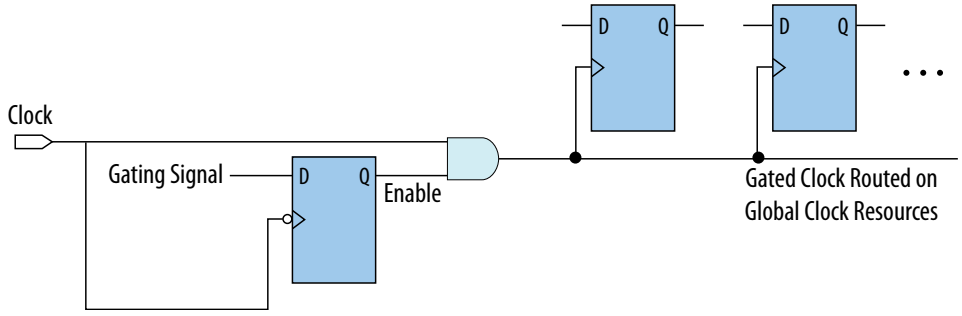
From a functional point of view, you can shut down a clock domain in a purely synchronous manner using a synchronous clock enable signal. However, when using a synchronous clock enable scheme, the clock network continues toggling. This practice does not reduce power consumption as much as gating the clock at the source does. In most cases, use a synchronous scheme.

1.2.2.5.1. 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 7. 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

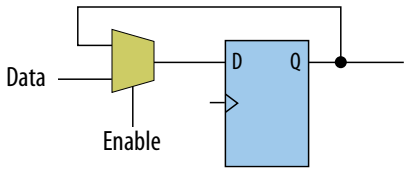
Auto Gated Clock Conversion logic option
In *Intel Quartus Prime Help*

1.2.2.6. 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 8. Synchronous Clock Enable



1.2.3. 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.

1.2.3.1. 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.

1.2.3.2. 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.

1.2.3.3. 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 1. Compilation Settings that Impact Timing Closure

Setting	Location	Effect on Timing Closure
Perform Physical Synthesis for Combinational logic for Performance	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**	If enabled, the Netlist Optimization report panel identifies logic that physical synthesis can modify. You can use this information to modify the design so that the associated optimization can be turned off to save compile time.
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**	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

Design Evaluation for Timing Closure
In *Intel Quartus Prime Standard Edition Handbook Volume 2*

1.2.3.4. 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

Exploring Paths in the Chip Planner
In *Intel Quartus Prime Standard Edition Handbook Volume 2*

1.2.4. 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

Power Optimization
In *Intel Quartus Prime Standard Edition Handbook Volume 2*

1.2.5. 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 94
- Metastability Analysis and Optimization Techniques
  In *Intel Quartus Prime Standard Edition Handbook Volume 2*

1.3. Checking Design Violations

To improve the reliability, timing performance, and logic utilization of your design, avoid design rule violations. The Intel Quartus Prime software provides the Design Assistant tool that automatically checks for design rule violations and reports their location. The Design Assistant is supported only in the Quartus Prime Standard Edition software. The Design Assistant does not support Intel Arria® 10 or MAX 10 devices.

The Design Assistant is a design rule checking tool that allows you to check for design issues early in the design flow. The Design Assistant checks your design for adherence to Intel-recommended design guidelines. You can specify which rules you want the Design Assistant to apply to your design. This is useful if you know that your design violates particular rules that are not critical and you can allow these rule violations. The Design Assistant generates design violation reports with details about each violation based on the settings that you specified.

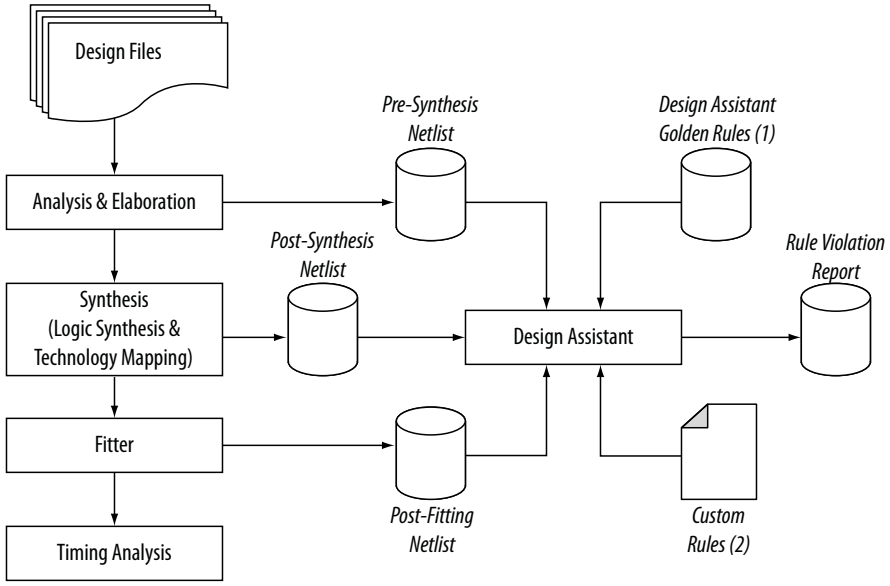
This section provides an introduction to the Intel Quartus Prime design flow with the Design Assistant, message severity levels, and an explanation about how to set up the Design Assistant. The last parts of the section describe the design rules and the reports generated by the Design Assistant.

1.3.1. Validating Against Design Rules

You can run the Design Assistant following design synthesis or compilation. The Design Assistant performs a post-fit netlist analysis of your design.

The default is to apply all the rules to your project. If there are some rules that are unimportant to your design, you can turn off the rules that you do not want the Design Assistant to use.

Figure 9. Intel Quartus Prime Design Flow with the Design Assistant



1. Database of the default rules for the Design Assistant.
2. A file that contains the **.xml** codes of the custom rules for the Design Assistant. For more details about how to create this file .

   The Design Assistant analyzes your design netlist at different stages of the compilation flow and may yield different warnings or errors, even though the netlists are functionally the same. Your pre-synthesis, post-synthesis, and post-fitting netlists might be different due to optimizations performed by the Intel Quartus Prime software. For example, a warning message in a pre-synthesis netlist may be removed after the netlist has been synthesized into a post-synthesis or post-fitting netlist.

   The exact operation of the Design Assistant depends on when you run it:

- When you run the Design Assistant after running a full compilation or fitting, the Design Assistant performs a post-fitting analysis on the design.
- When you run the Design Assistant after performing Analysis and Synthesis, the Design Assistant performs post-synthesis analysis on the design.
- When you start the Design Assistant after performing Analysis and Elaboration, the Design Assistant performs a pre-synthesis analysis on the design. You can also perform pre-synthesis analysis with the Design Assistant using the command-line. You can use the `-rtl` option with the `quartus_drc` executable, as shown in the following example:

  ```
  quartus_drc <project_name> --rtl=on
  ```

  If your design violates a design rule, the Design Assistant generates warning messages and information messages about the violated rule. The Design Assistant displays these messages in the Messages window, in the Design Assistant Messages report, and in the Design Assistant report files. You can find the Design Assistant report files called <project_name>**.drc.rpt** in the <project_name> subdirectory of the project directory.

1.3.2. Creating Custom Design Rules

You can define and validate your design against your own custom set of design rules. You can save these rules in a text file (with any file extension) with the XML format.

You then specify the path to that file in the Design Assistant settings page and run the Design Assistant for violation checking.

Refer to the following location to locate the file that contains the default rules for the Design Assistant:

<Intel Quartus Prime install path>**\quartus\libraries\design-assistant\da_golden_rule.xml**

1.3.2.1. Custom Design Rule Examples

The following examples of custom rules show how to check node relationships and clock relationships in a design.

This example shows the XML codes for checking SR latch structures in a design.

Example 1. Detecting SR Latches in a Design

```
<DA_RULE ID="EX01" SEVERITY="CRITICAL" NAME="Checking Design for SR Latch"
DEFAULT_RUN="YES">
<RULE_DEFINITION>
   <FORBID>
   <OR>
      <NODE NAME="NODE_1" TYPE="SRLATCH" />
      <HAS_NODE NODE_LIST="NODE_1" />
      <NODE NAME="NODE_1" TOTAL_FANIN="EQ2" />
      <NODE NAME="NODE_2" TOTAL_FANIN="EQ2" />
      <AND>
         <NODE_RELATIONSHIP FROM_NAME="NODE_1" FROM_TYPE="NAND" TO_NAME="NODE_2"
TO_TYPE="NAND" />
         <NODE_RELATIONSHIP FROM_NAME="NODE_2" FROM_TYPE="NAND" TO_NAME="NODE_1"
TO_TYPE="NAND" />
      </AND>
      <AND>
         <NODE_RELATIONSHIP FROM_NAME="NODE_1" FROM_TYPE="NOR" TO_NAME="NODE_2"
TO_TYPE="NOR" />
         <NODE_RELATIONSHIP FROM_NAME="NODE_2" FROM_TYPE="NOR" TO_NAME="NODE_1"
TO_TYPE="NOR" />
      </AND>
      </OR>
   </FORBID>
</RULE_DEFINITION>

<REPORTING_ROOT>
   <MESSAGE NAME="Rule %ARG1%: Found %ARG2% node(s) related to this rule.">
      <MESSAGE_ARGUMENT NAME="ARG1" TYPE="ATTRIBUTE" VALUE="ID" />
      <MESSAGE_ARGUMENT NAME="ARG2" TYPE="TOTAL_NODE" VALUE="NODE_1" />
   </MESSAGE>
</REPORTING_ROOT>
</DA_RULE>
```

The possible SR latch structures are specified in the rule definition section. Codes defined in the <AND></AND> block are tied together, meaning that each statement in the block must be true for the block to be fulfilled (AND gate similarity). In the

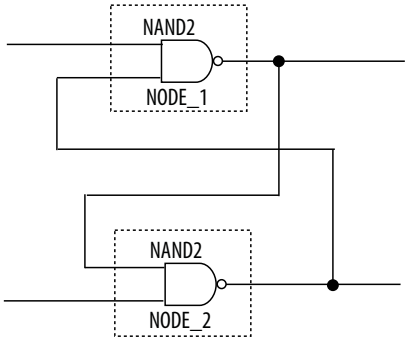
<OR></OR> block, as long as one statement in the block is true, the block is fulfilled (OR gate similarity). If no <AND></AND> or <OR></OR> blocks are specified, the default is <AND></AND>.

The <FORBID></FORBID> section contains the undesirable condition for the design, which in this case is the SR latch structures. If the condition is fulfilled, the Design Assistant highlights a rule violation.

Example 2. Detecting SR Latches in a Design

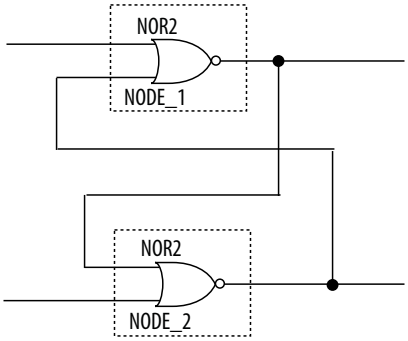
```
<AND>
  <NODE_RELATIONSHIP FROM_NAME="NODE_1" FROM_TYPE="NAND" TO_NAME="NODE_2"
   TO_TYPE="NAND" />
  <NODE_RELATIONSHIP FROM_NAME="NODE_2" FROM_TYPE="NAND" TO_NAME="NODE_1"
   TO_TYPE="NAND" />
</AND>
```

Figure 10. Undesired Condition 1




```
<AND>
 <NODE_RELATIONSHIP FROM_NAME="NODE_1" FROM_TYPE="NOR" TO_NAME="NODE_2"
TO_TYPE="NOR" />
 <NODE_RELATIONSHIP FROM_NAME="NODE_2" FROM_TYPE="NOR" TO_NAME="NODE_1"
TO_TYPE="NOR" />
</AND>
```

Figure 11. Undesired Condition 2



This example shows how to use the CLOCK_RELATIONSHIP attribute to relate nodes to clock domains. This example checks for correct synchronization in data transfer between asynchronous clock domains. Synchronization is done with cascaded registers, also called synchronizers, at the receiving clock domain. The code in This example checks for the synchronizer configuration based on the following guidelines:

- The cascading registers need to be triggered on the same clock edge
- There is no logic between the register output of the transmitting clock domain and the cascaded registers in the receiving asynchronous clock domain.

Example 3. Detecting Incorrect Synchronizer Configuration

```
<DA_RULE ID="EX02" SEVERITY="HIGH" NAME="Data Transfer Not Synch Correctly"
DEFAULT_RUN="YES">

<RULE_DEFINITION>
<DECLARE>
   <NODE NAME="NODE_1" TYPE="REG" />
   <NODE NAME="NODE_2" TYPE="REG" />
   <NODE NAME="NODE_3" TYPE="REG" />
</DECLARE>
<FORBID>
   <NODE_RELATIONSHIP FROM_NAME="NODE_1" TO_NAME="NODE_2" TO_PORT="D_PORT"
CLOCK_RELATIONSHIP="ASYN" />
   <NODE_RELATIONSHIP FROM_NAME="NODE_2" TO_NAME="NODE_3" TO_PORT="D_PORT"
CLOCK_RELATIONSHIP="!ASYN" />
   <OR>
      <NODE_RELATIONSHIP FROM_NAME="NODE_1" TO_NAME="NODE_2" TO_PORT="D_PORT"
REQUIRED_THROUGH="YES" THROUGH_TYPE="COMB" CLOCK_RELATIONSHIP="ASYN" />
      <CLOCK_RELATIONSHIP NAME="SEQ_EDGE|ASYN" NODE_LIST="NODE_2, NODE_3" />
   </OR>
</FORBID>
</RULE_DEFINITION>

<REPORTING_ROOT>
<MESSAGE NAME="Rule %ARG1%: Found %ARG2% node(s) related to this rule.">
   <MESSAGE_ARGUMENT NAME="ARG1" TYPE="ATTRIBUTE" VALUE="ID" />
   <MESSAGE_ARGUMENT NAME="ARG2" TYPE="TOTAL_NODE" VALUE="NODE_1" />
   <MESSAGE NAME="Source node(s): %ARG3%, Destination node(s): %ARG4%">
      <MESSAGE_ARGUMENT NAME="ARG3" TYPE="NODE" VALUE="NODE_1" />
      <MESSAGE_ARGUMENT NAME="ARG4" TYPE="NODE" VALUE="NODE_2" />
   </MESSAGE>
</MESSAGE>
</REPORTING_ROOT>
</DA_RULE>
```

The codes differentiate the clock domains. ASYN means asynchronous, and !ASYN means non-asynchronous. This notation is useful for describing nodes that are in different clock domains. The following lines from the example state that NODE_2 and NODE_3 are in the same clock domain, but NODE_1 is not.

```
<NODE_RELATIONSHIP FROM_NAME="NODE_1" TO_NAME="NODE_2" TO_PORT="D_PORT"
CLOCK_RELATIONSHIP="ASYN" />

<NODE_RELATIONSHIP FROM_NAME="NODE_2" TO_NAME="NODE_3" TO_PORT="D_PORT"
CLOCK_RELATIONSHIP="!ASYN" />
```

The next line of code states that NODE_2 and NODE_3 have a clock relationship of either sequential edge or asynchronous.

```
<CLOCK_RELATIONSHIP NAME="SEQ_EDGE|ASYN" NODE_LIST="NODE_2, NODE_3" />
```

The <FORBID></FORBID> section contains the undesirable condition for the design, which in this case is the undesired configuration of the synchronizer. If the condition is fulfilled, the Design Assistant highlights a rule violation.

The possible SR latch structures are specified in the rule definition section. Codes defined in the <AND></AND> block are tied together, meaning that each statement in the block must be true for the block to be fulfilled (AND gate similarity). In the <OR></OR> block, as long as one statement in the block is true, the block is fulfilled (OR gate similarity). If no <AND></AND> or <OR></OR> blocks are specified, the default is <AND></AND>.

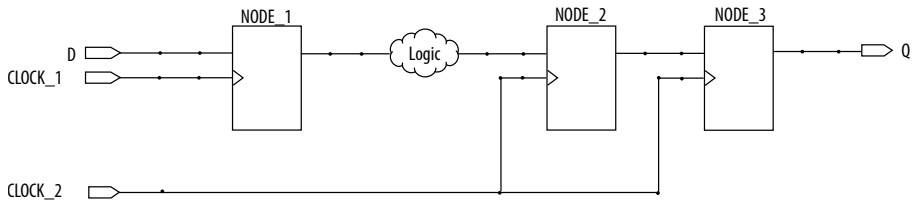
The <FORBID></FORBID> section contains the undesirable condition for the design, which in this case is the SR latch structures. If the condition is fulfilled, the Design Assistant highlights a rule violation.

The following examples show the undesired conditions from with their equivalent block diagrams:

Example 4. Undesired Condition 3

```
<NODE_RELATIONSHIP FROM_NAME="NODE_1" TO_NAME="NODE_2" TO_PORT="D_PORT"
CLOCK_RELATIONSHIP="ASYN" />

<NODE_RELATIONSHIP FROM_NAME="NODE_2" TO_NAME="NODE_3" TO_PORT="D_PORT"
CLOCK_RELATIONSHIP="!ASYN" />

<NODE_RELATIONSHIP FROM_NAME="NODE_1" TO_NAME="NODE_2" TO_PORT="D_PORT"
REQUIRED_THROUGH="YES"
   THROUGH_TYPE="COMB" CLOCK_RELATIONSHIP="ASYN" />
```

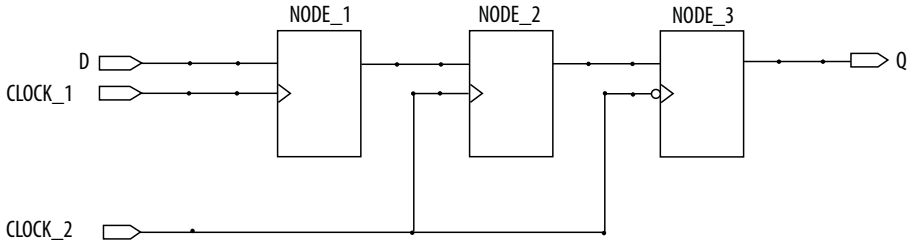
Figure 12. Undesired Condition 3



Example 5. Undesired Condition 4

```
<NODE_RELATIONSHIP FROM_NAME="NODE_1" TO_NAME="NODE_2" TO_PORT="D_PORT"
CLOCK_RELATIONSHIP="ASYN" />

<NODE_RELATIONSHIP FROM_NAME="NODE_2" TO_NAME="NODE_3" TO_PORT="D_PORT"
CLOCK_RELATIONSHIP="!ASYN" />

<CLOCK_RELATIONSHIP NAME="SEQ_EDGE|ASYN" NODE_LIST="NODE_2, NODE_3" />
```

Figure 13. Undesired Condition 4



1.4. 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.

1.4.1. 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

1.4.1.1. 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 (`sync lr`). 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 14. Synchronous Reset

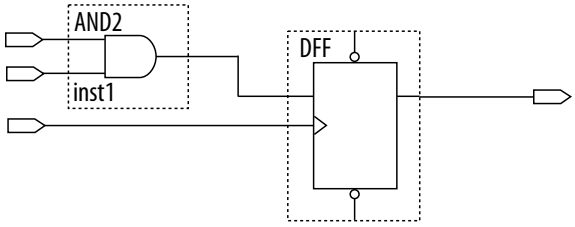
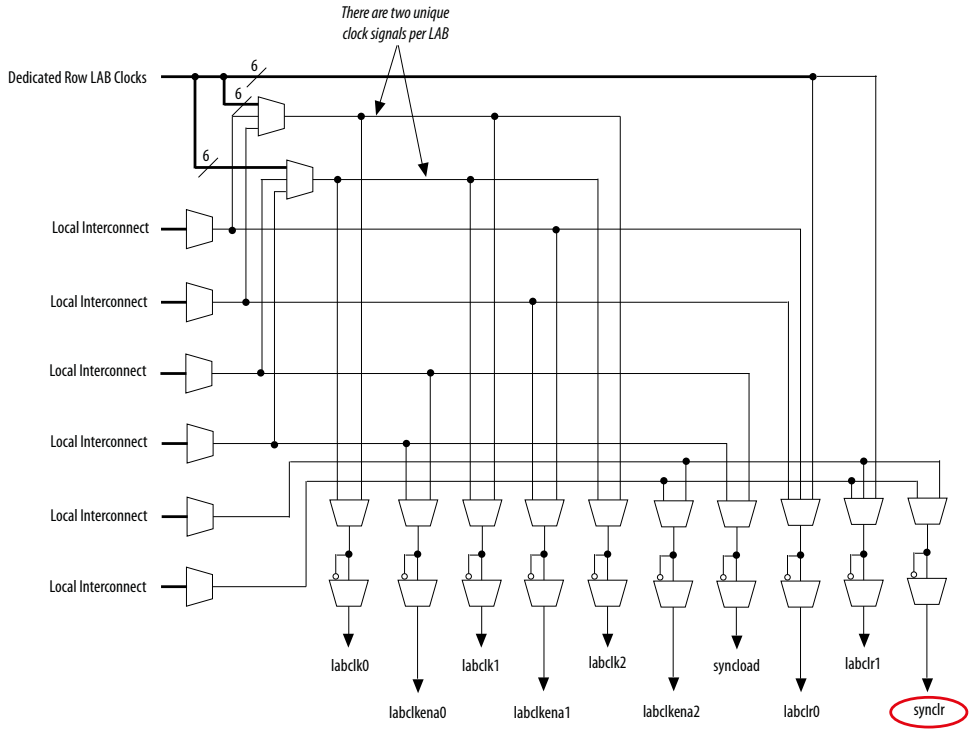
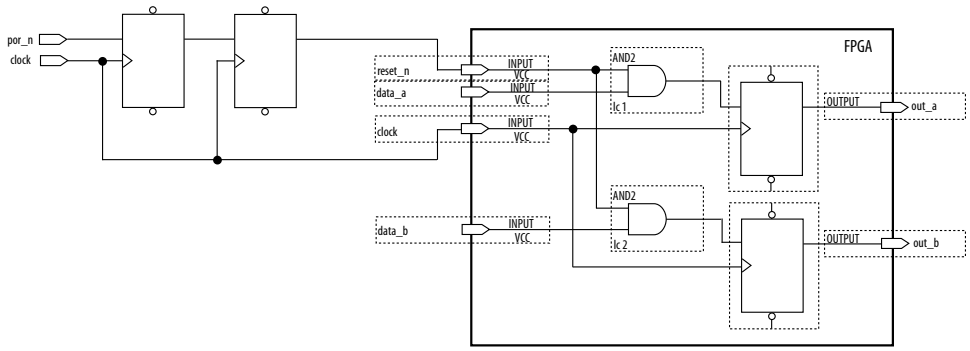


Figure 15. LAB-Wide Control Signals



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 16. 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 6. 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`.

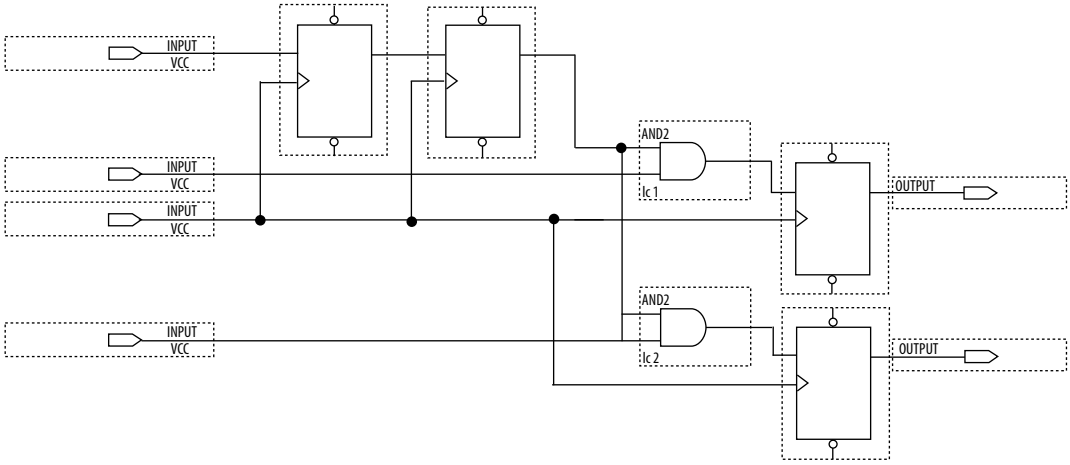
Example 7. SDC Constraints for Externally Synchronized Reset

```
# 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 17. 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 8. 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
```

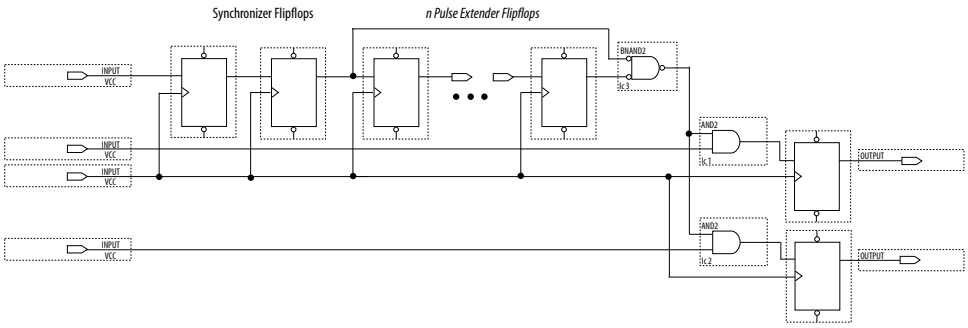
The SDC constraints are similar to the external synchronous reset, except that the input reset cannot be constrained because it is asynchronous. Cut the input path with a `set_false_path` statement to avoid these being considered as unconstrained paths.

Example 9. 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 18. 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.

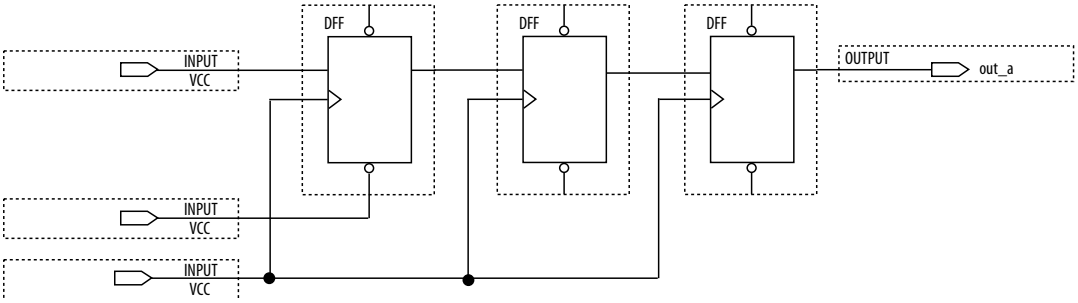
1.4.1.2. Using Asynchronous Resets

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 19. 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 10. 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 11. 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.

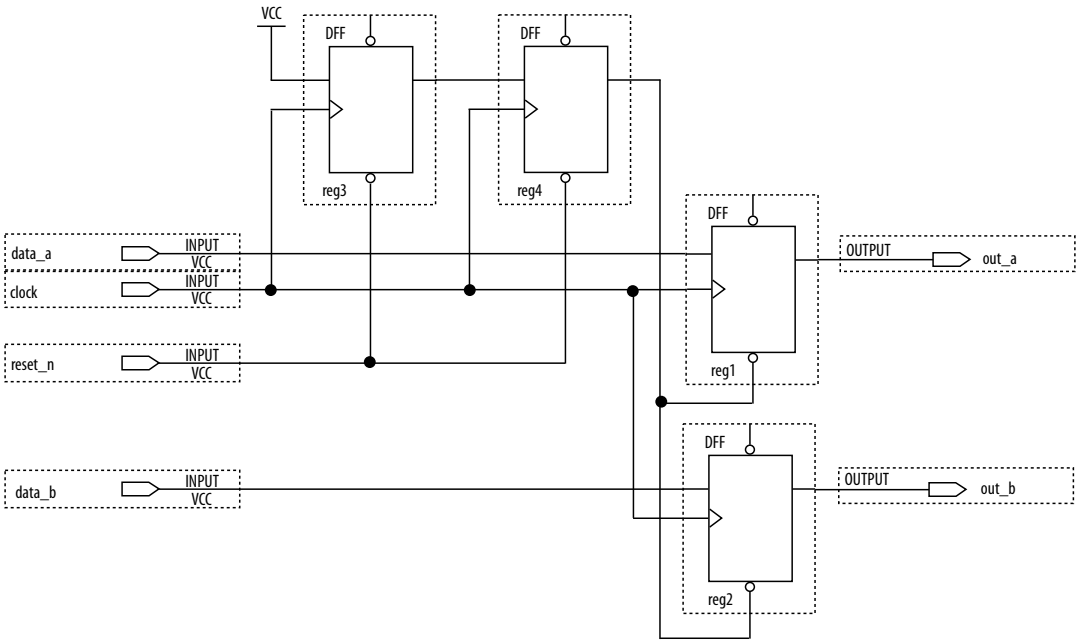
1.4.1.3. Use Synchronized Asynchronous Reset

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 20. 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 12. 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.

1.4.2. 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.

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.

1.4.3. 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.

1.4.3.1. 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.

1.4.4. 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.

1.5. 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. For Intel Quartus Prime integrated synthesis, add the synthesis attribute `ramstyle="no_rw_check"` to allow the software to choose the read-during-write behavior of a RAM, rather than using the read-during-write behavior

specified in your HDL code. Using this type of attribute prevents the synthesis tool from using extra logic to implement the memory block and, in some cases, can allow memory inference when it would otherwise be impossible.

Related Information

Inferring RAM functions from HDL Code on page 45

1.6. Recommended Design Practices Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • Created subtopic: *Clock Region Assignments in Intel Arria 10 and Older Device Families* from content in topic: *Use Clock Region Assignments to Optimize Clock Constraints*.
2017.11.06	17.1.0	• Updated topic: *Optimizing Timing Closure*.
2016.05.03	16.0.0	• Replaced Internally Synchronized Reset code sample with corrected version. • Stated limitations about deprecated physical synthesis options. • Clarified limitations of support for 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.
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.

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 • 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

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

2. 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 (including other Quartus software products and other EDA tools), refer to the synthesis tool vendor's documentation.

Related Information

- Recommended Design Practices on page 4
- Advanced Synthesis Cookbook
- Design Examples
- Reference Designs

2.1. 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.

2.1.1. 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.

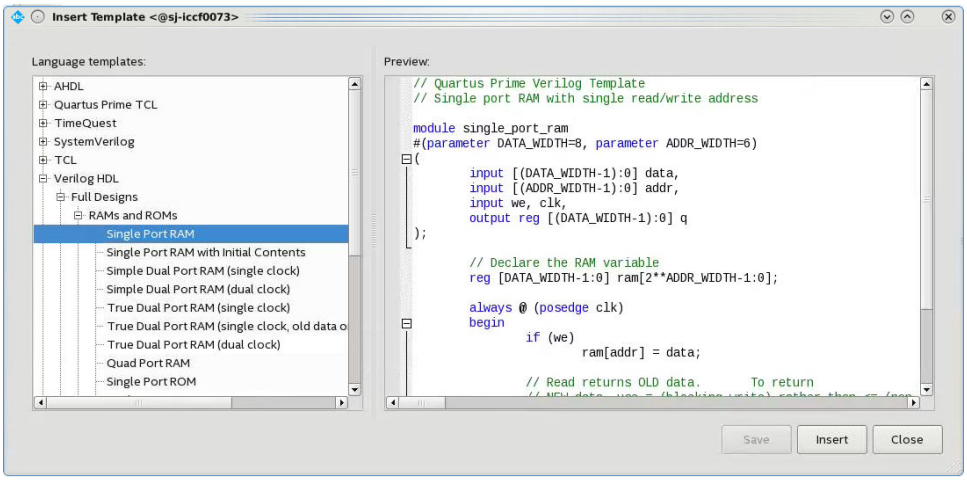
Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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 21. 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.

2.2. 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.

Related Information

Intel FPGA IP Core Literature

2.3. 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

DSP Solutions Center

2.3.1. 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 13. 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 14. 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 15. 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 16. 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;
```

2.3.2. 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 17. 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
```

Example 18. Verilog HDL Signed Multiply-Adder

```
module sig_altmult_add (dataa, datab, datac, datad, clock, aclr, result);
   input signed [15:0] dataa, datab, datac, datad;
   input clock, aclr;
   output reg signed [32:0] result;

   reg signed [15:0] dataa_reg, datab_reg, datac_reg, datad_reg;
   reg signed [31:0] mult0_result, mult1_result;

   always @ (posedge clock or posedge aclr) begin
```

```
       if (aclr) begin
           dataa_reg <= 16'b0;
           datab_reg <= 16'b0;
           datac_reg <= 16'b0;
           datad_reg <= 16'b0;
           mult0_result <= 32'b0;
           mult1_result <= 32'b0;
           result <= 33'b0;
        end
        else begin
           dataa_reg <= dataa;
           datab_reg <= datab;
           datac_reg <= datac;
           datad_reg <= datad;
           mult0_result <= dataa_reg * datab_reg;
           mult1_result <= datac_reg * datad_reg;
           result <= mult0_result + mult1_result;
        end
   end
endmodule
```

Example 19. VHDL Signed Multiply-Accumulator

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY sig_altmult_accum IS
   PORT (
      a: IN SIGNED(7 DOWNTO 0);
      b: IN SIGNED (7 DOWNTO 0);
      clk: IN STD_LOGIC;
      aclr: IN STD_LOGIC;
      accum_out: OUT SIGNED (15 DOWNTO 0)
   ) ;
END sig_altmult_accum;

ARCHITECTURE rtl OF sig_altmult_accum IS
   SIGNAL a_reg, b_reg: SIGNED (7 DOWNTO 0);
   SIGNAL pdt_reg: SIGNED (15 DOWNTO 0);
   SIGNAL adder_out: SIGNED (15 DOWNTO 0);
BEGIN
   PROCESS (clk, aclr)
   BEGIN
      IF (aclr = '1') then
          a_reg <= (others => '0');
          b_reg <= (others => '0');
          pdt_reg <= (others => '0');
          adder_out <= (others => '0');
      ELSIF (rising_edge(clk)) THEN
          a_reg <= (a);
          b_reg <= (b);
          pdt_reg <= a_reg * b_reg;
          adder_out <= adder_out + pdt_reg;
      END IF;
   END process;
   accum_out <= adder_out;
END rtl;
```

Example 20. VHDL Unsigned Multiply-Adder

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY unsignedmult_add IS
   PORT (
      a: IN UNSIGNED (7 DOWNTO 0);
      b: IN UNSIGNED (7 DOWNTO 0);
```

```
      c: IN UNSIGNED (7 DOWNTO 0);
      d: IN UNSIGNED (7 DOWNTO 0);
      clk: IN STD_LOGIC;
      aclr: IN STD_LOGIC;
      result: OUT UNSIGNED (15 DOWNTO 0)
   );
END unsignedmult_add;

ARCHITECTURE rtl OF unsignedmult_add IS
   SIGNAL a_reg, b_reg, c_reg, d_reg: UNSIGNED (7 DOWNTO 0);
   SIGNAL pdt_reg, pdt2_reg: UNSIGNED (15 DOWNTO 0);
   SIGNAL result_reg: UNSIGNED (15 DOWNTO 0);
BEGIN
   PROCESS (clk, aclr)
   BEGIN
      IF (aclr = '1') THEN
         a_reg <= (OTHERS => '0');
         b_reg <= (OTHERS => '0');
         c_reg <= (OTHERS => '0');
         d_reg <= (OTHERS => '0');
         pdt_reg <= (OTHERS => '0');
         pdt2_reg <= (OTHERS => '0');
         result_reg <= (OTHERS => '0');

      ELSIF (rising_edge(clk)) THEN
         a_reg <= a;
         b_reg <= b;
         c_reg <= c;
         d_reg <= d;
         pdt_reg <= a_reg * b_reg;
         pdt2_reg <= c_reg * d_reg;
         result_reg <= pdt_reg + pdt2_reg;
      END IF;
      END PROCESS;
   result <= result_reg;
END rtl;
```

Related Information

- DSP Design Examples
- AN639: Inferring Stratix® V DSP Blocks for FIR Filtering

2.4. 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. Most of these designs can also be found on the Design Examples page on the Altera website.

Table 2. 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

- Instantiating IP Cores in HDL
  In *Introduction to Intel FPGA IP Cores*
- Design Examples
- Embedded Memory Blocks in Intel Arria 10 Devices
  In *Intel Arria 10 Core Fabric and General Purpose I/Os Handbook*

2.4.1. 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.

2.4.1.1. 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.

2.4.1.2. 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 21. 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
```

Example 22. Verilog RAM with Reset Signal that Affects RAM: Not Supported in Device Architecture

```
module bad_reset
(
    input clock,
    input reset,
    input we,
    input [7:0] data_in,
    input [4:0] address,
    output reg [7:0] data_out,
    input d,
    output reg q
);

    reg [7:0] mem [0:31];
    integer i;

    always @ (posedge clock or posedge reset)
    begin
        if (reset == 1'b1)
            q <= 0;
        else
        begin
            if (we == 1'b1)
                mem[address] <= data_in;

            data_out <= mem[address];
            q <= d;
        end
    end
endmodule
```

Related Information

Specifying Initial Memory Contents at Power-Up on page 60

2.4.1.3. 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 23. 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 Standard Edition integrated 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.

Synchronous RAM blocks require a synchronous read, so Intel Quartus Prime Standard Edition integrated synthesis packs either data output registers or read address registers into the RAM block. When the read address registers are packed into the RAM block, the read address signals connected to the RAM block contain the next value of the read address signals indexing the HDL variable, which impacts which

clock cycle the read and the write occur, and changes the read-during-write conditions. Therefore, bypass logic may still be added to the design to preserve the read-during-write behavior, even if the `no_rw_check` attribute is set.

2.4.1.4. 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.

2.4.1.5. 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 24. 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 25. 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;
```

2.4.1.6. 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.

For Intel Quartus Prime Standard Edition integrated synthesis, if you do not require the read-through-write capability, add the synthesis attribute `ramstyle="no_rw_check"` to allow the Intel Quartus Prime software to choose the read-during-write behavior of a RAM, rather than using the behavior specified by your HDL code. This attribute may prevent generation of extra bypass logic, but it is not always possible to eliminate the requirement for bypass logic.

The following examples use a blocking assignment for the write so that the data is assigned intermediately.

Example 26. 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 27. 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 28. 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 29. 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;
```

2.4.1.7. 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.

When Intel Quartus Prime integrated synthesis infers this type of RAM, it issues a warning because of the undefined read-during-write behavior.

Example 30. 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 31. 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 47

2.4.1.8. 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**—This mode matches the behavior of synchronous memory blocks.
- **Read old data**—This mode is supported only in device families that support M144K and M9K memory blocks.

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 Standard Edition integrated synthesis supports this mode by creating bypass logic around the synchronous memory block.
- **Read old data**—Synchronous memory blocks 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 memory . 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 32. Verilog HDL True Dual-Port RAM with Single Clock

```
module true_dual_port_ram_single_clock
#(parameter DATA_WIDTH = 8, 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];

    always @ (posedge clk)
    begin // Port a
        if (we_a)
        begin
            ram[addr_a] <= data_a;
```

```
            q_a <= data_a;
        end
        else
            q_a <= ram[addr_a];
    end
    always @ (posedge clk)
    begin // Port b
        if (we_b)
        begin
            ram[addr_b] <= data_b;
            q_b <= data_b;
        end
        else
            q_b <= ram[addr_b];
    end
endmodule
```

Example 33. 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 is undefined. 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 34. VHDL True Dual-Port RAM with Single Clock

```
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 signal.
    signal ram : memory_t;

begin
    process(clk)
    begin
        if(rising_edge(clk)) then -- Port A
            if(we_a = '1') then
                ram(addr_a) <= data_a;
                -- Read-during-write on same port returns NEW data
                q_a <= data_a;
            else
              -- Read-during-write on mixed port returns OLD
data
                q_a <= ram(addr_a);
            end if;
        end if;
    end process;

    process(clk)
    begin
        if(rising_edge(clk)) then -- Port B
            if(we_b = '1') then
                ram(addr_b) <= data_b;
                -- Read-during-write on same port returns NEW data
                q_b <= data_b;
            else
              -- Read-during-write on mixed port returns OLD data
                q_b <= ram(addr_b);
            end if;
        end if;
    end process;
end rtl;
```

The port behavior inferred in the Intel Quartus Prime software for the above example is:

```
PORT_A_READ_DURING_WRITE_MODE = "new_data_no_nbe_read"
PORT_B_READ_DURING_WRITE_MODE = "new_data_no_nbe_read"
MIXED_PORT_FEED_THROUGH_MODE = "old"
```

Related Information

Guideline: Customize Read-During-Write Behavior
In *Intel Arria 10 Core Fabric and General Purpose I/Os Handbook*

2.4.1.9. 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 Standard Edition integrated 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 35. 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 36. 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 37. 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 38. 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;
```

2.4.1.10. 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 Standard Edition integrated 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 39. SystemVerilog Simple Dual-Port Synchronous RAM with Byte Enable

```
module byte_enabled_simple_dual_port_ram
(
    input we, clk,
    input [5:0] waddr, raddr, // address width = 6
    input [3:0] be,         // 4 bytes per word
    input [31:0] wdata,      // byte width = 8, 4 bytes per word
    output reg [31:0] q      // byte width = 8, 4 bytes per word
);
    // use a multi-dimensional packed array
    //to model individual bytes within the word
   logic [3:0][7:0] ram[0:63];    // # words = 1 << address width

   always_ff@(posedge clk)
   begin
        if(we) begin
           if(be[0]) ram[waddr][0] <= wdata[7:0];
           if(be[1]) ram[waddr][1] <= wdata[15:8];
           if(be[2]) ram[waddr][2] <= wdata[23:16];
        if(be[3]) ram[waddr][3] <= wdata[31:24];
        end
        q <= ram[raddr];
   end
endmodule
```

Example 40. 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
port (
    we, clk : in  std_logic;
    waddr, raddr : in  integer range 0 to 63 ;     -- address width = 6
    be      : in  std_logic_vector (3 downto 0);   -- 4 bytes per word
    wdata   : in  std_logic_vector(31 downto 0);   -- byte width = 8
    q       : out std_logic_vector(31 downto 0) ); -- byte width = 8
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 3) of std_logic_vector(7 downto 0);
    type ram_t is array (0 to 63) 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 3 generate
            q(8*(i+1) - 1 downto 8*i) <= q_local(i);
    end generate unpack;

    process(clk)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                if(be(0) = '1') then
                    ram(waddr)(0) <= wdata(7 downto 0);
                end if;
                if be(1) = '1' then
                    ram(waddr)(1) <= wdata(15 downto 8);
                end if;
                if be(2) = '1' then
                    ram(waddr)(2) <= wdata(23 downto 16);
                end if;
                if be(3) = '1' then
                    ram(waddr)(3) <= wdata(31 downto 24);
                end if;
            end if;
            q_local <= ram(raddr);
        end if;
    end process;
end rtl;
```

2.4.1.11. 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 Standard Edition integrated synthesis automatically converts the initial block into a Memory Initialization File (`.mif`) for the inferred RAM.

Example 41. 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 Standard Edition integrated 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 42. 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 Standard Edition integrated synthesis automatically converts the default value into a `.mif` file for the inferred RAM.

Example 43. 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;
```

2.4.2. 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.

Note: If you use Intel Quartus Prime Standard Edition integrated synthesis, you can direct the Intel Quartus Prime software to infer ROM blocks for all sizes with the **Allow Any ROM Size for Recognition** option in the **Advanced Analysis & Synthesis Settings** dialog box.

Some synthesis tools provide options to control the implementation of inferred ROM blocks for Intel FPGA devices with synchronous memory blocks. For example, Intel Quartus Prime Standard Edition integrated synthesis provides the `romstyle` synthesis attribute to specify the type of memory block or to specify the use of regular logic instead of a dedicated memory block.

For device architectures with synchronous RAM blocks, such as the Arria series, Cyclone® series, or Stratix® series devices, 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 44. 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 45. 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 46. 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 47. 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;
```

2.4.3. Inferring Shift Registers in HDL Code

To infer shift registers, synthesis tools detect a group of shift registers of the same length, and convert them to an Intel FPGA shift register IP core.

For detection, all shift registers must have the following characteristics:

- Use the same clock and clock enable
- No other secondary signals
- 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 Standard Edition integrated synthesis uses the following guidelines:

- The Intel Quartus Prime software determines whether to infer the Intel FPGA shift register IP core 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 Standard Edition integrated synthesis determines which shift registers are implemented in RAM blocks for logic by using:
  - 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 shift register IP if the number of taps times the length between each tap is greater than or equal to 64 (*N x 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 ($W \times N \times L > 32$), the Intel Quartus Prime synthesis infers Intel FPGA shift register IP core.
- 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 Intel FPGA shift register IP core, 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.

2.4.3.1. Simple Shift Register

The examples in this section show a simple, single-bit wide, 67-bit long shift register.

Intel Quartus Prime synthesis implements the register (`W = 1` and `M = 67`) in an ALTSHIFT_TAPS IP core for supported devices and maps it to RAM in supported devices, which may be placed in dedicated RAM blocks or MLAB memory. If the length of the register is less than 67 bits, Intel Quartus Prime synthesis implements the shift register in logic.

Example 48. Verilog HDL Single-Bit Wide, 64-Bit Long Shift Register

```
module shift_1x67 (clk, shift, sr_in, sr_out);
    input clk, shift;
    input sr_in;
    output sr_out;

    reg [66:0] sr;

    always @ (posedge clk)
    begin
```

```
        if (shift == 1'b1)
        begin
            sr[66:1] <= sr[65:0];
            sr[0] <= sr_in;
        end
    end
    assign sr_out = sr[65];
endmodule
```

Example 49. VHDL Single-Bit Wide, 64-Bit Long Shift Register

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
ENTITY shift_1x67 IS
    PORT (
        clk: IN STD_LOGIC;
        shift: IN STD_LOGIC;
        sr_in: IN STD_LOGIC;
        sr_out: OUT STD_LOGIC
    );
END shift_1x67;

ARCHITECTURE arch OF shift_1x67 IS
    TYPE sr_length IS ARRAY (66 DOWNTO 0) OF STD_LOGIC;
    SIGNAL sr: sr_length;
BEGIN
    PROCESS (clk)
        BEGIN
        IF (rising_edge(clk)) THEN
            IF (shift = '1') THEN
            sr(66 DOWNTO 1) <= sr(65 DOWNTO 0);
            sr(0) <= sr_in;
            END IF;
        END IF;
    END PROCESS;
    sr_out <= sr(65);
END arch;
```

2.4.3.2. Shift Register with Evenly Spaced Taps

The following examples show a Verilog HDL and VHDL 8-bit wide, 64-bit long shift register (`W > 1` and `M = 64`) with evenly spaced taps at 15, 31, and 47.

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 50. Verilog HDL 8-Bit Wide, 64-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 [64:0];
    integer n;
    always @ (posedge clk)
        begin
        if (shift == 1'b1)
            begin
            for (n = 64; n>0; n = n-1)
                begin
                sr[n] <= sr[n-1];
                end
            sr[0] <= sr_in;
        end
```

```
    end
    assign sr_tap_one = sr[16];
    assign sr_tap_two = sr[32];
    assign sr_tap_three = sr[48];
    assign sr_out = sr[64];
endmodule
```

Example 51. VHDL 8-Bit Wide, 64-Bit Long Shift Register with Evenly Spaced Taps

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
ENTITY shift_8x64_taps IS
    PORT (
        clk: IN STD_LOGIC;
        shift: IN STD_LOGIC;
        sr_in: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        sr_tap_one: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        sr_tap_two : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        sr_tap_three: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        sr_out: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END shift_8x64_taps;

ARCHITECTURE arch OF shift_8x64_taps IS
    SUBTYPE sr_width IS STD_LOGIC_VECTOR(7 DOWNTO 0);
    TYPE sr_length IS ARRAY (63 DOWNTO 0) OF sr_width;
    SIGNAL sr: sr_length;
BEGIN
    PROCESS (clk)
    BEGIN
        IF (rising_edge(clk)) THEN
            IF (shift = '1') THEN
                sr(63 DOWNTO 1) <= sr(62 DOWNTO 0);
                sr(0) <= sr_in;
            END IF;
        END IF;
    END PROCESS;
    sr_tap_one <= sr(15);
    sr_tap_two <= sr(31);
    sr_tap_three <= sr(47);
    sr_out <= sr(63);
END arch;
```

2.5. 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.

2.5.1. 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 4

2.5.1.1. Specifying a Power-Up Value

Options available in synthesis tools allow you to specify power-up conditions for the design. Intel Quartus Prime Standard Edition integrated 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 Standard Edition integrated 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 52. 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 53. 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.

2.5.2. 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 (aclr)—highest priority
2. Asynchronous Load (aload)—not available on Intel Arria 10 devices
3. Enable (ena)
4. Synchronous Clear (sclr)
5. Synchronous Load (sload)
6. 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.

The following Verilog HDL and VHDL examples create a register with the aclr, aload, and ena control signals.

Example 54. Verilog HDL D-Type Flipflop (Register) With ena, aclr, and aload Control Signals

This example does not have adata on the sensitivity list. This is a limitation of the Verilog HDL language—there is no way to describe an asynchronous load signal (in which q toggles if adata toggles while aload is high). Despite this limitation, many synthesis tools infer an aload signal from this construct. When they perform such inference, you may see information or warning messages from the synthesis tool.

```
module dff_control(clk, aclr, aload, ena, data, adata, q);
    input clk, aclr, aload, ena, data, adata;
    output q;

    reg q;

    always @ (posedge clk or posedge aclr or posedge aload)
    begin
        if (aclr)
            q <= 1'b0;
        else if (aload)
            q <= adata;
        else if (ena)
            q <= data;
    end
endmodule
```

Example 55. VHDL D-Type Flipflop (Register) With ena, aclr, and aload Control Signals

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dff_control IS
    PORT (
        clk: IN STD_LOGIC;
        aclr: IN STD_LOGIC;
        aload: IN STD_LOGIC;
        adata: IN STD_LOGIC;
        ena: IN STD_LOGIC;
      data: IN STD_LOGIC;
q: OUT STD_LOGIC
    );
END dff_control;
ARCHITECTURE rtl OF dff_control IS
BEGIN
    PROCESS (clk, aclr, aload, adata)
    BEGIN
IF (aclr = '1') THEN
q <= '0';
ELSIF (aload = '1') THEN
q <= adata;
ELSE
            IF (rising_edge(clk)) THEN
                IF (ena     ='1') THEN
q <= data;
                END IF;
            END IF;
        END IF;
    END PROCESS;
END rtl;
```

Related Information

Clock Enable Multicycle
In *Intel Quartus Prime Timing Analyzer Cookbook*

2.5.3. 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 7

2.5.3.1. 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 56. 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;
```

2.5.3.2. 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 Standard 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 4-input LUT-based devices, such as Stratix devices, the Cyclone series, and MAX® II devices, all latches in the **User Specified and Inferred Latches** table with a single LUT in the feedback loop are free of timing hazards when a single input changes. Because of the hardware behavior of the LUT, the output does not glitch when a single input toggles between two values that are supposed to produce the same output value, such as a D-type input toggling when the enable input is inactive or a set input toggling when a `reset` input with higher priority is active. This hardware behavior of the LUT means that no cover term is required for a loop around a single LUT. The Intel Quartus Prime software uses a single LUT in the feedback loop whenever possible. A latch that has data, enable, set, and `reset` inputs in addition to the output fed back to the input cannot be implemented in a single 4-input LUT. If the Intel Quartus Prime software cannot implement the latch with a single-LUT loop because there are too many inputs, the **User Specified and Inferred Latches** table indicates that the latch is not free of timing hazards.

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 57. 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 58. 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 59. 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 Standard Edition integrated synthesis also creates safe latches when possible for instantiations of an Altera latch IP core. Altera 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 Altera 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 Altera latch IP core, Intel Quartus Prime Standard Edition integrated 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 Altera latch IP core implementation depends on the synthesis tool. Some third-party synthesis tools list the number of Altera 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.

2.6. 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.

2.6.1. 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.

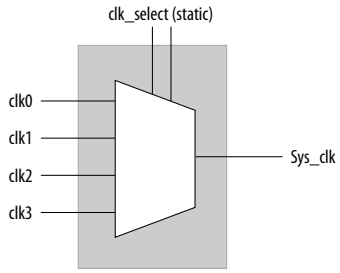
2.6.2. Clock Multiplexing

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.

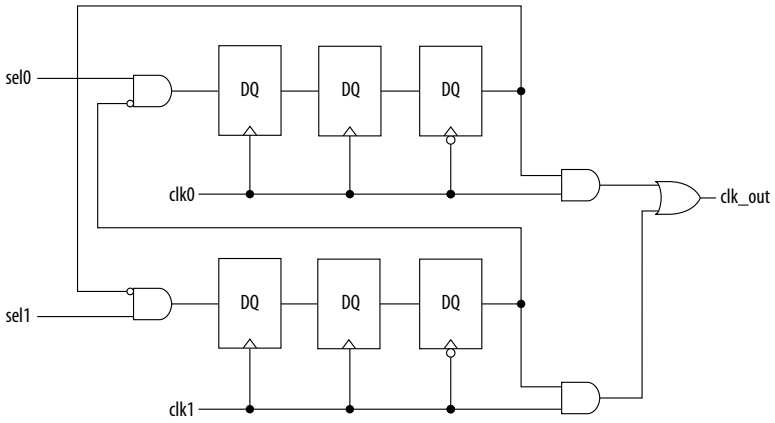
Figure 22. Simple Clock Multiplexer in a 6-Input LUT



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 23. 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 60. 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
```

Related Information

Intel FPGA IP Core Literature

2.6.3. 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.

This section explains why coding recommendations are different for Intel 4-input LUT devices and 6-input LUT devices.

2.6.3.1. Architectures with 4-Input LUTs in Logic Elements

Architectures such as Stratix devices and the Cyclone series of devices contain 4-input LUTs as the standard combinational structure in the LE.

If your design can tolerate pipelining, the fastest way to add three numbers A, B, and C in devices that use 4-input lookup tables is to addA + B, register the output, and then add the registered output to C. Adding A + B takes one level of logic (one bit is added in one LE), so this runs at full clock speed. This can be extended to as many numbers as desired.

Adding five numbers in devices that use 4-input lookup tables requires four adders and three levels of registers for a total of 64 LEs (for 16-bit numbers).

Example 61. Verilog HDL Pipelined Binary Tree

```
module binary_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, sum3, sum4;
    reg [width-1:0] sumreg1, sumreg2, sumreg3, sumreg4;
    // Registers

    always @ (posedge clk)
        begin
            sumreg1 <= sum1;
            sumreg2 <= sum2;
            sumreg3 <= sum3;
            sumreg4 <= sum4;
        end

    // 2-bit additions
    assign sum1 = A + B;
    assign sum2 = C + D;
    assign sum3 = sumreg1 + sumreg2;
    assign sum4 = sumreg3 + E;
    assign out = sumreg4;
endmodule
```

2.6.3.2. 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.

Note: You cannot pack a LAB full when using this type of coding style because of the number of LAB inputs. However, in a typical design, the Intel Quartus Prime Fitter can pack other logic into each LAB to take advantage of the unused ALMs.

Example 62. 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
```

2.6.4. 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 Primeintegrated 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.

Related Information

Intel Quartus Prime Integrated Synthesis

2.6.4.1. 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.

2.6.4.1.1. 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 63. 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.

2.6.4.1.2. SystemVerilog State Machine Coding Example

Use the following coding style to describe state machines in SystemVerilog.

Example 64. 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 Standard Edition integrated 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
```

2.6.4.2. 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.

2.6.4.2.1. 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 65. 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;
```

2.6.5. 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*.

Related Information

Advanced Synthesis Cookbook

2.6.5.1. Intel Quartus Prime Software Option for Multiplexer Restructuring

Intel Quartus Prime Standard Edition integrated 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.

2.6.5.2. 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.

2.6.5.2.1. 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.

For device families using 4-input LUTs, such as the Cyclone series and Stratix series devices, Intel Quartus Prime implements the 4:1 binary multiplexer efficiently by using two 4-input LUTs. Intel Quartus Prime decomposes larger binary multiplexers into 4:1 multiplexer blocks, possibly with a residual 2:1 multiplexer at the head.

Example 66. Verilog HDL Binary-Encoded Multiplexers

```
case (sel)
   2'b00: z = a;
   2'b01: z = b;
   2'b10: z = c;
   2'b11: z = d;
endcase
```

2.6.5.2.2. 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 67. 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
```

2.6.5.2.3. 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.

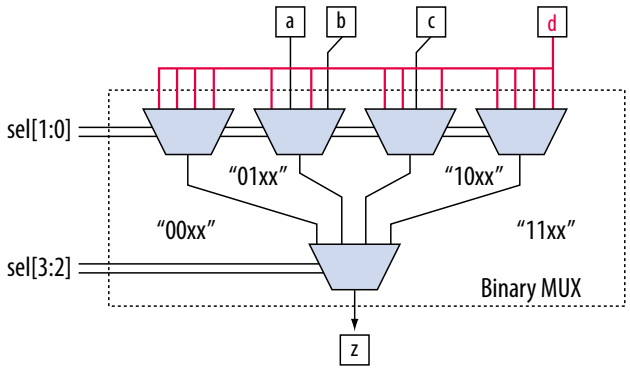
Synthesis tools commonly infer these structures from `IF`, `ELSE`, `WHEN`, `SELECT`, and `?:` statements in VHDL or Verilog HDL.

Example 68. 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 24. 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.

2.6.5.3. 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.

2.6.5.4. 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.

2.6.6. 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.

2.6.6.1. 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 Standard Edition integrated 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.

2.6.6.2. 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

2.6.6.3. 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 an incremental 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.

2.6.6.4. 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.

2.6.6.5. 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.

2.6.6.6. 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 69
- Designing with Low-Level Primitives User Guide

2.6.7. 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 Standard Edition integrated 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.In devices with 4-input LUTs, the capacity is one bit of comparison per cell, similar to an add/subtract chain. 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):

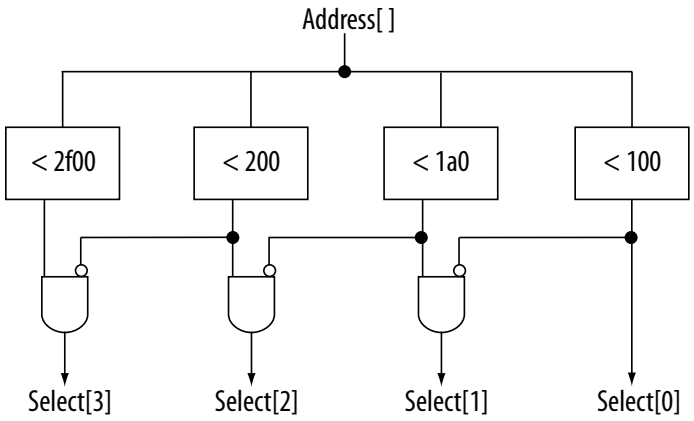
```
wire [6:0] a,b;
wire [7:0] tmp = a - b;
wire alb = tmp[7]
```

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 25. Example Logic Structure for Using Comparators to Check a Bus Value Range



2.6.8. 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.

2.7. 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 Standard Edition integrated synthesis from performing optimizations across a logic cell
- Create carry and cascade chains using CARRY, CARRY_SUM, and CASCADE primitives
- 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*.

Related Information

Designing with Low-Level Primitives User Guide

2.8. Recommended HDL Coding Styles Revision History

The following revisions history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • Renamed topic: "Use the Device Synchronous Load (sload) Signal to Initialize" to "Initialize the Device with the Synchronous Load (sload) Signal"
2017.05.08	17.0.0	• Updated example: Verilog HDL Multiply-Accumulator • 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. • 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. • Created example: Avoid this VHDL Coding Style.
2016.05.03	16.0.0	• Updated example code templates with latest coding styles.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
2015.05.04	15.0.0	Added information and reference about ramstyle attribute for sift register inference.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.

continued...

Document Version	Intel Quartus Prime Version	Changes
2014.08.18	14.0.a10.0	• Added recommendation to use register pipelining to obtain high performance in DSP designs.
2014.06.30	14.0.0	Removed obsolete MegaWizard Plug-In Manager support.
November 2013	13.1.0	Removed HardCopy device support.
June 2012	12.0.0	• Revised section on inserting Altera templates. • Code update for Example 11-51. • Minor corrections and updates.
November 2011	11.1.0	• Updated document template. • Minor updates and corrections.
December 2010	10.1.0	• Changed to new document template. • Updated Unintentional Latch Generation content. • Code update for Example 11-18.
July 2010	10.0.0	• Added support for mixed-width RAM • Updated support for no_rw_check for inferring RAM blocks • Added support for byte-enable
November 2009	9.1.0	• Updated support for Controlling Inference and Implementation in Device RAM Blocks • Updated support for Shift Registers
March 2009	9.0.0	• Corrected and updated several examples • Added support for Arria II GX devices • Other minor changes to chapter
November 2008	8.1.0	Changed to 8-1/2 x 11 page size. No change to content.
May 2008	8.0.0	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

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

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.

Related Information

- Understanding Metastability in FPGAs
  For more information about metastability due to signal synchronization, its effects in FPGAs, and how MTBF is calculated

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Reliability Report
  For information about Intel device reliability

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 97
- MTBF Optimization on page 100

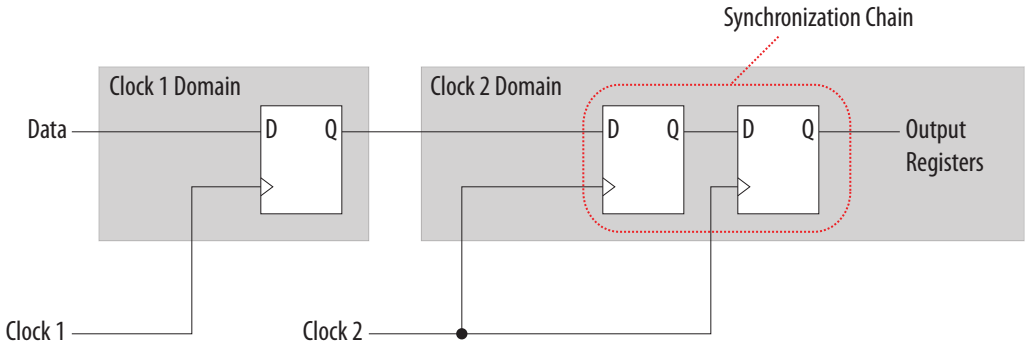
3.1.1. Synchronization Register Chains

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.

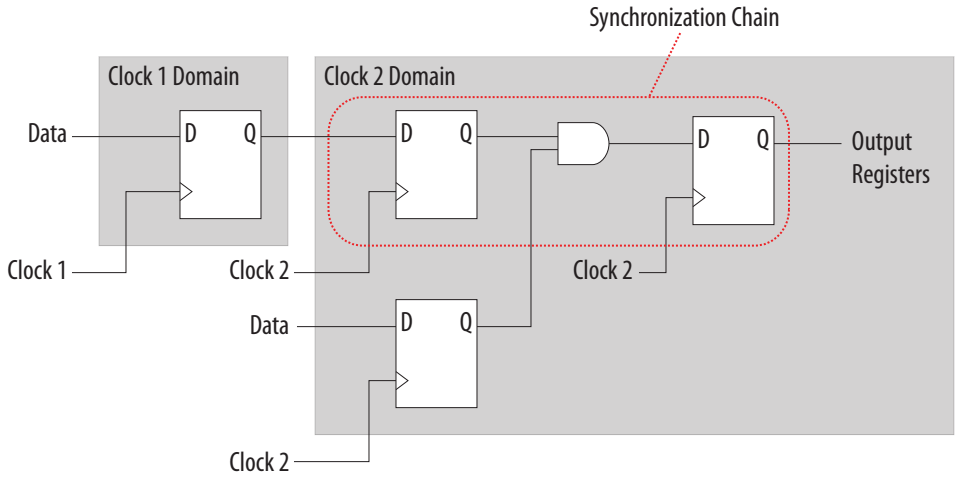
The length of the synchronization register chain is the number of registers in the synchronizing clock domain that meet the above requirements. The figure shows a sample two-register synchronization chain.

Figure 26. Sample Synchronization Register Chain



The path between synchronization registers can contain combinational logic if all registers of the synchronization register chain are in the same clock domain. The figure shows an example of a synchronization register chain that includes logic between the registers.

Figure 27. Sample Synchronization Register Chain Containing Logic



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 96

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

Identify Synchronizers for Metastability Analysis on page 96

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 98
- MTBF Optimization on page 100
- Synchronizer Data Toggle Rate in MTBF Calculation on page 100
- Understanding Metastability in FPGAs
  For more information about how metastability MTBF is calculated

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 96
- How Timing Constraints Affect Synchronizer Identification and Metastability Analysis on page 96

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 99

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 100

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 96

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. The Timing Optimization Advisor (available from the Tools menu) gives similar suggestions in the Metastability Optimization section.

Related Information

Metastability Reports on page 98

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 96

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 96

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 100

3.4.4. Optimize Metastability During Fitting

Ensure that the **Optimize Design for Metastability** setting is turned on.

Related Information

MTBF Optimization on page 100

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 101

3.4.6. Set Fitter Effort to Standard Fit instead of Auto Fit

If your design MTBF is too low after following the other guidelines, you can try increasing the Fitter effort to perform more metastability optimization. The default **Auto Fit** setting reduces the Fitter's effort after meeting the design's timing and routing requirements to reduce compilation time.

This effort reduction can result in less metastability optimization if the timing requirements are easy to meet. If **Auto Fit** reduces the Fitter's effort during your design compilation, setting the Fitter effort to **Standard Fit** might improve the design's MTBF results. To modify the **Fitter Effort**, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**.

3.4.7. 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 Altera FIFO 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.

3.4.8. 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 Standard Edition Settings File Reference Manual
  For information about all settings and constraints in the Intel Quartus Prime software.
- Tcl Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*
- Command Line Scripting
  In *Intel Quartus Prime Standard 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 100

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 3. 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 98

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 100

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 101

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
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2017.11.06	17.1.0	• Corrected broken links to other documents.
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.

continued...

Document Version	Intel Quartus Prime Version	Changes
November 2011	10.0.2	Template update.
December 2010	10.0.1	Changed to new document template.
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.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Design Compilation

Updated for Quartus® Prime Design Suite: **18.1**

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

Contents

1. Quartus® Prime Incremental Compilation for Hierarchical and Team-Based Design.....7
1.1. About Quartus® Prime Incremental Compilation.....7
1.2. Deciding Whether to Use an Incremental Compilation Flow.....7
1.2.1. Flat Compilation Flow with No Design Partitions.....7
1.2.2. Incremental Compilation Flow With Design Partitions.....8
1.2.3. Team-Based Design Flows and IP Delivery.....11
1.3. Incremental Compilation Summary.....13
1.3.1. Incremental Compilation Single Quartus Prime Project Flow.....13
1.3.2. Steps for Incremental Compilation.....13
1.3.3. Creating Design Partitions.....14
1.4. Common Design Scenarios Using Incremental Compilation.....15
1.4.1. Reducing Compilation Time When Changing Source Files for One Partition.....15
1.4.2. Optimizing a Timing-Critical Partition.....16
1.4.3. Adding Design Logic Incrementally or Working With an Incomplete Design.....17
1.4.4. Debugging Incrementally With the Signal Tap Logic Analyzer.....18
1.4.5. Functional Safety IP Implementation.....19
1.5. Deciding Which Design Blocks Should Be Design Partitions.....26
1.5.1. Impact of Design Partitions on Design Optimization.....29
1.5.2. Design Partition Assignments Compared to Physical Placement Assignments.....30
1.5.3. Using Partitions With Third-Party Synthesis Tools.....30
1.5.4. Assessing Partition Quality.....31
1.6. Specifying the Level of Results Preservation for Subsequent Compilations.....32
1.6.1. Netlist Type for Design Partitions.....33
1.6.2. Fitter Preservation Level for Design Partitions.....34
1.6.3. Where Are the Netlist Databases Saved?.....34
1.6.4. Deleting Netlists.....35
1.6.5. What Changes Initiate the Automatic Resynthesis of a Partition?.....35
1.7. Exporting Design Partitions from Separate Quartus Prime Projects.....37
1.7.1. Preparing the Top-Level Design.....39
1.7.2. Project Management— Making the Top-Level Design Available to Other Designers.....40
1.7.3. Exporting Partitions.....42
1.7.4. Viewing the Contents of a Quartus Prime Exported Partition File (.qxp).....43
1.7.5. Integrating Partitions into the Top-Level Design.....43
1.8. Team-Based Design Optimization and Third-Party IP Delivery Scenarios.....46
1.8.1. Using an Exported Partition to Send to a Design Without Including Source Files.....46
1.8.2. Creating Precompiled Design Blocks (or Hard-Wired Macros) for Reuse.....47
1.8.3. Designing in a Team-Based Environment.....49
1.8.4. Enabling Designers on a Team to Optimize Independently.....51
1.8.5. Performing Design Iterations With Lower-Level Partitions.....54
1.9. Creating a Design Floorplan With LogicLock Regions.....56
1.9.1. Creating and Manipulating LogicLock Regions.....57
1.9.2. Changing Partition Placement with LogicLock Changes.....57
1.10. Incremental Compilation Restrictions.....58
1.10.1. When Timing Performance May Not Be Preserved Exactly.....58
1.10.2. When Placement and Routing May Not Be Preserved Exactly.....58

1.10.3. Using Incremental Compilation With Quartus Prime Archive Files..... 58
1.10.4. Formal Verification Support.....59
1.10.5. Signal Probe Pins and Engineering Change Orders..... 59
1.10.6. Signal Tap Logic Analyzer in Exported Partitions..... 60
1.10.7. External Logic Analyzer Interface in Exported Partitions.....60
1.10.8. Assignments Made in HDL Source Code in Exported Partitions..... 60
1.10.9. Design Partition Script Limitations..... 60
1.10.10. Restrictions on IP Core Partitions..... 62
1.10.11. Restrictions on Arria® 10 Transceiver..... 63
1.10.12. Register Packing and Partition Boundaries.....63
1.10.13. I/O Register Packing..... 63
1.11. Scripting Support.....64
1.11.1. Tcl Scripting and Command-Line Examples..... 64
1.12. Document Revision History.....68

2. Best Practices for Incremental Compilation Partitions and Floorplan Assignments.....70

2.1. About Incremental Compilation and Floorplan Assignments.....70
2.2. Incremental Compilation Overview..... 70
2.2.1. Recommendations for the Netlist Type..... 71
2.3. Design Flows Using Incremental Compilation..... 72
2.3.1. Using Standard Flow.....72
2.3.2. Using Team-Based Flow..... 72
2.3.3. Combining Design Flows..... 72
2.3.4. Project Management in Team-Based Design Flows..... 73
2.4. Why Plan Partitions and Floorplan Assignments?..... 74
2.4.1. Partition Boundaries and Optimization.....74
2.5. Guidelines for Incremental Compilation.....77
2.5.1. General Partitioning Guidelines..... 77
2.5.2. Design Partition Guidelines.....79
2.5.3. Consider a Cascaded Reset Structure.....91
2.5.4. Design Partition Guidelines for Third-Party IP Delivery.....92
2.6. Checking Partition Quality..... 96
2.6.1. Incremental Compilation Advisor.....97
2.6.2. Design Partition Planner.....97
2.6.3. Viewing Design Partition Planner and Floorplan Side-by-Side.....99
2.6.4. Partition Statistics Report..... 100
2.6.5. Report Partition Timing in the Timing Analyzer..... 100
2.6.6. Check if Partition Assignments Impact the Quality of Results..... 100
2.7. Including SDC Constraints from Lower-Level Partitions for Third-Party IP Delivery..... 101
2.7.1. Creating an .sdc File with Project-Wide Constraints..... 102
2.7.2. Creating an .sdc with Partition-Specific Constraints..... 103
2.7.3. Consolidating the .sdc in the Top-Level Design..... 104
2.8. Introduction to Design Floorplans..... 105
2.8.1. The Difference between Logical Partitions and Physical Regions..... 105
2.8.2. Why Create a Floorplan?..... 106
2.8.3. When to Create a Floorplan..... 107
2.9. Design Floorplan Placement Guidelines.....108
2.9.1. Flow for Creating a Floorplan..... 108
2.9.2. Assigning Partitions to LogicLock Regions..... 109
2.9.3. How to Size and Place Regions.....109

2.9.4. Modifying Region Size and Origin....110
2.9.5. I/O Connections....111
2.9.6. LogicLock Resource Exclusions....111
2.9.7. Creating Non-Rectangular Regions....113
2.10. Checking Floorplan Quality....113
2.10.1. Incremental Compilation Advisor....114
2.10.2. LogicLock Region Resource Estimates....114
2.10.3. LogicLock Region Properties Statistics Report....114
2.10.4. Locate the Quartus Prime Timing Analyzer Path in the Chip Planner....114
2.10.5. Inter-Region Connection Bundles....114
2.10.6. Routing Utilization....114
2.10.7. Ensure Floorplan Assignments Do Not Significantly Impact Quality of Results 114
2.11. Recommended Design Flows and Application Examples....115
2.11.1. Create a Floorplan for Major Design Blocks....115
2.11.2. Create a Floorplan Assignment for One Design Block with Difficult Timing....116
2.11.3. Create a Floorplan as the Project Lead in a Team-Based Flow....116
2.12. Document Revision History....117

3. Quartus Prime Integrated Synthesis....119
3.1. Design Flow....119
3.1.1. Quartus Prime Integrated Synthesis Design and Compilation Flow....121
3.2. Language Support....122
3.2.1. Verilog and SystemVerilog Synthesis Support....123
3.2.2. VHDL Synthesis Support....127
3.2.3. AHDL Support....128
3.2.4. Schematic Design Entry Support....129
3.2.5. State Machine Editor....129
3.2.6. Design Libraries....129
3.2.7. Using Parameters/Generics....132
3.3. Incremental Compilation....136
3.3.1. Partitions for Preserving Hierarchical Boundaries....137
3.3.2. Parallel Synthesis....137
3.3.3. Quartus Prime Exported Partition File as Source....138
3.4. Quartus Prime Synthesis Options....138
3.4.1. Setting Synthesis Options....138
3.4.2. Optimization Technique....142
3.4.3. Auto Gated Clock Conversion....142
3.4.4. Enabling Timing-Driven Synthesis....144
3.4.5. SDC Constraint Protection....144
3.4.6. PowerPlay Power Optimization....144
3.4.7. Limiting Resource Usage in Partitions....144
3.4.8. Restructure Multiplexers....146
3.4.9. Synthesis Effort....147
3.4.10. Fitter Intial Placement Seed....147
3.4.11. State Machine Processing....147
3.4.12. Safe State Machine....151
3.4.13. Power-Up Level....152
3.4.14. Power-Up Don’t Care....153
3.4.15. Remove Duplicate Registers....153
3.4.16. Preserve Registers....153
3.4.17. Disable Register Merging/Don’t Merge Register....154

3.4.18. Noprune Synthesis Attribute/Preserve Fan-out Free Register Node..... 155
3.4.19. Keep Combinational Node/Implement as Output of Logic Cell..... 156
3.4.20. Disabling Synthesis Netlist Optimizations with dont_retime Attribute..... 157
3.4.21. Disabling Synthesis Netlist Optimizations with dont_replicate Attribute..... 157
3.4.22. Maximum Fan-Out..... 158
3.4.23. Controlling Clock Enable Signals with Auto Clock Enable Replacement and direct_enable.....159
3.5. Inferring Multiplier, DSP, and Memory Functions from HDL Code..... 160
3.5.1. Multiply-Accumulators and Multiply-Adders..... 160
3.5.2. Shift Registers..... 161
3.5.3. RAM and ROM.....161
3.5.4. Resource Aware RAM, ROM, and Shift-Register Inference.....162
3.5.5. Auto RAM to Logic Cell Conversion.....163
3.5.6. RAM Style and ROM Style—for Inferred Memory..... 163
3.5.7. RAM Style Attribute—For Shift Registers Inference.....165
3.5.8. Disabling Add Pass-Through Logic to Inferred RAMs no_rw_check Attribute.... 166
3.5.9. RAM Initialization File—for Inferred Memory.....168
3.5.10. Multiplier Style—for Inferred Multipliers..... 169
3.5.11. Full Case Attribute..... 171
3.5.12. Parallel Case..... 172
3.5.13. Translate Off and On / Synthesis Off and On.....173
3.5.14. Ignore translate_off and synthesis_off Directives..... 173
3.5.15. Read Comments as HDL..... 174
3.5.16. Use I/O Flipflops.....174
3.5.17. Specifying Pin Locations with chip_pin.....176
3.5.18. Using altera_attribute to Set Quartus Prime Logic Options.....177
3.6. Analyzing Synthesis Results.....179
3.6.1. Analysis & Synthesis Section of the Compilation Report..... 180
3.6.2. Project Navigator..... 180
3.7. Analyzing and Controlling Synthesis Messages..... 180
3.7.1. Quartus Prime Messages..... 180
3.7.2. VHDL and Verilog HDL Messages..... 181
3.8. Node-Naming Conventions in Quartus Prime Integrated Synthesis..... 184
3.8.1. Hierarchical Node-Naming Conventions..... 184
3.8.2. Node-Naming Conventions for Registers (DFF or D Flipflop Atoms).....184
3.8.3. Register Changes During Synthesis.....185
3.8.4. Preserving Register Names..... 187
3.8.5. Node-Naming Conventions for Combinational Logic Cells..... 187
3.8.6. Preserving Combinational Logic Names..... 188
3.9. Scripting Support.....189
3.9.1. Adding an HDL File to a Project and Setting the HDL Version.....189
3.9.2. Assigning a Pin..... 191
3.9.3. Creating Design Partitions for Incremental Compilation.....191
3.10. Document Revision History..... 192

4. Reducing Compilation Time.....195

4.1. Strategies to Reduce the Overall Compilation Time..... 195
4.1.1. Running Rapid Recompile..... 195
4.1.2. Enabling Multi-Processor Compilation..... 196
4.1.3. Using Incremental Compilation.....200
4.1.4. Using Block-Based Compilation..... 201

4.2. Reducing Synthesis Time and Synthesis Netlist Optimization Time..... 201
4.2.1. Settings to Reduce Synthesis Time and Synthesis Netlist Optimization Time... 201
4.2.2. Use Appropriate Coding Style to Reduce Synthesis Time..... 202
4.3. Reducing Placement Time.....202
4.3.1. Fitter Effort Setting..... 202
4.3.2. Placement Effort Multiplier Settings..... 203
4.3.3. Physical Synthesis Effort Settings..... 203
4.3.4. Preserving Placement with Incremental Compilation.....203
4.4. Reducing Routing Time..... 203
4.4.1. Identifying Routing Congestion with the Chip Planner.....204
4.5. Reducing Static Timing Analysis Time..... 205
4.6. Setting Process Priority.....205
4.7. Reducing Compilation Time Revision History..... 205

A. Quartus Prime Standard Edition User Guides..... 207

683283 | 2021.10.22

Send Feedback

1. Quartus® Prime Incremental Compilation for Hierarchical and Team-Based Design

1.1. About Quartus® Prime Incremental Compilation

This manual provides information and design scenarios to help you partition your design to take advantage of the Quartus® II incremental compilation feature.

The ability to iterate rapidly through FPGA design and debugging stages is critical. The Quartus® Prime software introduced the FPGA industry's first true incremental design and compilation flow, with the following benefits:

- Preserves the results and performance for unchanged logic in your design as you make changes elsewhere.
- Reduces design iteration time by an average of 75% for small changes in large designs, so that you can perform more design iterations per day and achieve timing closure efficiently.
- Facilitates modular hierarchical and team-based design flows, as well as design reuse and intellectual property (IP) delivery.

Quartus Prime incremental compilation supports the Arria®, Stratix®, and Cyclone® series of devices.

1.2. Deciding Whether to Use an Incremental Compilation Flow

The Quartus Prime incremental compilation feature enhances the standard Quartus Prime design flow by allowing you to preserve satisfactory compilation results and performance of unchanged blocks of your design.

1.2.1. Flat Compilation Flow with No Design Partitions

In the flat compilation flow with no design partitions, all the source code is processed and mapped during the Analysis and Synthesis stage, and placed and routed during the Fitter stage whenever the design is recompiled after a change in any part of the design. One reason for this behavior is to ensure optimal push-button quality of results. By processing the entire design, the Compiler can perform global optimizations to improve area and performance.

You can use a flat compilation flow for small designs, such as designs in CPLD devices or low-density FPGA devices, when the timing requirements are met easily with a single compilation. A flat design is satisfactory when compilation time and preserving results for timing closure are not concerns.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

1.2.1.1. Incremental Capabilities Available When A Design Has No Partitions

The Quartus Prime software has incremental compilation features available even when you do not partition your design, including Smart Compilation, Rapid Recompile, and incremental debugging. These features work in either an incremental or flat compilation flow.

1.2.1.1.1. With Smart Compilation

In any Quartus Prime compilation flow, you can use Smart Compilation to allow the Compiler to determine which compilation stages are required, based on the changes made to the design since the last smart compilation, and then skip any stages that are not required. For example, when Smart Compilation is turned on, the Compiler skips the Analysis and Synthesis stage if all the design source files are unchanged. When Smart Compilation is turned on, if you make any changes to the logic of a design, the Compiler does not skip any compilation stage. You can turn on Smart Compilation on the **Compilation Process Settings** page of the **Setting** dialog box.

Note: Arria 10 devices do not support the smart compilation feature.

Related Information

Smart Compilation online help

1.2.1.1.2. With Rapid Recompile

The Quartus Prime software also includes a Rapid Recompile feature that instructs the Compiler to reuse the compatible compilation results if most of the design has not changed since the last compilation. This feature reduces compilation times for small and isolated design changes. You do not have control over which parts of the design are recompiled using this option; the Compiler determines which parts of the design must be recompiled. The Rapid Recompile feature preserves performance and can save compilation time by reducing the amount of changed logic that must be recompiled.

1.2.1.1.3. With Signal Tap Logic Analyzer

During the debugging stage of the design cycle, you can add the Signal Tap to your design, even if the design does not have partitions. To preserve the compilation netlist for the entire design, instruct the software to reuse the compilation results for the automatically-created "Top" partition that contains the entire design.

1.2.2. Incremental Compilation Flow With Design Partitions

In the standard incremental compilation design flow, the top-level design is divided into design partitions, which can be compiled and optimized together in the top-level Quartus Prime project. You can preserve fitting results and performance for completed partitions while other parts of the design are changing, which reduces the compilation times for each design iteration.

If you use the incremental compilation feature at any point in your design flow, it is easier to accommodate the guidelines for partitioning a design and creating a floorplan if you start planning for incremental compilation at the beginning of your design cycle.

Incremental compilation is recommended for large designs and high resource densities when preserving results is important to achieve timing closure. The incremental compilation feature also facilitates team-based design flows that allow designers to create and optimize design blocks independently, when necessary.

To take advantage of incremental compilation, start by splitting your design along any of its hierarchical boundaries into design blocks to be compiled incrementally, and set each block as a design partition. The Quartus Prime software synthesizes each individual hierarchical design partition separately, and then merges the partitions into a complete netlist for subsequent stages of the compilation flow. When recompiling your design, you can use source code, post-synthesis results, or post-fitting results to preserve satisfactory results for each partition.

In a team-based environment, part of your design may be incomplete, or it may have been developed by another designer or IP provider. In this scenario, you can add the completed partitions to the design incrementally. Alternatively, other designers or IP providers can develop and optimize partitions independently and the project lead can later integrate the partitions into the top-level design.

Related Information

- Team-Based Design Flows and IP Delivery on page 11
- Incremental Compilation Summary on page 13
- Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.2.2.1. Impact of Using Incremental Compilation with Design Partitions

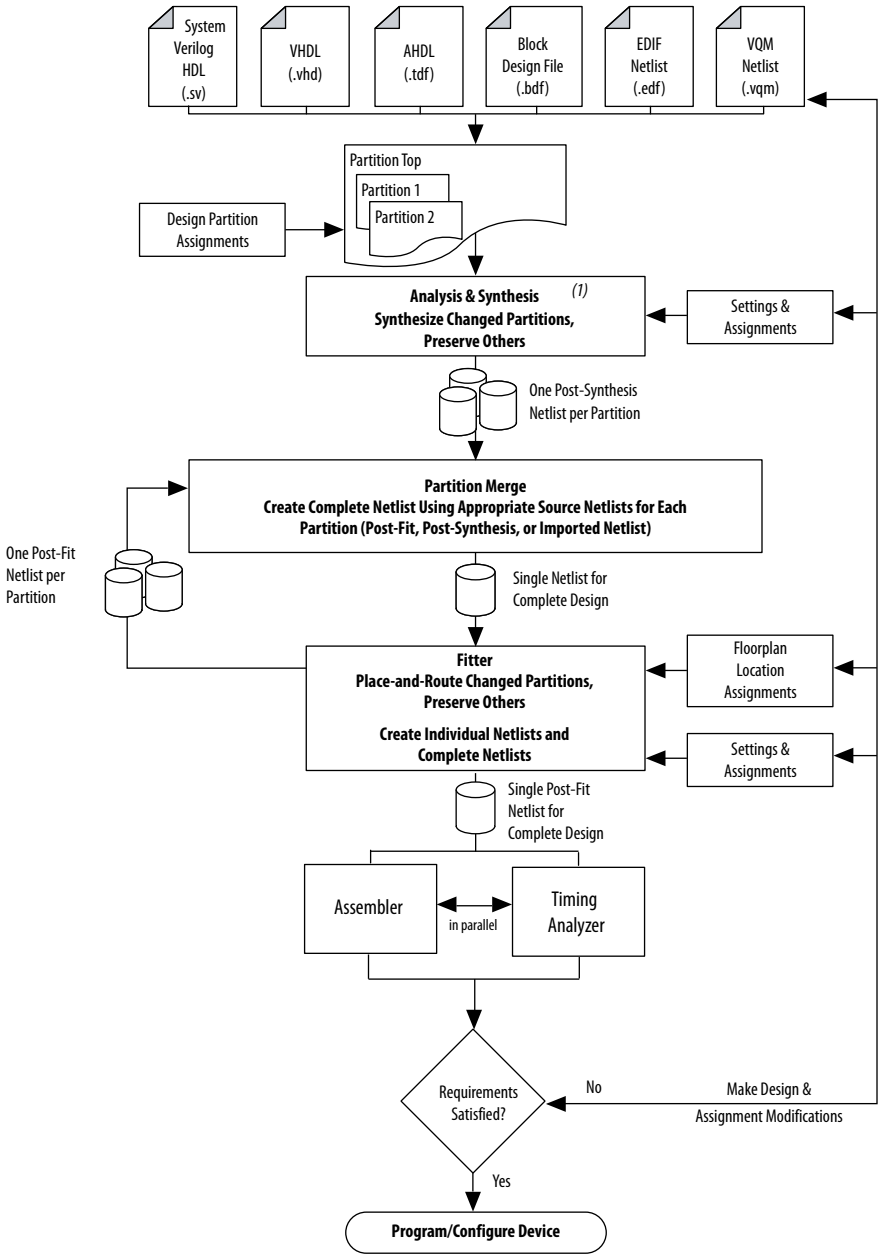
Table 1. Impact Summary of Using Incremental Compilation

Characteristic	Impact of Incremental Compilation with Design Partitions
Compilation Time Savings	Typically saves an average of 75% of compilation time for small design changes in large designs when post-fit netlists are preserved; there are savings in both Quartus Prime Integrated Synthesis and the Fitter. (1)
Performance Preservation	Excellent performance preservation when timing critical paths are contained within a partition, because you can preserve post-fitting information for unchanged partitions.
Node Name Preservation	Preserves post-fitting node names for unchanged partitions.
Area Changes	The area (logic resource utilization) might increase because cross-boundary optimizations are limited, and placement and register packing are restricted.
f_{MAX} Changes	The design's maximum frequency might be reduced because cross-boundary optimizations are limited. If the design is partitioned and the floorplan location assignments are created appropriately, there might be no negative impact on f_{MAX}.

(1) Quartus Prime incremental compilation does not reduce processing time for the early "pre-fitter" operations, such as determining pin locations and clock routing, so the feature cannot reduce compilation time if runtime is dominated by those operations.

1.2.2.2. Quartus Prime Design Stages for Incremental Compilation

Figure 1. Design Stages for Incremental Compilation



Note: When you use EDIF or VQM netlists created by third-party EDA synthesis tools, Analysis and Synthesis creates the design database, but logic synthesis and technology mapping are performed only for black boxes.

1.2.2.2.1. Analysis and Synthesis Stage

The figure above shows a top-level partition and two lower-level partitions. If any part of the design changes, Analysis and Synthesis processes the changed partitions and keeps the existing netlists for the unchanged partitions. After completion of Analysis and Synthesis, there is one post-synthesis netlist for each partition.

1.2.2.2.2. Partition Merge Stage

The Partition Merge step creates a single, complete netlist that consists of post-synthesis netlists, post-fit netlists, and netlists exported from other Quartus Prime projects, depending on the netlist type that you specify for each partition.

1.2.2.2.3. Fitter Stage

The Fitter then processes the merged netlist, preserves the placement and routing of unchanged partitions, and refits only those partitions that have changed. The Fitter generates the complete netlist for use in future stages of the compilation flow, including timing analysis and programming file generation, which can take place in parallel if more than one processor is enabled for use in the Quartus Prime software. The Fitter also generates individual netlists for each partition so that the Partition Merge stage can use the post-fit netlist to preserve the placement and routing of a partition, if specified, for future compilations.

1.2.2.2.4. How to Compare Incremental Compilation Results with Flat Design Results

If you define partitions, but want to check your compilation results without partitions in a “what if” scenario, you can direct the Compiler to ignore all partitions assignments in your project and compile the design as a "flat" netlist. When you turn on the **Ignore partitions assignments during compilation** option on the **Incremental Compilation** page, the Quartus Prime software disables all design partition assignments in your project and runs a full compilation ignoring all partition boundaries and netlists. Turning off the **Ignore partitions assignments during compilation** option restores all partition assignments and netlists for subsequent compilations.

1.2.3. Team-Based Design Flows and IP Delivery

The Quartus Prime software supports various design flows to enable team-based design and third-party IP delivery. A top-level design can include one or more partitions that are designed or optimized by different designers or IP providers, as well as partitions that will be developed as part of a standard incremental methodology.

1.2.3.1. With a Single Quartus Prime Project

In a team-based environment, part of your design may be incomplete because it is being developed elsewhere. The project lead or system architect can create empty placeholders in the top-level design for partitions that are not yet complete. Designers or IP providers can create and verify HDL code separately, and then the project lead later integrates the code into the single top-level Quartus Prime project. In this scenario, you can add the completed partitions to the design incrementally, however, the design flow allows all design optimization to occur in the top-level design for easiest design integration. Altera recommends using a single Quartus Prime project whenever possible because using multiple projects can add significant up-front and debugging time to the development cycle.

1.2.3.2. With Multiple Quartus Prime Projects

Alternatively, partition designers can design their partition in a copy of the top-level design or in a separate Quartus Prime project. Designers export their completed partition as either a post-synthesis netlist or optimized placed and routed netlist, or both, along with assignments such as LogicLock™ regions, as appropriate. The project lead then integrates each design block as a design partition into the top-level design. Altera recommends that designers export and reuse post-synthesis netlists, unless optimized post-fit results are required in the top-level design, to simplify design optimization.

1.2.3.2.1. Additional Planning Needed

Teams with a bottom-up design approach often want to optimize placement and routing of design partitions independently and may want to create separate Quartus Prime projects for each partition. However, optimizing design partitions in separate Quartus Prime projects, and then later integrating the results into a top-level design, can have the following potential drawbacks that require careful planning:

- Achieving timing closure for the full design may be more difficult if you compile partitions independently without information about other partitions in the design. This problem may be avoided by careful timing budgeting and special design rules, such as always registering the ports at the module boundaries.
- Resource budgeting and allocation may be required to avoid resource conflicts and overuse. Creating a floorplan with LogicLock regions is recommended when design partitions are developed independently in separate Quartus Prime projects.
- Maintaining consistency of assignments and timing constraints can be more difficult if there are separate Quartus Prime projects. The project lead must ensure that the top-level design and the separate projects are consistent in their assignments.

1.2.3.3. Collaboration on a Team-Based Design

A unique challenge of team-based design and IP delivery for FPGAs is the fact that the partitions being developed independently must share a common set of resources. To minimize issues that might arise from sharing a common set of resources, you can design partitions within a single Quartus Prime project or a copy of the top-level design. A common project ensures that designers have a consistent view of the top-level project framework.

For timing-critical partitions being developed and optimized by another designer, it is important that each designer has complete information about the top-level design in order to maintain timing closure during integration, and to obtain the best results. When you want to integrate partitions from separate Quartus Prime projects, the project lead can perform most of the design planning, and then pass the top-level design constraints to the partition designers. Preferably, partition designers can obtain a copy of the top-level design by checking out the required files from a source control system. Alternatively, the project lead can provide a copy of the top-level project framework, or pass design information using Quartus Prime-generated design partition scripts. In the case that a third-party designer has no information about the top-level design, developers can export their partition from an independent project if required.

Related Information

- Exporting Design Partitions from Separate Quartus Prime Projects on page 37

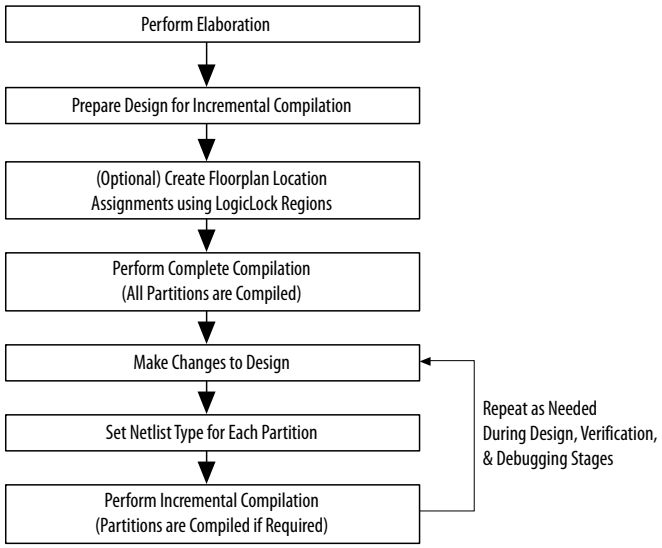
- Project Management— Making the Top-Level Design Available to Other Designers on page 40

1.3. Incremental Compilation Summary

1.3.1. Incremental Compilation Single Quartus Prime Project Flow

The figure illustrates the incremental compilation design flow when all partitions are contained in one top-level design.

Figure 2. Top-Down Design Flow



1.3.2. Steps for Incremental Compilation

For an interactive introduction to implementing an incremental compilation design flow, refer to the **Getting Started Tutorial** on the Help menu in the Quartus Prime software.

1.3.2.1. Preparing a Design for Incremental Compilation

1. Elaborate your design, or run any compilation flow (such as a full compilation) that includes the elaboration step. Elaboration is the part of the synthesis process that identifies your design's hierarchy.
2. Designate specific instances in the design hierarchy as design partitions.
3. If required for your design flow, create a floorplan with LogicLock regions location assignments for timing-critical partitions that change with future compilations. Assigning a partition to a physical region on the device can help maintain quality of results and avoid conflicts in certain situations.

Related Information

- Creating Design Partitions on page 14
- Creating a Design Floorplan With LogicLock Regions on page 56

1.3.2.2. Compiling a Design Using Incremental Compilation

The first compilation after making partition assignments is a full compilation, and prepares the design for subsequent incremental compilations. In subsequent compilations of your design, you can preserve satisfactory compilation results and performance of unchanged partitions with the **Netlist Type** setting in the Design Partitions window. The **Netlist Type** setting determines which type of netlist or source file the Partition Merge stage uses in the next incremental compilation. You can choose the Source File, Post-Synthesis netlist, or Post-Fit netlist.

Related Information

Specifying the Level of Results Preservation for Subsequent Compilations on page 32

1.3.3. Creating Design Partitions

There are several ways to designate a design instance as a design partition.

Related Information

Deciding Which Design Blocks Should Be Design Partitions on page 26

1.3.3.1. Creating Design Partitions in the Project Navigator

You can right-click an instance in the list under the **Hierarchy** tab in the Project Navigator and use the sub-menu to create and delete design partitions.

1.3.3.2. Creating Design Partitions in the Design Partitions Window

The Design Partitions window, available from the Assignments menu, allows you to create, delete, and merge partitions, and is the main window for setting the netlist type to specify the level of results preservation for each partition on subsequent compilations.

The Design Partitions window also lists recommendations at the bottom of the window with links to the Incremental Compilation Advisor, where you can view additional recommendations about partitions. The **Color** column indicates the color of each partition as it appears in the Design Partition Planner and Chip Planner.

You can right-click a partition in the window to perform various common tasks, such as viewing property information about a partition, including the time and date of the compilation netlists and the partition statistics.

When you create a partition, the Quartus Prime software automatically generates a name based on the instance name and hierarchy path. You can edit the partition name in the Design Partitions Window so that you avoid referring to them by their hierarchy path, which can sometimes be long. This is especially useful when using command-line commands or assignments, or when you merge partitions to give the partition a meaningful name. Partition names can be from 1 to 1024 characters in length and must be unique. The name can consist of alphanumeric characters and the pipe (|), colon (:), and underscore (_) characters.

Related Information

Netlist Type for Design Partitions on page 33

1.3.3.3. Creating Design Partitions 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 that follow Altera's guidelines.

The Design Partition Planner displays a visual representation of design connectivity and hierarchy, as well as partitions and entity relationships. You can explore the connectivity between entities in the design, evaluate existing partitions with respect to connectivity between entities, and try new partitioning schemes in "what if" scenarios.

When you extract design blocks from the top-level design and drag them into the Design Partition Planner, connection bundles are drawn between entities, showing the number of connections existing between pairs of entities. In the Design Partition Planner, you can then set extracted design blocks as design partitions.

The Design Partition Planner also has an **Auto-Partition** feature that creates partitions based on the size and connectivity of the hierarchical design blocks.

Related Information

Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.3.3.4. Creating Design Partitions With Tcl Scripting

You can also create partitions with Tcl scripting commands.

Related Information

Scripting Support on page 64

1.3.3.5. Automatically-Generated Partitions

The Compiler creates some partitions automatically as part of the compilation process, which appear in some post-compilation reports. For example, the `sld_hub` partition is created for tools that use JTAG hub connections, such as the SignalTap II Logic Analyzer. The `hard_block` partition is created to contain certain "hard" or dedicated logic blocks in the device that are implemented in a separate partition so that they can be shared throughout the design.

1.4. Common Design Scenarios Using Incremental Compilation

Related Information

Steps for Incremental Compilation on page 13

1.4.1. Reducing Compilation Time When Changing Source Files for One Partition

Scenario background: You set up your design to include partitions for several of the major design blocks, and now you have just performed a lengthy compilation of the entire design. An error is found in the HDL source file for one partition and it is being fixed. Because the design is currently meeting timing requirements, and the fix is not expected to affect timing performance, it makes sense to compile only the affected partition and preserve the rest of the design.

Use the flow in this example to update the source file in one partition without having to recompile the other parts of the design. To reduce the compilation time, instruct the software to reuse the post-fit netlists for the unchanged partitions. This flow also preserves the performance of these blocks, which reduces additional timing closure efforts.

Perform the following steps to update a single source file:

1. Apply and save the fix to the HDL source file.
2. On the Assignments menu, open the **Design Partitions window**.
3. Change the netlist type of each partition, including the top-level entity, to **Post-Fit** to preserve as much as possible for the next compilation.
   - The Quartus Prime software recompiles partitions by default when changes are detected in a source file. You can refer to the Partition Dependent Files table in the Analysis and Synthesis report to determine which partitions were recompiled. If you change an assignment but do not change the logic in a source file, you can set the netlist type to **Source File** for that partition to instruct the software to recompile the partition's source design files and its assignments.
4. Click **Start Compilation** to incrementally compile the fixed HDL code. This compilation should take much less time than the initial full compilation.
5. Simulate the design to ensure that the error is fixed, and use the Timing Analyzer report to ensure that timing results have not degraded.

Related Information

List of Compilation and Simulation Reports online help

1.4.2. Optimizing a Timing-Critical Partition

Scenario background: You have just performed a lengthy full compilation of a design that consists of multiple partitions. The Timing Analyzer reports that the clock timing requirement is not met, and you have to optimize one particular partition. You want to try optimization techniques such as raising the Placement Effort Multiplier and running Design Space Explorer II. Because these techniques all involve significant compilation time, you should apply them to only the partition in question.

Use the flow in this example to optimize the results of one partition when the other partitions in the design have already met their requirements. You can use this flow iteratively to lock down the performance of one partition, and then move on to optimization of another partition.

Perform the following steps to preserve the results for partitions that meet their timing requirements, and to recompile a timing-critical partition with new optimization settings:

1. Open the **Design Partitions** window.
2. For the partition in question, set the netlist type to **Source File**.
   - If you change a setting that affects only the Fitter, you can save additional compilation time by setting the netlist type to **Post-Synthesis** to reuse the synthesis results and refit the partition.
3. For the remaining partitions (including the top-level entity), set the netlist type to **Post-Fit**.

- You can optionally set the **Fitter Preservation Level** on the **Advanced** tab in the **Design Partitions Properties** dialog box to **Placement** to allow for the most flexibility during routing.

4. Apply the desired optimization settings.
5. Click **Start Compilation** to perform incremental compilation on the design with the new settings. During this compilation, the Partition Merge stage automatically merges the critical partition's new synthesis netlist with the post-fit netlists of the remaining partitions. The Fitter then refits only the required partition. Because the effort is reduced as compared to the initial full compilation, the compilation time is also reduced.

To use Design Space Explorer II, perform the following steps:

1. Repeat steps 1–3 of the previous procedure.
2. Save the project and run Design Space Explorer II.

1.4.3. Adding Design Logic Incrementally or Working With an Incomplete Design

Scenario background: You have one or more partitions that are known to be timing-critical in your full design. You want to focus on developing and optimizing this subset of the design first, before adding the rest of the design logic.

Use this flow to compile a timing-critical partition or partitions in isolation, optionally with extra optimizations turned on. After timing closure is achieved for the critical logic, you can preserve its content and placement and compile the remaining partitions with normal or reduced optimization levels. For example, you may want to compile an IP block that comes with instructions to perform optimization before you incorporate the rest of your custom logic.

To implement this design flow, perform the following steps:

1. Partition the design and create floorplan location assignments. For best results, ensure that the top-level design includes the entire project framework, even if some parts of the design are incomplete and are represented by an empty wrapper file.
2. For the partitions to be compiled first, in the Design Partitions window, set the netlist type to **Source File**.
3. For the remaining partitions, set the netlist type to **Empty**.
4. To compile with the desired optimizations turned on, click **Start Compilation**.
5. Check the Timing Analyzer reports to ensure that timing requirements are met. If so, proceed to step 6. Otherwise, repeat steps 4 and 5 until the requirements are met.
6. In the Design Partitions window, set the netlist type to **Post-Fit** for the first partitions. You can set the **Fitter Preservation Level** on the **Advanced** tab in the **Design Partitions Properties** dialog box to **Placement** to allow more flexibility during routing if exact placement and routing preservation is not required.

7. Change the netlist type from **Empty** to **Source File** for the remaining partitions, and ensure that the completed source files are added to the project.
8. Set the appropriate level of optimizations and compile the design. Changing the optimizations at this point does not affect any fitted partitions, because each partition has its netlist type set to **Post-Fit**.
9. Check the Timing Analyzer reports to ensure that timing requirements are met. If not, make design or option changes and repeat step 8 and step 9 until the requirements are met.

The flow in this example is similar to design flows in which a module is implemented separately and is later merged into the top-level. Generally, optimization in this flow works only if each critical path is contained within a single partition. Ensure that if there are any partitions representing a design file that is missing from the project, you create a placeholder wrapper file to define the port interface.

Related Information

- Designing in a Team-Based Environment on page 49
- Deciding Which Design Blocks Should Be Design Partitions on page 26
- Empty Partitions on page 40

1.4.4. Debugging Incrementally With the Signal Tap Logic Analyzer

Scenario background: Your design is not functioning as expected, and you want to debug the design using the Signal Tap Logic Analyzer. To maintain reduced compilation times and to ensure that you do not negatively affect the current version of your design, you want to preserve the synthesis and fitting results and add the Signal Tap to your design without recompiling the source code.

Use this flow to reduce compilation times when you add the logic analyzer to debug your design, or when you want to modify the configuration of the Signal Tap File without modifying your design logic or its placement.

It is not necessary to create design partitions in order to use the Signal Tap incremental compilation feature. The Signal Tap Logic Analyzer acts as its own separate design partition.

Perform the following steps to use the Signal Tap Logic Analyzer in an incremental compilation flow:

1. Open the Design Partitions window.
2. Set the netlist type to **Post-fit** for all partitions to preserve their placement.
   - The netlist type for the top-level partition defaults to **Source File**, so be sure to change this "Top" partition in addition to any design partitions that you have created.
3. If you have not already compiled the design with the current set of partitions, perform a full compilation. If the design has already been compiled with the current set of partitions, the design is ready to add the Signal Tap Logic Analyzer.
4. Set up your SignalTap II File using the **post-fitting** filter in the **Node Finder** to add signals for logic analysis. This allows the Fitter to add the SignalTap II logic to the post-fit netlist without modifying the design results.

To add signals from the pre-synthesis netlist, set the partition's netlist type to **Source File** and use the **presynthesis** filter in the **Node Finder**. This allows the software to resynthesize the partition and to tap directly to the pre-synthesis node names that you choose. In this case, the partition is resynthesized and refit, so the placement is typically different from previous fitting results.

Related Information

Design Debugging Using the SignalTap II Embedded Logic Analyzer documentation

1.4.5. Functional Safety IP Implementation

In functional safety designs, recertification is required when logic is modified in safety or standard areas of the design. Recertification is required because the FPGA programming file has changed. You can reduce the amount of required recertification if you use the functional safety separation flow in the software. By partitioning your safety IP (SIP) from standard logic, you ensure that the safety critical areas of the design remain the same when the standard areas in your design are modified. The safety-critical areas remain the same at the bit level.

The functional safety separation flow supports only Cyclone IV and Cyclone V device families.

Related Information

AN 704: FPGA-based Safety Separation Design Flow for Rapid Functional Safety Certification

This design flow significantly reduces the certification efforts for the lifetime of an FPGA-based industrial system containing both safety critical and nonsafety critical components.

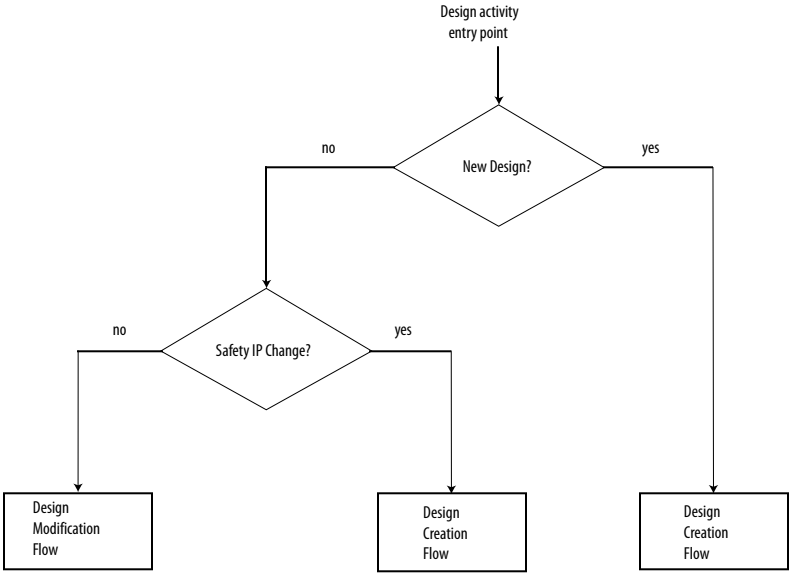
1.4.5.1. Software Tool Impact on Safety

The Quartus Prime software can partition your design into safety partitions and standard partitions, but the Quartus Prime software does not perform any online safety-related functionality. The Quartus Prime software generates a bitstream that performs the safety functions. For the purpose of compliance with a functional safety standard, the Quartus Prime software should be considered as an offline support tool.

1.4.5.2. Functional Safety Separation Flow

The functional safety separation flow consists of two separate work flows. The design creation flow and the design modification flow both use incremental compilation, but the two flows have different use-case scenarios.

Figure 3. Functional Safety Separation Flow

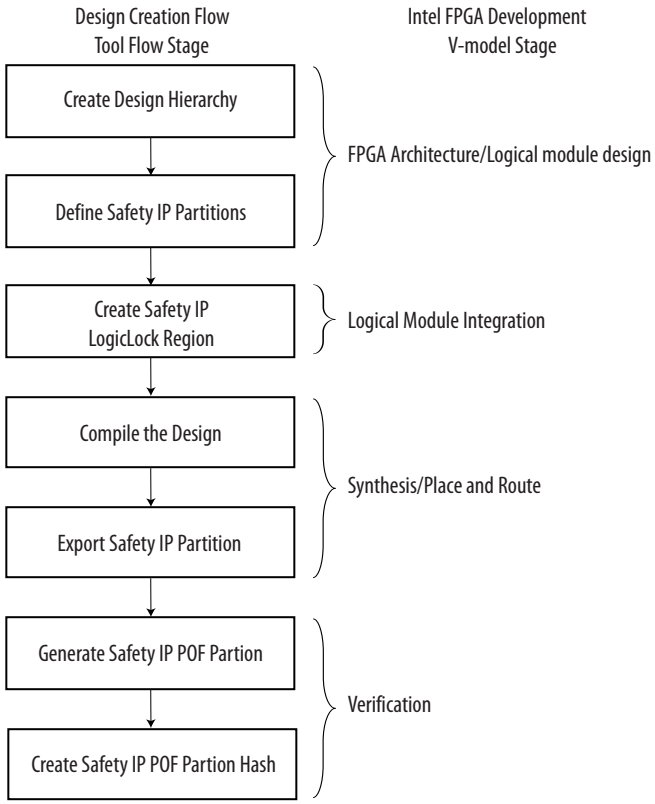


1.4.5.2.1. Design Creation Flow

The design creation flow describes the necessary steps for initial design creation in a way that allows you to modify your design. Some of the steps are architectural constraints and the remaining steps are steps that you need to perform in the Quartus Prime software. Use the design creation flow for the first pass certification of your product.

When you make modifications to the safety IP in your design, you must use the design creation flow.

Figure 4. Design Creation Flow



The design creation flow becomes active when you have a valid safety IP partition in your Quartus Prime project and that safety IP partition does not have place and route data from a previous compile. In the design creation flow, the Assembler generates a Partial Settings Mask (`.psm`) file for each safety IP partition. Each `.psm` file contains a list of programming bits for its respective safety IP partition.

The Quartus Prime software determines whether to use the design creation flow or design modification flow on a per partition basis. It is possible to have multiple safety IP partitions in a design where some are running the design creation flow and others are running the design modification flow.

To reset the complete design to the design creation flow, remove the previous place and route data by cleaning the project (removing the dbs). Alternatively, use the partition import flow, to selectively reset the design. You can remove the netlists for the imported safety IP partitions individually using the **Design Partitions** window.

Related Information

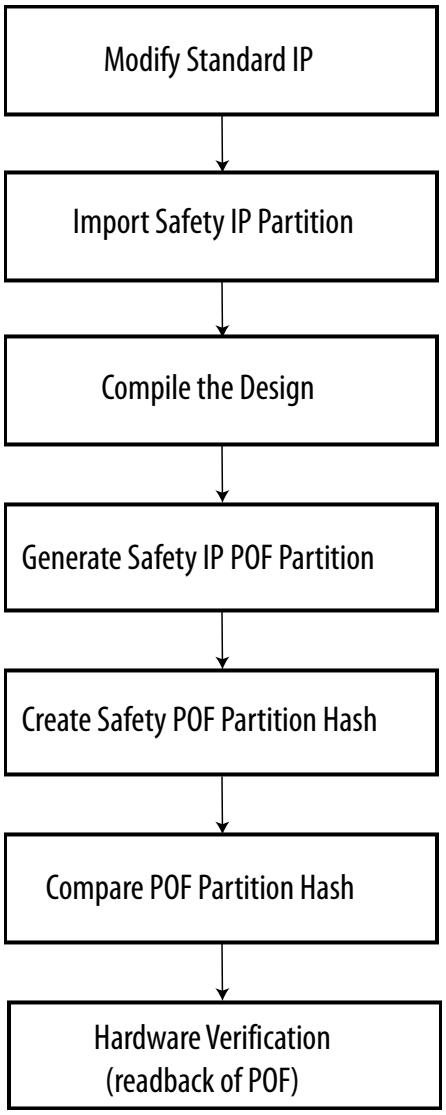
- Exporting and Importing Your Safety IP on page 26
- Design Partitions Window online help

1.4.5.2.2. Design Modification Flow

The design modification flow describes the necessary steps to make modifications to the standard IP in your design. This flow ensures that the previously compiled safety IP that the project uses remains unchanged when you change or compile standard IP.

Use the design modification flow only after you qualify your design in the design creation flow.

Figure 5. Design Modification Flow



When the design modification flow is active for a safety IP partition, the Fitter runs in Strict Preservation mode for that partition. The Assembler performs run-time checks that compare the Partial Settings Mask information matches the `.psm` file generated in the design creation flow. If the Assembler detects a mismatch, a "Bad Mask!" or "ASM_STRICT_PRESERVATION_BITS_UTILITY::compare_masked_byte_array failed" internal error message is shown. If you see either error message while compiling your design, contact your Intel support representative for assistance.

When a change is made to any HDL source file that belongs to a safety IP, the default behavior of the Quartus Prime software is to resynthesize and perform a clean place and route for that partition, which then activates the design creation flow for that partition. To change this default behavior and keep the design modification flow active, do the following:

- Use the partition export/import flow.

or

- Use the Design Partitions window to modify the design partition properties and turn on **Ignore changes in source files and strictly use the specified netlist, if available**.

The Fitter applies the same design flow to all partitions that belong to the same safety IP. If more than one safety IP is used in the design, the Fitter may evoke different flows for different safety IPs.

Note: If your safety IP is a sub-block in a Platform Designer system, every time you regenerate HDL for the Platform Designer system, the timestamp for the safety IP HDL changes. This results in resynthesis of the safety IP, unless the default behavior (described above) is changed.

Related Information

- Exporting and Importing Your Safety IP on page 26
- Design Partitions Window online help

1.4.5.3. How to Turn On the Functional Safety Separation Flow

Every safety-related IP component in your design should be implemented in a partition(s) so the safety IPs are protected from recompilation. Use the global assignment PARTITION_ENABLE_STRICT_PRESERVATION to identify safety IP in your design.

```
set_global_assignment -name PARTITION_ENABLE_STRICT_PRESERVATION <ON/OFF> -
section_id <partition_name>
```

When this global assignment is designated as ON for a partition, the partition is protected from recompilation, exported as a safety IP, and included in the safety IP POF mask. Specifying the value as ON for any partition turns on the functional safety separation flow.

When this global assignment is designated as OFF, the partition is considered as standard IP or as not having a PARTITION_ENABLE_STRICT_PRESERVATION assignment at all. Logic that is not assigned to a partition is considered as part of the top partition and treated as standard logic.

Note: Only partitions and I/O pins can be assigned to SIP.

A partition assigned to safety IP can contain safety logic only. If the parent partition is assigned to a safety IP, then all the child partitions for this parent partition are considered as part of the safety IP. If you do not explicitly specify a child partition as a safety IP, a critical warning notifies you that the child partition is treated as part of a safety IP.

A design can contain several safety IPs. All the partitions containing logic that implements a single safety IP function should belong with the same top-level parent partition.

You can also turn on the functional safety separation flow from the **Design Partition Properties** dialog box. Click the **Advanced** tab and turn on **Allow partition to be strictly preserved for safety**.

When the functional safety separation flow is active, you can view which partitions in your design have the Strict Preservation property turned on. The **Design Partitions** window displays a on or off value for safety IP in your design (in the **Strict Preservation** column).

1.4.5.4. Preservation of Device Resources

The preservation of the partition's netlist atoms and the atoms placement and routing, in the design modification flow, is done by setting the netlist type to **Post-fit** with the Fitter preservation level set to **Placement and Routing Preserved**.

1.4.5.5. Preservation of Placement in the Device with LogicLock

In order to fix the safety IP logic into specific areas of the device, you should define LogicLock regions. By using preserved LogicLock regions, device placement is reserved for the safety IP to prevent standard logic from being placed into the unused resources of the safety IP region. You establish a fixed size and origin to ensure location preservation. You need to use LogicLock to ensure a valid safety IP POF mask is generated when you turn on the functional safety separation flow. The POF comparison tool for functional safety can check that the safety region is unchanged between compiles. A LogicLock region assigned to a safety IP can only contain safety IP logic.

1.4.5.6. Assigning I/O Pins

You use a global assignment or the **Design Partition Properties** dialog box to specify that a pin is assigned to a safety IP partition.

Use the following global assignment to assign a pin to a safety IP partition:

```
set_instance_assignment -name ENABLE_STRICT_PRESERVATION ON/OFF -to <hpath> -section_id
<region_name>
```

- <*hpath*> refers to an I/O pin (pad).
- <*region_name*> refers to the top-level safety IP partition name.

A value of ON indicates that the pin is a safety pin that should be preserved with the safety IP block. A value of OFF indicates that the pin that connects to the safety IP, should be treated as a standard pin, and is not preserved with the safety IP.

You also turn on strict preservation for I/O pins in the **Design Partition Properties** dialog box. Click the **Advanced** tab and choose **On** for I/O pins that you want to preserve.

Note: All pins that connect to a safety IP partition must have an explicit assignment. The software reports an error if a pin that connects to the safety IP partition does not have an assignment or if a pin does not connect to the specified <*region_name*>.

If an IO_REG group contains a pin that is assigned to a safety IP partition, all of the pins in the IO_REG group are reserved for the safety IP partition. All pins in the IO_REG group must be assigned to the same safety IP partition, and none of the pins in the group can be assigned to standard signals.

1.4.5.7. General Guidelines for Implementation

- An internal clock source, such as a PLL, should be implemented in a safe partition.
- An I/O pin driving the external clock should be indicated as a safety pin.
- To export a safety IP containing several partitions, the top-level partition for the safety IP should be exported. A safety IP containing several partitions is flattened and converted into a single partition during export. This hierarchical safety IP is flattened to ensure bit-level settings are preserved.
- Hard blocks implemented in a safe partition needs to stay with the safe partition.

1.4.5.8. Reports for Safety IP

When you have the functional safety separation flow turned on, the Quartus Prime software displays safety IP and standard IP information in the Fitter report.

1.4.5.8.1. Fitter Report

The Fitter report includes information for each safety IP and the respective partition and I/O usage. The report contains the following information:

- Safety IP name defined as the name of the top-level safety IP partition
- Effective design flow for the safety IP
- Names of all partitions that belong to the safety IP
- Number of safety/standard inputs to the safety IP
- Number of safety/standard outputs to the safety IP
- LogicLock region names along with size and locations for the regions
- I/O pins used for the respective safety IP in your design
- Safety-related error messages

1.4.5.9. SIP Partial Bitstream Generation

The Programmer generates a bitstream file containing only the bits for a safety IP. This partial preserved bitstream (`.ppb`) file is for the safety IP region mask. The command lines to generate the partial bitstream file are the following:

- `quartus_cpf --genppb safe1.psm design.sof safe1.rbf.ppb`
- `quartus_cpf -c safe1.psm safe1.rbf.ppb`

The `.ppb` file is generated in two steps.

1. Generation of partial SOF.
2. Generation of `.ppb` file using the partial SOF.

The `.psm` file, `.ppb` file, and MD5 hash signature (`.md5.sign`) file created during partial bitstream generation should be archived for use in future design modification flow compiles.

1.4.5.10. Exporting and Importing Your Safety IP

Safety IP Partition Export

After you have successfully compiled the safety IP(s) in the Quartus Prime software, save the safety IP partition place and route information for use in any subsequent design modification flow. Saving the partition information allows the safety IP to be imported to a clean Quartus Prime project where no previous compilation results have been removed (even if the version of the Quartus Prime software being used is newer than the Quartus Prime software version with which the safety IP was originally compiled). Use the **Design Partitions** window to export the design partition. Verify that only the post-fit netlist and export routing options are turned on when you generate the `.qxp` file for each safety IP. The `.qxp` files should be archived along with the partial bitstream files for use in later design modification flow compiles.

Safety IP Partition Import

You can import a previously exported safety IP partition into your Quartus Prime project. There are two use-cases for this.

- (Optional) Import into the original project to ensure that any potential source code changes do not trigger the design creation flow unintentionally.
- Import into a new or clean project where you want to use the design modification flow for the safety IP. As the exported partition is independent of your Quartus Prime software version, you can import the `.qxp` into a future Quartus Prime software release.

To import a previously exported design partition, use the **Design Partitions** window and import the `.qxp`.

1.4.5.11. POF Comparison Tool for Verification

There is a separate safe/standard partitioning verification tool that is licensed to safety users. Along with the `.ppb` file, a `.md5.sign` file is generated. The MD5 hash signature can be used for verification. For more detailed verification, the POF comparison tool should be used. This POF comparison tool is available in the Altera Functional Safety Data Package.

1.5. Deciding Which Design Blocks Should Be Design Partitions

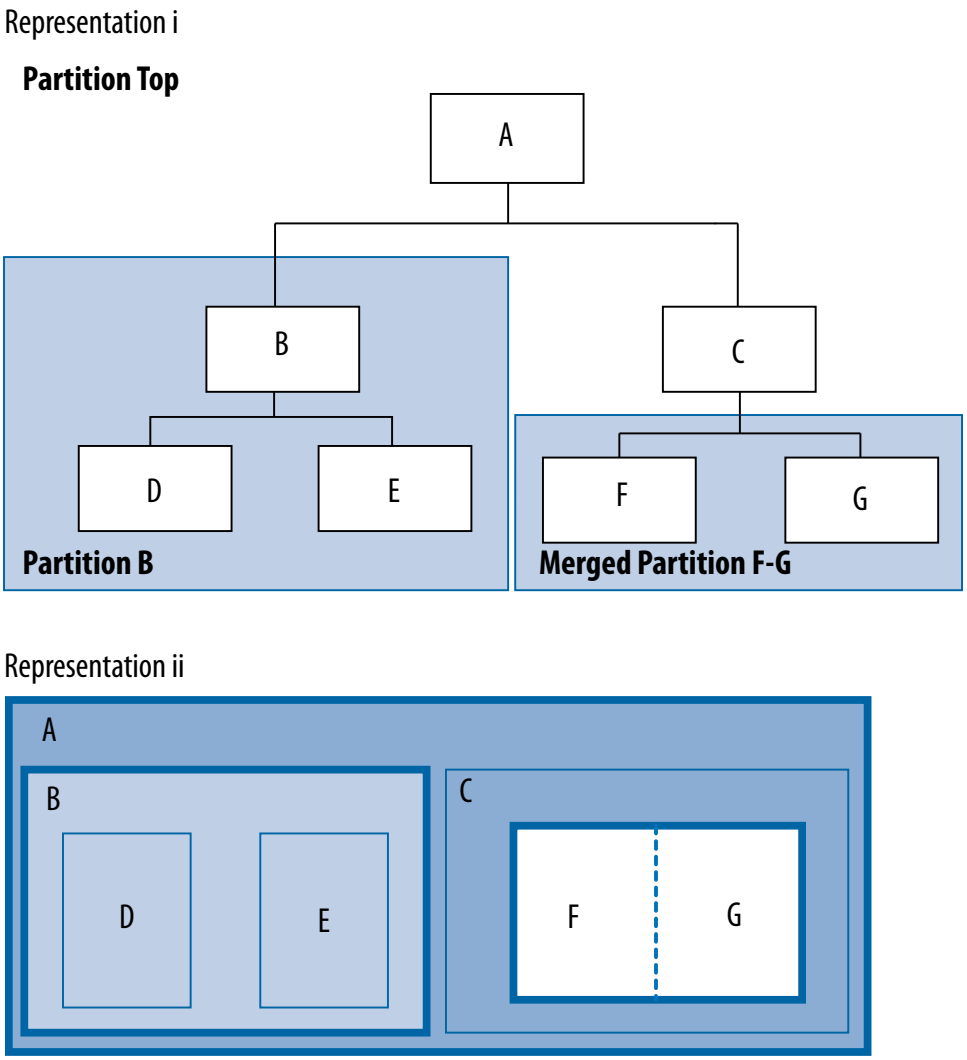
The incremental compilation design flow requires more planning than flat compilations. For example, you might have to structure your source code or design hierarchy to ensure that logic is grouped correctly for optimization.

It is a common design practice to create modular or hierarchical designs in which you develop each design entity separately, and then instantiate them in a higher-level entity, forming a complete design. The Quartus Prime software does not automatically consider each design entity or instance to be a design partition for incremental compilation; instead, you must designate one or more design hierarchies below the top-level project as a design partition. Creating partitions might prevent the Compiler from performing optimizations across partition boundaries. However, this allows for separate synthesis and placement for each partition, making incremental compilation possible.

Partitions must have the same boundaries as hierarchical blocks in the design because a partition cannot be a portion of the logic within a hierarchical entity. You can merge partitions that have the same immediate parent partition to create a single partition that includes more than one hierarchical entity in the design. When you declare a partition, every hierarchical instance within that partition becomes part of the same partition. You can create new partitions for hierarchical instances within an existing partition, in which case the instances within the new partition are no longer included in the higher-level partition, as described in the following example.

In the figure below, a complete design is made up of instances **A**, **B**, **C**, **D**, **E**, **F**, and **G**. The shaded boxes in Representation i indicate design partitions in a "tree" representation of the hierarchy. In Representation ii, the lower-level instances are represented inside the higher-level instances, and the partitions are illustrated with different colored shading. The top-level partition, called "Top", automatically contains the top-level entity in the design, and contains any logic not defined as part of another partition. The design file for the top level may be just a wrapper for the hierarchical instances below it, or it may contain its own logic. In this example, partition **B** contains the logic in instances **B**, **D**, and **E**. Entities **F** and **G** were first identified as separate partitions, and then merged together to create a partition **F**-**G**. The partition for the top-level entity **A**, called "Top", includes the logic in one of its lower-level instances, **C**, because **C** was not defined as part of any other partition.

Figure 6. Partitions in a Hierarchical Design



You can create partition assignments to any design instance. The instance can be defined in HDL or schematic design, or come from a third-party synthesis tool as a VQM or EDIF netlist instance.

To take advantage of incremental compilation when source files change, create separate design files for each partition. If you define two different entities as separate partitions but they are in the same design file, you cannot maintain incremental compilation because the software would have to recompile both partitions if you changed either entity in the design file. Similarly, if two partitions rely on the same lower-level entity definition, changes in that lower-level affect both partitions.

The remainder of this section provides information to help you choose which design blocks you should assign as partitions.

1.5.1. Impact of Design Partitions on Design Optimization

The boundaries of your design partitions can impact the design's quality of results. Creating partitions might prevent the Compiler from performing logic optimizations across partition boundaries, which allows the software to synthesize and place each partition separately in an incremental flow. Therefore, consider partitioning guidelines to help reduce the effect of partition boundaries.

Whenever possible, register all inputs and outputs of each partition. This helps avoid any delay penalty on signals that cross partition boundaries and keeps each register-to-register timing path within one partition for optimization. In addition, minimize the number of paths that cross partition boundaries. If there are timing-critical paths that cross partition boundaries, rework the partitions to avoid these inter-partition paths. Including as many of the timing-critical connections as possible inside a partition allows you to effectively apply optimizations to that partition to improve timing, while leaving the rest of the design unchanged.

Avoid constant partition inputs and outputs. You can also merge two or more partitions to allow cross-boundary optimizations for paths that cross between the partitions, as long as the partitions have the same parent partition. Merging related logic from different hierarchy blocks into one partition can be useful if you cannot change the design hierarchy to accommodate partition assignments.

If critical timing paths cross partition boundaries, you can perform timing budgeting and make timing assignments to constrain the logic in each partition so that the entire timing path meets its requirements. In addition, because each partition is optimized independently during synthesis, you may have to perform resource allocation to ensure that each partition uses an appropriate number of device resources. If design partitions are compiled in separate Quartus Prime projects, there may be conflicts related to global routing resources for clock signals when the design is integrated into the top-level design. You can use the Global Signal logic option to specify which clocks should use global or regional routing, use the ALTCLK_CTRL IP core to instantiate a clock control block and connect it appropriately in both the partitions being developed in separate Quartus Prime projects, or find the compiler-generated clock control node in your design and make clock control location assignments in the Assignment Editor.

1.5.1.1. Turning On Supported Cross-boundary Optimizations

You can improve the optimizations performed between design partitions by turning on supported cross-boundary optimizations. These optimizations are turned on a per partition basis and you can select the optimizations as individual assignments. This allows the cross-boundary optimization feature to give you more control over the optimizations that work best for your design. You can turn on the cross-boundary optimizations for your design partitions on the **Advanced** tab of the **Design Partition Properties** dialog box. Once you change the optimization settings, the Quartus Prime software recompiles your partition from source automatically. Cross-boundary optimizations include the following: propagate constants, propagate inversions on partition inputs, merge inputs fed by a common source, merge electrically equivalent bidirectional pins, absorb internal paths, and remove logic connected to dangling outputs.

Cross-boundary optimizations are implemented top-down from the parent partition into the child partition, but not vice-versa. Also, cross-boundary optimizations cannot be enabled for partitions that allow multiple personas (partial reconfiguration partitions).

Related Information

Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.5.2. Design Partition Assignments Compared to Physical Placement Assignments

Design partitions for incremental compilation are logical partitions, which is different from physical placement assignments in the device floorplan. A logical design partition does not refer to a physical area of the device and does not directly control the placement of instances. A logical design partition sets up a virtual boundary between design hierarchies so that each is compiled separately, preventing logical optimizations from occurring between them. When the software compiles the design source code, the logic in each partition can be placed anywhere in the device unless you make additional placement assignments.

If you preserve the compilation results using a Post-Fit netlist, it is not necessary for you to back-annotate or make any location assignments for specific logic nodes. You should not use the incremental compilation and logic placement back-annotation features in the same Quartus Prime project. The incremental compilation feature does not use placement "assignments" to preserve placement results; it simply reuses the netlist database that includes the placement information.

You can assign design partitions to physical regions in the device floorplan using LogicLock region assignments. In the Quartus Prime software, LogicLock regions are used to constrain blocks of a design to a particular region of the device. Altera recommends using LogicLock regions for timing-critical design blocks that will change in subsequent compilations, or to improve the quality of results and avoid placement conflicts in some cases.

Related Information

- Creating a Design Floorplan With LogicLock Regions on page 56
- Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.5.3. Using Partitions With Third-Party Synthesis Tools

If you are using a third-party synthesis tool, set up your tool to create a separate VQM or EDIF netlist for each hierarchical partition. In the Quartus Prime software, assign the top-level entity from each netlist to be a design partition. The VQM or EDIF netlist file is treated as the source file for the partition in the Quartus Prime software.

1.5.3.1. Synopsys Synplify Pro/Premier and Mentor Graphics Precision RTL Plus

The Synplify Pro and Synplify Premier software include the MultiPoint synthesis feature to perform incremental synthesis for each design block assigned as a Compile Point in the user interface or a script. The Precision RTL Plus software includes an incremental synthesis feature that performs block-based synthesis based on Partition assignments in the source HDL code. These features provide automated block-based incremental synthesis flows and create different output netlist files for each block when set up for an Altera device.

Using incremental synthesis within your synthesis tool ensures that only those sections of a design that have been updated are resynthesized when the design is compiled, reducing synthesis run time and preserving the results for the unchanged blocks. You can change and resynthesize one section of a design without affecting other sections of the design.

1.5.3.2. Other Synthesis Tools

You can also partition your design and create different netlist files manually with the basic Synplify software (non-Pro/Premier), the basic Precision RTL software (non-Plus), or any other supported synthesis tool by creating a separate project or implementation for each partition, including the top level. Set up each higher-level project to instantiate the lower-level VQM/EDIF netlists as black boxes. Synplify, Precision, and most synthesis tools automatically treat a design block as a black box if the logic definition is missing from the project. Each tool also includes options or attributes to specify that the design block should be treated as a black box, which you can use to avoid warnings about the missing logic.

1.5.4. Assessing Partition Quality

The Quartus Prime software provides various tools to assess the quality of your assigned design partitions. You can take advantage of these tools to assess your partition quality, and use the information to improve your design or assignments as required to achieve the best results.

1.5.4.1. Partition Statistics Reports

After compilation, you can view statistics about design partitions in the Partition Merge Partition Statistics report, and on the **Statistics** tab in the **Design Partitions Properties** dialog box.

The Partition Merge Partition Statistics report lists statistics about each partition. The statistics for each partition (each row in the table) include the number of logic cells it contains, as well as the number of input and output pins it contains, and how many are registered or unconnected.

You can also view post-compilation statistics about the resource usage and port connections for a particular partition on the **Statistics** tab in the **Design Partition Properties** dialog box.

Related Information

Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.5.4.2. Partition Timing Reports

You can generate a Partition Timing Overview report and a Partition Timing Details report by clicking **Report Partitions** in the Tasks pane in the Timing Analyzer, or using the `report_partitions` Tcl command.

The Partition Timing Overview report shows the total number of failing paths for each partition and the worst-case slack for any path involving the partition.

The Partition Timing Details report shows the number of failing partition-to-partition paths and worst-case slack for partition-to-partition paths, to provide a more detailed breakdown of where the critical paths in the design are located with respect to design partitions.

1.5.4.3. Incremental Compilation Advisor

You can use the Incremental Compilation Advisor to check that your design follows Altera's recommendations for creating design partitions and floorplan location assignments.

Recommendations are split into **General Recommendations**, **Timing Recommendations**, and **Team-Based Design Recommendations** that apply to design flows in which partitions are compiled independently in separate Quartus Prime projects before being integrated into the top-level design. Each recommendation provides an explanation, describes the effect of the recommendation, and provides the action required to make a suggested change. In some cases, there is a link to the appropriate Quartus Prime settings page where you can make a suggested change to assignments or settings. For some items, if your design does not follow the recommendation, the **Check Recommendations** operation creates a table that lists any nodes or paths in your design that could be improved. The relevant timing-independent recommendations for the design are also listed in the Design Partitions window and the LogicLock Regions window.

To verify that your design follows the recommendations, go to the **Timing Independent Recommendations** page or the **Timing Dependent Recommendations** page, and then click **Check Recommendations**. For large designs, these operations can take a few minutes.

After you perform a check operation, symbols appear next to each recommendation to indicate whether the design or project setting follows the recommendations, or if some or all of the design or project settings do not follow the recommendations. Following these recommendations is not mandatory to use the incremental compilation feature. The recommendations are most important to ensure good results for timing-critical partitions.

For some items in the Advisor, if your design does not follow the recommendation, the **Check Recommendations** operation lists any parts of the design that could be improved. For example, if not all of the partition I/O ports follow the **Register All Non-Global Ports** recommendation, the advisor displays a list of unregistered ports with the partition name and the node name associated with the port.

When the advisor provides a list of nodes, you can right-click a node, and then click **Locate** to cross-probe to other Quartus Prime features, such as the RTL Viewer, Chip Planner, or the design source code in the text editor.

Note: Opening a new Timing Analyzer report resets the Incremental Compilation Advisor results, so you must rerun the Check Recommendations process.

1.6. Specifying the Level of Results Preservation for Subsequent Compilations

The netlist type of each design partition allows you to specify the level of results preservation. The netlist type determines which type of netlist or source file the Partition Merge stage uses in the next incremental compilation.

When you choose to preserve a post-fit compilation netlist, the default level of Fitter preservation is the highest degree of placement and routing preservation supported by the device family. The advanced Fitter Preservation Level setting allows you to specify the amount of information that you want to preserve from the post-fit netlist file.

1.6.1. Netlist Type for Design Partitions

Before starting a new compilation, ensure that the appropriate netlist type is set for each partition to preserve the desired level of compilation results. The table below describes the settings for the netlist type, explains the behavior of the Quartus Prime software for each setting, and provides guidance on when to use each setting.

Table 2. Partition Netlist Type Settings

Netlist Type	Quartus Prime Software Behavior for Partition During Compilation
Source File	Always compiles the partition using the associated design source file(s). (2) Use this netlist type to recompile a partition from the source code using new synthesis or Fitter settings.
Post-Synthesis	Preserves post-synthesis results for the partition and reuses the post-synthesis netlist when the following conditions are true: • A post-synthesis netlist is available from a previous synthesis. • No change that initiates an automatic resynthesis has been made to the partition since the previous synthesis. (3) Compiles the partition from the source files if resynthesis is initiated or if a post-synthesis netlist is not available. (2) Use this netlist type to preserve the synthesis results unless you make design changes, but allow the Fitter to refit the partition using any new Fitter settings.
Post-Fit	Preserves post-fit results for the partition and reuses the post-fit netlist when the following conditions are true: • A post-fit netlist is available from a previous fitting. • No change that initiates an automatic resynthesis has been made to the partition since the previous fitting. (3) When a post-fit netlist is not available, the software reuses the post-synthesis netlist if it is available, or otherwise compiles from the source files. Compiles the partition from the source files if resynthesis is initiated. (2) The Fitter Preservation Level specifies what level of information is preserved from the post-fit netlist. Assignment changes, such as Fitter optimization settings, do not cause a partition set to Post-Fit to recompile.
Empty	Uses an empty placeholder netlist for the partition. The partition's port interface information is required during Analysis and Synthesis to connect the partition correctly to other logic and partitions in the design, and peripheral nodes in the source file including pins and PLLs are preserved to help connect the empty partition to the rest of the design and preserve timing of any lower-level non-empty partitions within empty partitions. If the source file is not available, you can create a wrapper file that defines the design block and specifies the input, output, and bidirectional ports. In Verilog HDL: a module declaration, and in VHDL: an entity and architecture declaration.

continued...

(2) If you use Rapid Recompile, the Quartus Prime software might not recompile the entire partition from the source code as described in this table; it will reuse compatible results if there have been only small changes to the logic in the partition.

(3) You can turn on the **Ignore changes in source files and strictly use the specified netlist, if available** option on the **Advanced** tab in the **Design Partitions Properties** dialog box to specify whether the Compiler should ignore source file changes when deciding whether to recompile the partition.

Netlist Type	Quartus Prime Software Behavior for Partition During Compilation
	You can use this netlist type to skip the compilation of a partition that is incomplete or missing from the top-level design. You can also set an empty partition if you want to compile only some partitions in the design, such as to optimize the placement of a timing-critical block such as an IP core before incorporating other design logic, or if the compilation time is large for one partition and you want to exclude it. If the project database includes a previously generated post-synthesis or post-fit netlist for an unchanged Empty partition, you can set the netlist type from **Empty** directly to **Post-Synthesis** or **Post-Fit** and the software reuses the previous netlist information without recompiling from the source files.

Related Information

- What Changes Initiate the Automatic Resynthesis of a Partition? on page 35
- Fitter Preservation Level for Design Partitions on page 34
- Incremental Capabilities Available When A Design Has No Partitions on page 8

1.6.2. Fitter Preservation Level for Design Partitions

The default Fitter Preservation Level for partitions with a **Post-Fit** netlist type is the highest level of preservation available for the target device family and provides the most compilation time reduction.

You can change the advanced Fitter Preservation Level setting to provide more flexibility in the Fitter during placement and routing. You can set the Fitter Preservation Level on the **Advanced** tab in the **Design Partitions Properties** dialog box.

Table 3. Fitter Preservation Level Settings

Fitter Preservation Level	Quartus Prime Behavior for Partition During Compilation
Placement and Routing	Preserves the design partition's netlist atoms and their placement and routing. This setting reduces compilation times compared to Placement only, but provides less flexibility to the router to make changes if there are changes in other parts of the design. By default, the Fitter preserves the usage of high-speed programmable power tiles contained within the selected partition, for devices that support high-speed and low-power tiles. You can turn off the **Preserve high-speed tiles when preserving placement and routing** option on the **Advanced** tab in the **Design Partitions Properties** dialog box.
Placement	Preserves the netlist atoms and their placement in the design partition. Reroutes the design partition and does not preserve high-speed power tile usage.
Netlist Only	Preserves the netlist atoms of the design partition, but replaces and reroutes the design partition. A post-fit netlist with the atoms preserved can be different than the Post-Synthesis netlist because it contains Fitter optimizations; for example, Physical Synthesis changes made during a previous Fitting. You can use this setting to: • Preserve Fitter optimizations but allow the software to perform placement and routing again. • Reapply certain Fitter optimizations that would otherwise be impossible when the placement is locked down. • Resolve resource conflicts between two imported partitions.

1.6.3. Where Are the Netlist Databases Saved?

The incremental compilation database folder (**\incremental_db**) includes all the netlist information from previous compilations. To avoid unnecessary recompilations, these database files must not be altered or deleted.

If you archive or reproduce the project in another location, you can use a Quartus Prime Archive File (.**qar**). Include the incremental compilation database files to preserve post-synthesis or post-fit compilation results.

To manually create a project archive that preserves compilation results without keeping the incremental compilation database, you can keep all source and settings files, and create and save a Quartus Prime Settings File (.**qxp**) for each partition in the design that will be integrated into the top-level design.

Related Information

- Using Incremental Compilation With Quartus Prime Archive Files on page 58
- Exporting Design Partitions from Separate Quartus Prime Projects on page 37

1.6.4. Deleting Netlists

You can choose to abandon all levels of results preservation and remove all netlists that exist for a particular partition with the **Delete Netlists** command in the Design Partitions window. When you delete netlists for a partition, the partition is compiled using the associated design source file(s) in the next compilation. Resetting the netlist type for a partition to **Source** would have the same effect, though the netlists would not be permanently deleted and would be available for use in subsequent compilations. For an imported partition, the **Delete Netlists** command also optionally allows you to remove the imported **.qxp**.

1.6.5. What Changes Initiate the Automatic Resynthesis of a Partition?

A partition is synthesized from its source files if there is no post-synthesis netlist available from a previous synthesis, or if the netlist type is set to **Source File**. Additionally, certain changes to a partition initiate an automatic resynthesis of the partition when the netlist type is **Post Synthesis** or **Post-Fit**. The software resynthesizes the partition in these cases to ensure that the design description matches the post-place-and-route programming files.

The following list explains the changes that initiate a partition's automatic resynthesis when the netlist type is set to **Post-Synthesis** or **Post-Fit**:

- The device family setting has changed.
- Any dependent source design file has changed.
- The partition boundary was changed by an addition, removal, or change to the port boundaries of a partition (for example, a new partition has been defined for a lower-level instance within this partition).
- A dependent source file was compiled into a different library (so it has a different `-library` argument).
- A dependent source file was added or removed; that is, the partition depends on a different set of source files.

- The partition's root instance has a different entity binding. In VHDL, an instance may be bound to a specific entity and architecture. If the target entity or architecture changes, it triggers resynthesis.
- The partition has different parameters on its root hierarchy or on an internal AHDL hierarchy (AHDL automatically inherits parameters from its parent hierarchies). This occurs if you modified the parameters on the hierarchy directly, or if you modified them indirectly by changing the parameters in a parent design hierarchy.
- You have moved the project and compiled database between a Windows and Linux system. Due to the differences in the way new line feeds are handled between the operating systems, the internal checksum algorithm may detect a design file change in this case.

The software reuses the post-synthesis results but re-fits the design if you change the device setting within the same device family. The software reuses the post-fitting netlist if you change only the device speed grade.

Synthesis and Fitter assignments, such as optimization settings, timing assignments, or Fitter location assignments including pin assignments, do not trigger automatic recompilation in the incremental compilation flow. To recompile a partition with new assignments, change the netlist type for that partition to one of the following:

- **Source File** to recompile with all new settings
- **Post-Synthesis** to recompile using existing synthesis results but new Fitter settings
- **Post-Fit** with the **Fitter Preservation Level** set to **Placement** to rerun routing using existing placement results, but new routing settings (such as delay chain settings)

You can use the LogicLock Origin location assignment to change or fine-tune the previous Fitter results from a Post-Fit netlist.

Related Information

Changing Partition Placement with LogicLock Changes on page 57

1.6.5.1. Resynthesis Due to Source Code Changes

The Quartus Prime software uses an internal checksum algorithm to determine whether the contents of a source file have changed. Source files are the design description files used to create the design, and include Memory Initialization Files (.**mif**) as well as .**qxp** from exported partitions. When design files in a partition have dependencies on other files, changing one file may initiate an automatic recompilation of another file. The Partition Dependent Files table in the Analysis and Synthesis report lists the design files that contribute to each design partition. You can use this table to determine which partitions are recompiled when a specific file is changed.

For example, if a design has file **A.v** that contains entity **A**, **B.v** that contains entity **B**, and **C.v** that contains entity **C**, then the Partition Dependent Files table for the partition containing entity **A** lists file **A.v**, the table for the partition containing entity **B** lists file **B.v**, and the table for the partition containing entity **C** lists file **C.v**. Any dependencies are transitive, so if file **A.v** depends on **B.v**, and **B.v** depends on **C.v**, the entities in file **A.v** depend on files **B.v** and **C.v**. In this case, files **B.v** and **C.v** are listed in the report table as dependent files for the partition containing entity **A**.

Note: If you use Rapid Recompile, the Quartus Prime software might not recompile the entire partition from the source code as described in this section; it will reuse compatible results if there have been only small changes to the logic in the partition.

If you define module parameters in a higher-level module, the Quartus Prime software checks the parameter values when determining which partitions require resynthesis. If you change a parameter in a higher-level module that affects a lower-level module, the lower-level module is resynthesized. Parameter dependencies are tracked separately from source file dependencies; therefore, parameter definitions are not listed in the **Partition Dependent Files** list.

If a design contains common files, such as an **includes.v** file that is referenced in each entity by the command `include includes.v`, all partitions are dependent on this file. A change to **includes.v** causes the entire design to be recompiled. The VHDL statement `use work.all` also typically results in unnecessary recompilations, because it makes all entities in the work library visible in the current entity, which results in the current entity being dependent on all other entities in the design.

To avoid this type of problem, ensure that files common to all entities, such as a common include file, contain only the set of information that is truly common to all entities. Remove `use work.all` statements in your VHDL file or replace them by including only the specific design units needed for each entity.

Related Information

Incremental Capabilities Available When A Design Has No Partitions on page 8

1.6.5.2. Forcing Use of the Compilation Netlist When a Partition has Changed

Forcing the use of a post-compilation netlist when the contents of a source file has changed is recommended only for advanced users who understand when a partition must be recompiled. You might use this assignment, for example, if you are making source code changes but do not want to recompile the partition until you finish debugging a different partition, or if you are adding simple comments to the source file but you know the design logic itself is not being changed and you want to keep the previous compilation results.

To force the Fitter to use a previously generated netlist even when there are changes to the source files, right-click the partition in the Design Partitions window and then click **Design Partition Properties**. On the **Advanced** tab, turn on the **Ignore changes in source files and strictly use the specified netlist, if available** option.

Turning on this option can result in the generation of a functionally incorrect netlist when source design files change, because source file updates will not be recompiled. Use caution when setting this option.

1.7. Exporting Design Partitions from Separate Quartus Prime Projects

Partitions that are developed by other designers or team members in the same company or third-party IP providers can be exported as design partitions to a Quartus Prime Exported Partition File (.**qxp**), and then integrated into a top-level design. A .**qxp** is a binary file that contains compilation results describing the exported design

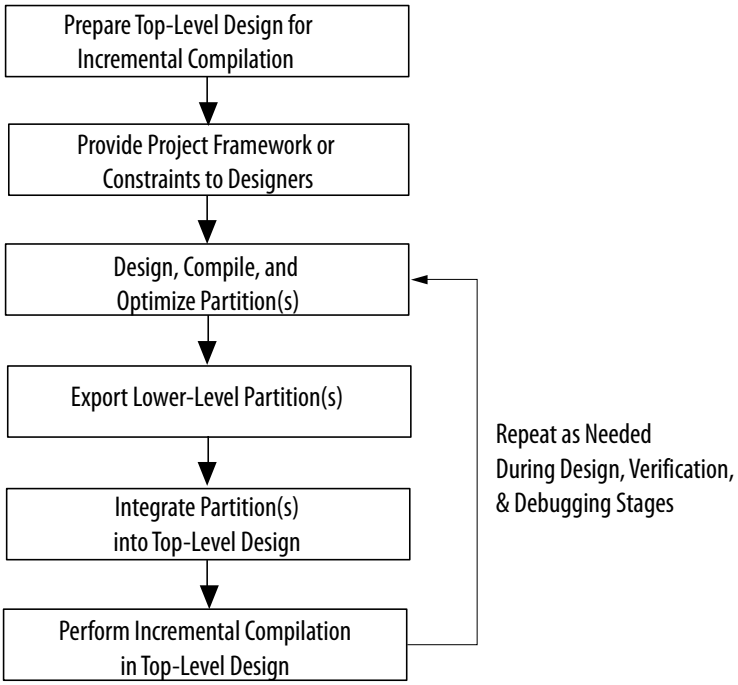
partition and includes a post-synthesis netlist, a post-fit netlist, or both, and a set of assignments, sometimes including LogicLock placement constraints. The .**qxp** does not contain the source design files from the original Quartus Prime project.

To enable team-based development and third-party IP delivery, you can design and optimize partitions in separate copies of the top-level Quartus Prime project framework, or even in isolation. If the designers have access to the top-level project framework through a source control system, they can access project files as read-only and develop their partition within the source control system. If designers do not have access to a source control system, the project lead can provide the designer with a copy of the top-level project framework to use as they develop their partitions. The project lead also has the option to generate design partition scripts to manage resource and timing budgets in the top-level design when partitions are developed outside the top-level project framework.

The exported compilation results of completed partitions are given to the project lead, preferably using a source control system, who is then responsible for integrating them into the top-level design to obtain a fully functional design. This type of design flow is required only if partition designers want to optimize their placement and routing independently, and pass their design to the project lead to reuse placement and routing results. Otherwise, a project lead can integrate source HDL from several designers in a single Quartus Prime project, and use the standard incremental compilation flow described previously.

The figure below illustrates the team-based incremental compilation design flow using a methodology in which partitions are compiled in separate Quartus Prime projects before being integrated into the top-level design. This flow can be used when partitions are developed by other designers or IP providers.

Figure 7. Team-Based Incremental Compilation Design Flow



Note: You cannot export or import partitions that have been merged.

Related Information

- Deciding Which Design Blocks Should Be Design Partitions on page 26
- Incremental Compilation Restrictions on page 58

1.7.1. Preparing the Top-Level Design

To prepare your design to incorporate exported partitions, first create the top-level project framework of the design to define the hierarchy for the subdesigns that will be implemented by other team members, designers, or IP providers.

In the top-level design, create project-wide settings, for example, device selection, global assignments for clocks and device I/O ports, and any global signal constraints to specify which signals can use global routing resources.

Next, create the appropriate design partition assignments and set the netlist type for each design partition that will be developed in a separate Quartus Prime project to **Empty**. It may be necessary to constrain the location of partitions with LogicLock region assignments if they are timing-critical and are expected to change in future compilations, or if the designer or IP provider wants to place and route their design partition independently, to avoid location conflicts.

Finally, provide the top-level project framework to the partition designers, preferably through a source control system.

Related Information

Creating a Design Floorplan With LogicLock Regions on page 56

1.7.1.1. Empty Partitions

You can use a design flow in which some partitions are set to an **Empty** netlist type to develop pieces of the design separately, and then integrate them into the top-level design at a later time. In a team-based design environment, you can set the netlist type to **Empty** for partitions in your design that will be developed by other designers or IP providers. The **Empty** setting directs the Compiler to skip the compilation of a partition and use an empty placeholder netlist for the partition.

When a netlist type is set to **Empty**, peripheral nodes including pins and PLLs are preserved and all other logic is removed. The peripheral nodes including pins help connect the empty partition to the design, and the PLLs help preserve timing of non-empty partitions within empty partitions.

When you set a design partition to **Empty**, a design file is required during Analysis and Synthesis to specify the port interface information so that it can connect the partition correctly to other logic and partitions in the design. If a partition is exported from another project, the .**qxp** contains this information. If there is no .**qxp** or design file to represent the design entity, you must create a wrapper file that defines the design block and specifies the input, output, and bidirectional ports. For example, in Verilog HDL, you should include a module declaration, and in VHDL, you should include an entity and architecture declaration.

1.7.2. Project Management— Making the Top-Level Design Available to Other Designers

In team-based incremental compilation flows, whenever possible, all designers or IP providers should work within the same top-level project framework. Using the same project framework among team members ensures that designers have the settings and constraints needed for their partition, and makes timing closure easier when integrating the partitions into the top-level design. If other designers do not have access to the top-level project framework, the Quartus Prime software provides tools for passing project information to partition designers.

1.7.2.1. Distributing the Top-Level Quartus Prime Project

There are several methods that the project lead can use to distribute the “skeleton” or top-level project framework to other partition designers or IP providers.

- If partition designers have access to the top-level project framework, the project will already include all the settings and constraints needed for the design. This framework should include PLLs and other interface logic if this information is important to optimize partitions.
  - If designers are part of the same design environment, they can check out the required project files from the same source control system. This is the recommended way to share a set of project files.
  - Otherwise, the project lead can provide a copy of the top-level project framework so that each design develops their partition within the same project framework.
- If a partition designer does not have access to the top-level project framework, the project lead can give the partition designer a Tcl script or other documentation to create the separate Quartus Prime project and all the assignments from the top-level design.

If the partition designers provide the project lead with a post-synthesis **.qxp** and fitting is performed in the top-level design, integrating the design partitions should be quite easy. If you plan to develop a partition in a separate Quartus Prime project and integrate the optimized post-fitting results into the top-level design, use the following guidelines to improve the integration process:

- Ensure that a LogicLock region constrains the partition placement and uses only the resources allocated by the project lead.
- Ensure that you know which clocks should be allocated to global routing resources so that there are no resource conflicts in the top-level design.
  - Set the Global Signal assignment to **On** for the high fan-out signals that should be routed on global routing lines.
  - To avoid other signals being placed on global routing lines, turn off **Auto Global Clock and Auto Global Register Controls** under **More Settings** on the Fitter page in the **Settings** dialog box. Alternatively, you can set the Global Signal assignment to **Off** for signals that should not be placed on global routing lines.

    Placement for LABs depends on whether the inputs to the logic cells within the LAB use a global clock. You may encounter problems if signals do not use global lines in the partition, but use global routing in the top-level design.
- Use the Virtual Pin assignment to indicate pins of a partition that do not drive pins in the top-level design. This is critical when a partition has more output ports than the number of pins available in the target device. Using virtual pins also helps optimize cross-partition paths for a complete design by enabling you to provide more information about the partition ports, such as location and timing assignments.
- When partitions are compiled independently without any information about each other, you might need to provide more information about the timing paths that may be affected by other partitions in the top-level design. You can apply location assignments for each pin to indicate the port location after incorporation in the top-level design. You can also apply timing assignments to the I/O ports of the partition to perform timing budgeting.

Related Information

Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.7.2.2. Generating Design Partition Scripts

If IP providers or designers on a team want to optimize their design blocks independently and do not have access to a shared project framework, the project lead must perform some or all of the following tasks to ensure successful integration of the design blocks:

- Determine which assignments should be propagated from the top-level design to the partitions. This requires detailed knowledge of which assignments are required to set up low-level designs.
- Communicate the top-level assignments to the partitions. This requires detailed knowledge of Tcl or other scripting languages to efficiently communicate project constraints.
- Determine appropriate timing and location assignments that help overcome the limitations of team-based design. This requires examination of the logic in the partitions to determine appropriate timing constraints.
- Perform final timing closure and resource conflict avoidance in the top-level design. Because the partitions have no information about each other, meeting constraints at the lower levels does not guarantee they are met when integrated at the top-level. It then becomes the project lead's responsibility to resolve the issues, even though information about the partition implementation may not be available.

Design partition scripts automate the process of transferring the top-level project framework to partition designers in a flow where each design block is developed in separate Quartus Prime projects before being integrated into the top-level design. If the project lead cannot provide each designer with a copy of the top-level project framework, the Quartus Prime software provides an interface for managing resources and timing budgets in the top-level design. Design partition scripts make it easier for partition designers to implement the instructions from the project lead, and avoid conflicts between projects when integrating the partitions into the top-level design. This flow also helps to reduce the need to further optimize the designs after integration.

You can use options in the **Generate Design Partition Scripts** dialog box to choose which types of assignments you want to pass down and create in the partitions being developed in separate Quartus Prime projects.

Related Information

Enabling Designers on a Team to Optimize Independently on page 51

1.7.3. Exporting Partitions

When partition designers achieve the design requirements in their separate Quartus Prime projects, each designer can export their design as a partition so it can be integrated into the top-level design by the project lead. The **Export Design Partition** dialog box, available from the Project menu, allows designers to export a design partition to a Quartus Prime Exported Partition File (.**qxp**) with a post-synthesis netlist, a post-fit netlist, or both. The project lead then adds the .**qxp** to the top-level design to integrate the partition.

A designer developing a timing-critical partition or who wants to optimize their partition on their own would opt to export their completed partition with a post-fit netlist, allowing for the partition to more reliably meet timing requirements after integration. In this case, you must ensure that resources are allocated appropriately

to avoid conflicts. If the placement and routing optimization can be performed in the top-level design, exporting a post-synthesis netlist allows the most flexibility in the top-level design and avoids potential placement or routing conflicts with other partitions.

When designing the partition logic to be exported into another project, you can add logic around the design block to be exported as a design partition. You can instantiate additional design components for the Quartus Prime project so that it matches the top-level design environment, especially in cases where you do not have access to the full top-level design project. For example, you can include a top-level PLL in the project, outside of the partition to be exported, so that you can optimize the design with information about the frequency multipliers, phase shifts, compensation delays, and any other PLL parameters. The software then captures timing and resource requirements more accurately while ensuring that the timing analysis in the partition is complete and accurate. You can export the partition for the top-level design without any auxiliary components that are instantiated outside the partition being exported.

If your design team uses makefiles and design partition scripts, the project lead can use the **make** command with the **master_makefile** command created by the scripts to export the partitions and create .**qxp** files. When a partition has been compiled and is ready to be integrated into the top-level design, you can export the partition with option on the **Export Design Partition** dialog box, available from the Project menu.

1.7.4. Viewing the Contents of a Quartus Prime Exported Partition File (.qxp)

The QXP report allows you to view a summary of the contents in a .**qxp** when you open the file in the Quartus Prime software. The .**qxp** is a binary file that contains compilation results so the file cannot be read in a text editor. The QXP report opens in the main Quartus Prime window and contains summary information including a list of the I/O ports, resource usage summary, and a list of the assignments used for the exported partition.

1.7.5. Integrating Partitions into the Top-Level Design

To integrate a partition developed in a separate Quartus Prime project into the top-level design, you can simply add the .**qxp** as a source file in your top-level design (just like a Verilog or VHDL source file). You can also use the **Import Design Partition** dialog box to import the partition.

The .**qxp** contains the design block exported from the partition and has the same name as the partition. When you instantiate the design block into a top-level design and include the .**qxp** as a source file, the software adds the exported netlist to the database for the top-level design. The .**qxp** port names are case sensitive if the original HDL of the partition was case sensitive.

When you use a .**qxp** as a source file in this way, you can choose whether you want the .**qxp** to be a partition in the top-level design. If you do not designate the .**qxp** instance as a partition, the software reuses just the post-synthesis compilation results from the .**qxp**, removes unconnected ports and unused logic just like a regular source file, and then performs placement and routing.

If you assigned the .**qxp** instance as a partition, you can set the netlist type in the Design Partitions Window to choose the level of results to preserve from the .**qxp**. To preserve the placement and routing results from the exported partition, set the netlist type to **Post-Fit** for the .**qxp** partition in the top-level design. If you assign the instance as a design partition, the partition boundary is preserved.

Related Information

Impact of Design Partitions on Design Optimization on page 29

1.7.5.1. Integrating Assignments from the .qxp

The Quartus Prime software filters assignments from .**qxp** files to include appropriate assignments in the top-level design. The assignments in the .**qxp** are treated like assignments made in an HDL source file, and are not listed in the Quartus Prime Settings File (.**qsf**) for the top-level design. Most assignments from the .**qxp** can be overridden by assignments in the top-level design.

1.7.5.1.1. Design Partition Assignments Within the Exported Partition

Design partition assignments defined within a separate Quartus Prime project are not added to the top-level design. All logic under the exported partition in the project hierarchy is treated as single instance in the .**qxp**.

1.7.5.1.2. Synopsys Design Constraint Files for the Quartus Prime Timing Analyzer

Timing assignments made for the Quartus Prime Timing Analyzer in a Synopsys Design Constraint File (.**sdc**) in the lower-level partition project are not added to the top-level design. Ensure that the top-level design includes all of the timing requirements for the entire project.

Related Information

Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.7.5.1.3. Global Assignments

The project lead should make all global project-wide assignments in the top-level design. Global assignments from the exported partition's project are not added to the top-level design. When it is possible for a particular constraint, the global assignment is converted to an instance-specific assignment for the exported design partition.

1.7.5.1.4. LogicLock Region Assignments

The project lead typically creates LogicLock region assignments in the top-level design for any lower-level partition designs where designer or IP providers plan to export post-fit information to be used in the top-level design, to help avoid placement conflicts between partitions. When you use the .**qxp** as a source file, LogicLock constraints from the exported partition are applied in the top-level design, but will not appear in your .**qsf** file or LogicLock Regions window for you to view or edit. The LogicLock region itself is not required to constrain the partition placement in the top-level design if the netlist type is set to **Post-Fit**, because the netlist contains all the placement information.

1.7.5.2. Integrating Encrypted IP Cores from .qxp Files

Proper license information is required to compile encrypted IP cores. If an IP core is exported as a **.qxp** from another Quartus Prime project, the top-level designer instantiating the **.qxp** must have the correct license. The software requires a full license to generate an unrestricted programming file. If you do not have a license, but the IP in the **.qxp** was compiled with OpenCore Plus hardware evaluation support, you can generate an evaluation programming file without a license. If the IP supports OpenCore simulation only, you can fully compile the design and generate a simulation netlist, but you cannot create programming files unless you have a full license.

1.7.5.3. Advanced Importing Options

You can use advanced options in the **Import Design Partition** dialog box to integrate a partition developed in a separate Quartus Prime project into the top-level design. The import process adds more control than using the **.qxp** as a source file, and is useful only in the following circumstances:

- **If you want LogicLock regions in your top-level design (.qsf)**—If you have regions in your partitions that are not also in the top-level design, the regions will be added to your **.qsf** during the import process.
- **If you want different settings or placement for different instantiations of the same entity**—You can control the setting import process with the advanced import options, and specify different settings for different instances of the same **.qxp** design block.

When you use the **Import Design Partition** dialog box to integrate a partition into the top-level design, the import process sets the partition's netlist type to **Imported** in the Design Partitions window.

After you compile the entire design, if you make changes to the place-and-route results (such as movement of an imported LogicLock region), use the **Post-Fit** netlist type on subsequent compilations. To discard an imported netlist and recompile from source code, you can compile the partition with the netlist type set to **Source File** and be sure to include the relevant source code in the top-level design. The import process sets the partition's Fitter Preservation Level to the setting with the highest degree of preservation supported by the imported netlist. For example, if a post-fit netlist is imported with placement information, the Fitter Preservation Level is set to **Placement**, but you can change it to the **Netlist Only** value.

When you import a partition from a **.qxp**, the **.qxp** itself is not part of the top-level design because the netlists from the file have been imported into the project database. Therefore if a new version of a **.qxp** is exported, the top-level designer must perform another import of the **.qxp**.

When you import a partition into a top-level design with the **Import Design Partition** dialog box, the software imports relevant assignments from the partition into the top-level design. If required, you can change the way some assignments are imported, as described in the following subsections.

Related Information

- Netlist Type for Design Partitions on page 33
- Fitter Preservation Level for Design Partitions on page 34

1.7.5.3.1. Importing LogicLock Assignments

LogicLock regions are set to a fixed size when imported. If you instantiate multiple instances of a subdesign in the top-level design, the imported LogicLock regions are set to a Floating location. Otherwise, they are set to a Fixed location. You can change the location of LogicLock regions after they are imported, or change them to a Floating location to allow the software to place each region but keep the relative locations of nodes within the region wherever possible. To preserve changes made to a partition after compilation, use the **Post-Fit** netlist type.

The LogicLock Member State assignment is set to **Locked** to signify that it is a preserved region.

LogicLock back-annotation and node location data is not imported because the .**qxp** contains all of the relevant placement information. Altera strongly recommends that you do not add to or delete members from an imported LogicLock region.

Related Information

Changing Partition Placement with LogicLock Changes on page 57

1.7.5.3.2. Advanced Import Settings

The **Advanced Import Settings** dialog box allows you to disable assignment import and specify additional options that control how assignments and regions are integrated when importing a partition into a top-level design, including how to resolve assignment conflicts.

1.8. Team-Based Design Optimization and Third-Party IP Delivery Scenarios

1.8.1. Using an Exported Partition to Send to a Design Without Including Source Files

Scenario background: A designer wants to produce a design block and needs to send out their design, but to preserve their IP, they prefer to send a synthesized netlist instead of providing the HDL source code to the recipient. You can use this flow to implement a black box.

Use this flow to package a full design as a single source file to send to an end customer or another design location.

As the sender in this scenario perform the following steps to export a design block:

1. Provide the device family name to the recipient. If you send placement information with the synthesized netlist, also provide the exact device selection so they can set up their project to match.
2. Create a black box wrapper file that defines the port interface for the design block and provide it to the recipient for instantiating the block as an empty partition in the top-level design.
3. Create a Quartus Prime project for the design block, and complete the design.
4. Export the level of hierarchy into a single .**qxp**. Following a successful compilation of the project, you can generate a .**qxp** from the GUI, the command-line, or with Tcl commands, as described in the following:

- If you are using the Quartus Prime GUI, use the **Export Design Partition** dialog box.
- If you are using command-line executables, run **quartus_cdb** with the `--incremental_compilation_export` option.
- If you are using Tcl commands, use the following command: `execute_flow -incremental_compilation_export`.

5. Select the option to include just the **Post-synthesis netlist** if you do not have to send placement information. If the recipient wants to reproduce your exact Fitter results, you can select the **Post-fitting netlist** option, and optionally enable **Export routing**.
6. If a partition contains sub-partitions, then the sub-partitions are automatically flattened and merged into the partition netlist before exporting. You can change this behavior and preserve the sub-partition hierarchy by turning off the **Flatten sub-partitions** option on the **Export Design Partition** dialog box. Optionally, you can use the `-dont_flatten` sub-option for the `export_partition` Tcl command.
7. Provide the .**qxp** to the recipient. Note that you do not have to send any of your design source code.

As the recipient in this example, first create a Quartus Prime project for your top-level design and ensure that your project targets the same device (or at least the same device family if the .**qxp** does not include placement information), as specified by the IP designer sending the design block. Instantiate the design block using the port information provided, and then incorporate the design block into a top-level design.

Add the .**qxp** from the IP designer as a source file in your Quartus Prime project to replace any empty wrapper file. If you want to use just the post-synthesis information, you can choose whether you want the file to be a partition in the top-level design. To use the post-fit information from the .**qxp**, assign the instance as a design partition and set the netlist type to **Post-Fit**.

Related Information

- Creating Design Partitions on page 14
- Netlist Type for Design Partitions on page 33

1.8.2. Creating Precompiled Design Blocks (or Hard-Wired Macros) for Reuse

Scenario background: An IP provider wants to produce and sell an IP core for a component to be used in higher-level systems. The IP provider wants to optimize the placement of their block for maximum performance in a specific Altera device and then deliver the placement information to their end customer. To preserve their IP, they also prefer to send a compiled netlist instead of providing the HDL source code to their customer.

Use this design flow to create a precompiled IP block (sometimes known as a hard-wired macro) that can be instantiated in a top-level design. This flow provides the ability to export a design block with post-synthesis or placement (and, optionally, routing) information and to import any number of copies of this pre-compiled block into another design.

The customer first specifies which Altera device is being used for this project and provides the design specifications.

As the IP provider in this example, perform the following steps to export a preplaced IP core (or hard macro):

1. Create a black box wrapper file that defines the port interface for the IP core and provide the file to the customer to instantiate as an empty partition in the top-level design.
2. Create a Quartus Prime project for the IP core.
3. Create a LogicLock region for the design hierarchy to be exported.

   Using a LogicLock region for the IP core allows the customer to create an empty placeholder region to reserve space for the IP in the design floorplan and ensures that there are no conflicts with the top-level design logic. Reserved space also helps ensure the IP core does not affect the timing performance of other logic in the top-level design. Additionally, with a LogicLock region, you can preserve placement either absolutely or relative to the origin of the associated region. This is important when a .**qxp** is imported for multiple partition hierarchies in the same project, because in this case, the location of at least one instance in the top-level design does not match the location used by the IP provider.
4. If required, add any logic (such as PLLs or other logic defined in the customer's top-level design) around the design hierarchy to be exported. If you do so, create a design partition for the design hierarchy that will exported as an IP core.
5. Optimize the design and close timing to meet the design specifications.
6. Export the level of hierarchy for the IP core into a single .**qxp**.
7. Provide the .**qxp** to the customer. Note that you do not have to send any of your design source code to the customer; the design netlist and placement and routing information is contained within the .**qxp**.

Related Information

- Creating Design Partitions on page 64
- Netlist Type for Design Partitions on page 33
- Changing Partition Placement with LogicLock Changes on page 57

Incorporate IP Core

As the customer in this example, incorporate the IP core in your design by performing the following steps:

1. Create a Quartus Prime project for the top-level design that targets the same device and instantiate a copy or multiple copies of the IP core. Use a black box wrapper file to define the port interface of the IP core.
2. Perform Analysis and Elaboration to identify the design hierarchy.
3. Create a design partition for each instance of the IP core with the netlist type set to **Empty**.
4. You can now continue work on your part of the design and accept the IP core from the IP provider when it is ready.
5. Include the .**qxp** from the IP provider in your project to replace the empty wrapper-file for the IP instance. Or, if you are importing multiple copies of the design block and want to import relative placement, follow these additional steps:

a. Use the **Import** command to select each appropriate partition hierarchy. You can import a .**qxp** from the GUI, the command-line, or with Tcl commands:
   - If you are using the Quartus Prime GUI, use the **Import Design Partition** command.
   - If you are using command-line executables, run **quartus_cdb** with the `incremental_compilation_import` option.
   - If you are using Tcl commands, use the following command:`execute_flow -incremental_compilation_import`.

b. When you have multiple instances of the IP block, you can set the imported LogicLock regions to floating, or move them to a new location, and the software preserves the relative placement for each of the imported modules (relative to the origin of the LogicLock region). Routing information is preserved whenever possible.

   Note: The Fitter ignores relative placement assignments if the LogicLock region's location in the top-level design is not compatible with the locations exported in the .**qxp**.

6. You can control the level of results preservation with the **Netlist Type** setting.

   If the IP provider did not define a LogicLock region in the exported partition, the software preserves absolute placement locations and this leads to placement conflicts if the partition is imported for more than one instance

1.8.3. Designing in a Team-Based Environment

Scenario background: A project includes several lower-level design blocks that are developed separately by different designers and instantiated exactly once in the top-level design.

This scenario describes how to use incremental compilation in a team-based design environment where each designer has access to the top-level project framework, but wants to optimize their design in a separate Quartus Prime project before integrating their design block into the top-level design.

As the project lead in this scenario, perform the following steps to prepare the top-level design:

1. Create a new Quartus Prime project to ultimately contain the full implementation of the entire design and include a "skeleton" or framework of the design that defines the hierarchy for the subdesigns implemented by separate designers. The

top-level design implements the top-level entity in the design and instantiates wrapper files that represent each subdesign by defining only the port interfaces, but not the implementation.

2. Make project-wide settings. Select the device, make global assignments such as device I/O ports, define the top-level timing constraints, and make any global signal allocation constraints to specify which signals can use global routing resources.
3. Make design partition assignments for each subdesign and set the netlist type for each design partition to be imported to **Empty** in the Design Partitions window.
4. Create LogicLock regions to create a design floorplan for each of the partitions that will be developed separately. This floorplan should consider the connectivity between partitions and estimates of the size of each partition based on any initial implementation numbers and knowledge of the design specifications.
5. Provide the top-level project framework to partition designers using one of the following procedures:
   - Allow access to the full project for all designers through a source control system. Each designer can check out the projects files as read-only and work on their blocks independently. This design flow provides each designer with the most information about the full design, which helps avoid resource conflicts and makes design integration easy.
   - Provide a copy of the top-level Quartus Prime project framework for each designer. You can use the **Copy Project** command on the Project menu or create a project archive.

Exporting Your Partition

As the designer of a lower-level design block in this scenario, design and optimize your partition in your copy of the top-level design, and then follow these steps when you have achieved the desired compilation results:

1. On the Project menu, click **Export Design Partition**.
2. In the **Export Design Partition** dialog box, choose the netlist(s) to export. You can export a Post-synthesis netlist if placement or performance preservation is not required, to provide the most flexibility for the Fitter in the top-level design. Select Post-fit netlist to preserve the placement and performance of the lower-level design block, and turn on **Export routing** to include the routing information, if required. One **.qxp** can include both post-synthesis and post-fitting netlists.
3. Provide the **.qxp** to the project lead.

Integrating Your Partitions

Finally, as the project lead in this scenario, perform these steps to integrate the **.qxp** files received from designers of each partition:

1. Add the **.qxp** as a source file in the Quartus Prime project, to replace any empty wrapper file for the previously **Empty** partition.
2. Change the netlist type for the partition from **Empty** to the required level of results preservation.

1.8.4. Enabling Designers on a Team to Optimize Independently

Scenario background: A project consists of several lower-level design blocks that are developed separately by different designers who do not have access to a shared top-level project framework. This scenario is similar to creating precompiled design blocks for reuse, but assumes that there are several design blocks being developed independently (instead of just one IP block), and the project lead can provide some information about the design to the individual designers. If the designers have shared access to the top-level design, use the instructions for designing in a team-based environment.

This scenario assumes that there are several design blocks being developed independently (instead of just one IP block), and the project lead can provide some information about the design to the individual designers.

This scenario describes how to use incremental compilation in a team-based design environment where designers or IP developers want to fully optimize the placement and routing of their design independently in a separate Quartus Prime project before sending the design to the project lead. This design flow requires more planning and careful resource allocation because design blocks are developed independently.

Related Information

- Creating Precompiled Design Blocks (or Hard-Wired Macros) for Reuse on page 47
- Designing in a Team-Based Environment on page 49

1.8.4.1. Preparing Your Top-level Design

As the project lead in this scenario, perform the following steps to prepare the top-level design:

1. Create a new Quartus Prime project to ultimately contain the full implementation of the entire design and include a "skeleton" or framework of the design that defines the hierarchy for the subdesigns implemented by separate designers. The top-level design implements the top-level entity in the design and instantiates wrapper files that represent each subdesign by defining only the port interfaces but not the implementation.
2. Make project-wide settings. Select the device, make global assignments such as device I/O ports, define the top-level timing constraints, and make any global signal constraints to specify which signals can use global routing resources.
3. Make design partition assignments for each subdesign and set the netlist type for each design partition to be imported to **Empty** in the Design Partitions window.
4. Create LogicLock regions. This floorplan should consider the connectivity between partitions and estimates of the size of each partition based on any initial implementation numbers and knowledge of the design specifications.
5. Provide the constraints from the top-level design to partition designers using one of the following procedures.

- Use design partition scripts to pass constraints and generate separate Quartus Prime projects. On the Project menu, use the **Generate Design Partition Scripts** command, or run the script generator from a Tcl or command prompt. Make changes to the default script options as required for your project. Altera recommends that you pass all the default constraints, including LogicLock regions, for all partitions and virtual pin location assignments. If partitions have not already been created by the other designers, use the partition script to set up the projects so that you can easily take advantage of makefiles. Provide each partition designer with the Tcl file to create their project with the appropriate constraints. If you are using makefiles, provide the makefile for each partition.
- Use documentation or manually-created scripts to pass all constraints and assignments to each partition designer.

1.8.4.2. Exporting Your Design

As the designer of a lower-level design block in this scenario, perform the appropriate set of steps to successfully export your design, whether the design team is using makefiles or exporting and importing the design manually.

If you are using makefiles with the design partition scripts, perform the following steps:

1. Use the **make** command and the makefile provided by the project lead to create a Quartus Prime project with all design constraints, and compile the project.
2. The information about which source file should be associated with which partition is not available to the software automatically, so you must specify this information in the makefile. You must specify the dependencies before the software rebuilds the project after the initial call to the makefile.
3. When you have achieved the desired compilation results and the design is ready to be imported into the top-level design, the project lead can use the `master_makefile` command to export this partition and create a .**qxp**, and then import it into the top-level design.

Exporting Without Makefiles

If you are not using makefiles, perform the following steps:

1. If you are using design partition scripts, source the Tcl script provided by the Project Lead to create a project with the required settings:
   - To source the Tcl script in the Quartus Prime software, on the Tools menu, click **Utility Windows** to open the Tcl console. Navigate to the script's directory, and type the following command: `source <filename>`.
   - To source the Tcl script at the system command prompt, type the following command: `quartus_cdb -t <filename>.tcl`
2. If you are not using design partition scripts, create a new Quartus Prime project for the subdesign, and then apply the following settings and constraints to ensure successful integration:

- Make LogicLock region assignments and global assignments (including clock settings) as specified by the project lead.
- Make Virtual Pin assignments for ports which represent connections to core logic instead of external device pins in the top-level design.
- Make floorplan location assignments to the Virtual Pins so they are placed in their corresponding regions as determined by the top-level design. This provides the Fitter with more information about the timing constraints between modules. Alternatively, you can apply timing I/O constraints to the paths that connect to virtual pins.

3. Proceed to compile and optimize the design as needed.
4. When you have achieved the desired compilation results, on the Project menu, click **Export Design Partition**.
5. In the **Export Design Partition** dialog box, choose the netlist(s) to export. You can export a Post-synthesis netlist instead if placement or performance preservation is not required, to provide the most flexibility for the Fitter in the top-level design. Select **Post-fit** to preserve the placement and performance of the lower-level design block, and turn on Export routing to include the routing information, if required. One .**qxp** can include both post-synthesis and post-fitting netlists.
6. Provide the .**qxp** to the project lead.

1.8.4.3. Importing Your Design

Finally, as the project lead in this scenario, perform the appropriate set of steps to import the .**qxp** files received from designers of each partition.

If you are using makefiles with the design partition scripts, perform the following steps:

1. Use the `master_makefile` command to export each partition and create .**qxp** files, and then import them into the top-level design.
2. The software does not have all the information about which source files should be associated with which partition, so you must specify this information in the makefile. The software cannot rebuild the project if source files change unless you specify the dependencies.

Importing Without Makefiles

If you are not using makefiles, perform the following steps:

1. Add the .**qxp** as a source file in the Quartus Prime project, to replace any empty wrapper file for the previously Empty partition.
2. Change the netlist type for the partition from Empty to the required level of results preservation.

1.8.4.4. Resolving Assignment Conflicts During Integration

When integrating lower-level design blocks, the project lead may notice some assignment conflicts. This can occur, for example, if the lower-level design block designers changed their LogicLock regions to account for additional logic or placement constraints, or if the designers applied I/O port timing constraints that differ from constraints added to the top-level design by the project lead. The project lead can address these conflicts by explicitly importing the partitions into the top-level design,

and using options in the **Advanced Import Settings** dialog box. After the project lead obtains the **.qxp** for each lower-level design block from the other designers, use the **Import Design Partition** command on the Project menu and specify the partition in the top-level design that is represented by the lower-level design block **.qxp**. Repeat this import process for each partition in the design. After you have imported each partition once, you can select all the design partitions and use the **Reimport using latest import files at previous locations** option to import all the files from their previous locations at one time. To address assignment conflicts, the project lead can take one or both of the following actions:

- Allow new assignments to be imported
- Allow existing assignments to be replaced or updated

When LogicLock region assignment conflicts occur, the project lead may take one of the following actions:

- Allow the imported region to replace the existing region
- Allow the imported region to update the existing region
- Skip assignment import for regions with conflicts

If the placement of different lower-level design blocks conflict, the project lead can also set the set the partition's **Fitter Preservation Level** to **Netlist Only**, which allows the software to re-perform placement and routing with the imported netlist.

1.8.4.5. Importing a Partition to be Instantiated Multiple Times

In this variation of the design scenario, one of the lower-level design blocks is instantiated more than once in the top-level design. The designer of the lower-level design block may want to compile and optimize the entity once under a partition, and then import the results as multiple partitions in the top-level design.

If you import multiple instances of a lower-level design block into the top-level design, the imported LogicLock regions are automatically set to Floating status.

If you resolve conflicts manually, you can use the import options and manual LogicLock assignments to specify the placement of each instance in the top-level design.

1.8.5. Performing Design Iterations With Lower-Level Partitions

Scenario background: A project consists of several lower-level subdesigns that have been exported from separate Quartus Prime projects and imported into the top-level design. In this example, integration at the top level has failed because the timing requirements are not met. The timing requirements might have been met in each individual lower-level project, but critical inter-partition paths in the top-level design are causing timing requirements to fail.

After trying various optimizations in the top-level design, the project lead determines that the design cannot meet the timing requirements given the current partition placements that were imported. The project lead decides to pass additional information to the lower-level partitions to improve the placement.

Use this flow if you re-optimize partitions exported from separate Quartus Prime projects by incorporating additional constraints from the integrated top-level design.

1.8.5.1. Providing the Complete Top-Level Project Framework

The best way to provide top-level design information to designers of lower-level partitions is to provide the complete top-level project framework using the following steps:

1. For all partitions other than the one(s) being optimized by a designer(s) in a separate Quartus Prime project(s), set the netlist type to **Post-Fit**.
2. Make the top-level design directory available in a shared source control system, if possible. Otherwise, copy the entire top-level design project directory (including database files), or create a project archive including the post-compilation database.
3. Provide each partition designer with a checked-out version or copy of the top-level design.
4. The partition designers recompile their designs within the new project framework that includes the rest of the design's placement and routing information as well top-level resource allocations and assignments, and optimize as needed.
5. When the results are satisfactory and the timing requirements are met, export the updated partition as a .**qxp**.

1.8.5.2. Providing Information About the Top-Level Framework

If this design flow is not possible, you can generate partition-specific scripts for individual designs to provide information about the top-level project framework with these steps:

1. In the top-level design, on the Project menu, click **Generate Design Partition Scripts**, or launch the script generator from Tcl or the command line.
2. If lower-level projects have already been created for each partition, you can turn off the **Create lower-level project if one does not exist** option.
3. Make additional changes to the default script options, as necessary. Altera recommends that you pass all the default constraints, including LogicLock regions, for all partitions and virtual pin location assignments. Altera also recommends that you add a maximum delay timing constraint for the virtual I/O connections in each partition.
4. The Quartus Prime software generates Tcl scripts for all partitions, but in this scenario, you would focus on the partitions that make up the cross-partition critical paths. The following assignments are important in the script:
   - Virtual pin assignments for module pins not connected to device I/O ports in the top-level design.
   - Location constraints for the virtual pins that reflect the initial top-level placement of the pin’s source or destination. These help make the lower-level placement “aware” of its surroundings in the top-level design, leading to a greater chance of timing closure during integration at the top level.
   - `INPUT_MAX_DELAY` and `OUTPUT_MAX_DELAY` timing constraints on the paths to and from the I/O pins of the partition. These constrain the pins to optimize the timing paths to and from the pins.
5. The partition designers source the file provided by the project lead.
   - To source the Tcl script from the Quartus Prime GUI, on the Tools menu, click **Utility Windows** and open the Tcl console. Navigate to the script’s directory, and type the following command:

```
source <filename>
```

— To source the Tcl script at the system command prompt, type the following command:

```
quartus_cdb -t <filename>.tcl
```

6. The partition designers recompile their designs with the new project information or assignments and optimize as needed. When the results are satisfactory and the timing requirements are met, export the updated partition as a .**qxp**.

   The project lead obtains the updated .**qxp** files from the partition designers and adds them to the top-level design. When a new .**qxp** is added to the files list, the software will detect the change in the "source file" and use the new .**qxp** results during the next compilation. If the project uses the advanced import flow, the project lead must perform another import of the new .**qxp**.

   You can now analyze the design to determine whether the timing requirements have been achieved. Because the partitions were compiled with more information about connectivity at the top level, it is more likely that the inter-partition paths have improved placement which helps to meet the timing requirements.

1.9. Creating a Design Floorplan With LogicLock Regions

A floorplan represents the layout of the physical resources on the device. Creating a design floorplan, or floorplanning, describe the process of mapping the logical design hierarchy onto physical regions in the device floorplan. After you have partitioned the design, you can create floorplan location assignments for the design to improve the quality of results when using the incremental compilation design flow. Creating a design floorplan is not a requirement to use an incremental compilation flow, but it is recommended in certain cases. Floorplan location planning can be important for a design that uses incremental compilation for the following reasons:

- To avoid resource conflicts between partitions, predominantly when partitions are imported from another Quartus Prime project
- To ensure a good quality of results when recompiling individual timing-critical partitions

Design floorplan assignments prevent the situation in which the Fitter must place a partition in an area of the device where most resources are already used by other partitions. A physical region assignment provides a reasonable region to re-place logic after a change, so the Fitter does not have to scatter logic throughout the available space in the device.

Floorplan assignments are not required for non-critical partitions compiled as part of the top-level design. The logic for partitions that are not timing-critical (such as simple top-level glue logic) can be placed anywhere in the device on each recompilation, if that is best for your design.

The simplest way to create a floorplan for a partitioned design is to create one LogicLock region per partition (including the top-level partition). If you have a compilation result for a partitioned design with no LogicLock regions, you can use the Chip Planner with the Design Partition Planner to view the partition placement in the device floorplan. You can draw regions in the floorplan that match the general location and size of the logic in each partition. Or, initially, you can set each region with the default settings of **Auto** size and **Floating** location to allow the Quartus Prime software to determine the preliminary size and location for the regions. Then, after compilation, use the Fitter-determined size and origin location as a starting point for

your design floorplan. Check the quality of results obtained for your floorplan location assignments and make changes to the regions as needed. Alternatively, you can perform synthesis, and then set the regions to the required size based on resource estimates. In this case, use your knowledge of the connections between partitions to place the regions in the floorplan.

Once you have created an initial floorplan, you can refine the region using tools in the Quartus Prime software. You can also use advanced techniques such as creating non-rectangular regions by merging LogicLock regions.

You can use the Incremental Compilation Advisor to check that your LogicLock regions meet Altera's guidelines.

Related Information

- Incremental Compilation Advisor on page 32
- Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.9.1. Creating and Manipulating LogicLock Regions

Options in the **LogicLock Regions Properties** dialog box, available from the Assignments menu, allow you to enter specific sizing and location requirements for a region. You can also view and refine the size and location of LogicLock regions in the Quartus Prime Chip Planner. You can select a region in the graphical interface in the Chip Planner and use handles to move or resize the region.

Options in the **Layer Settings** panel in the Chip Planner allow you to create, delete, and modify tasks to determine which objects, including LogicLock regions and design partitions, to display in the Chip Planner.

1.9.2. Changing Partition Placement with LogicLock Changes

When a partition is assigned to a LogicLock region as part of a design floorplan, you can modify the placement of a post-fit partition by moving the LogicLock region. Most assignment changes do not initiate a recompilation of a partition if the netlist type specifies that Fitter results should be preserved. For example, changing a pin assignment does not initiate a recompilation; therefore, the design does not use the new pin assignment unless you change the netlist type to **Post Synthesis** or **Source File**.

Similarly, if a partition's placement is preserved, and the partition is assigned to a LogicLock region, the Fitter always reuses the corresponding LogicLock region size specified in the post-fit netlist. That is, changes to the LogicLock **Size** setting do not initiate refitting if a partition's placement is preserved with the **Post-Fit** netlist type, or with .**qxp** that includes post-fit information.

However, you can use the LogicLock **Origin** location assignment to change or fine-tune the previous Fitter results. When you change the **Origin** setting for a region, the Fitter can move the region in the following manner, depending upon how the placement is preserved for that region's members:

- When you set a new region Origin, the Fitter uses the new origin and replaces the logic, preserving the relative placement of the member logic.
- When you set the region Origin to **Floating**, the following conditions apply:
  - If the region's member placement is preserved with an imported partition, the Fitter chooses a new Origin and re-places the logic, preserving the relative placement of the member logic within the region.
  - If the region's member placement is preserved with a **Post-Fit** netlist type, the Fitter does not change the Origin location, and reuses the previous placement results.

Related Information

What Changes Initiate the Automatic Resynthesis of a Partition? on page 35

1.10. Incremental Compilation Restrictions

1.10.1. When Timing Performance May Not Be Preserved Exactly

Timing performance might change slightly in a partition with placement and routing preserved when other partitions are incorporated or re-placed and routed. Timing changes are due to changes in parasitic loading or crosstalk introduced by the other (changed) partitions. These timing changes are very small, typically less than 30 ps on a timing path. Additional fan-out on routing lines when partitions are added can also degrade timing performance.

To ensure that a partition continues to meet its timing requirements when other partitions change, a very small timing margin might be required. The Fitter automatically works to achieve such margin when compiling any design, so you do not need to take any action.

1.10.2. When Placement and Routing May Not Be Preserved Exactly

The Fitter may have to refit affected nodes if the two nodes are assigned to the same location, due to imported netlists or empty partitions set to re-use a previous post-fit netlist. There are two cases in which routing information cannot be preserved exactly. First, when multiple partitions are imported, there might be routing conflicts because two lower-level blocks could be using the same routing wire, even if the floorplan assignments of the lower-level blocks do not overlap. These routing conflicts are automatically resolved by the Quartus Prime Fitter re-routing on the affected nets. Second, if an imported LogicLock region is moved in the top-level design, the relative placement of the nodes is preserved but the routing cannot be preserved, because the routing connectivity is not perfectly uniform throughout a device.

1.10.3. Using Incremental Compilation With Quartus Prime Archive Files

The post-synthesis and post-fitting netlist information for each design partition is stored in the project database, the **incremental_db** directory. When you archive a project, the database information is not included in the archive unless you include the compilation database in the .**qar** file.

If you want to re-use post-synthesis or post-fitting results, include the database files in the **Archive Project** dialog box so compilation results are preserved. Click **Advanced**, and choose a file set that includes the compilation database, or turn on **Incremental compilation database files** to create a Custom file set.

When you include the database, the file size of the .**qar** archive file may be significantly larger than an archive without the database.

The netlist information for imported partitions is already saved in the corresponding .**qxp**. Imported **.qxp** files are automatically saved in a subdirectory called **imported_partitions**, so you do not need to archive the project database to keep the results for imported partitions. When you restore a project archive, the partition is automatically reimported from the .**qxp** in this directory if it is available.

For new device families with advanced support, a version-compatible database might not be available. In this case, the archive will not include the compilation database. If you require the database files to reproduce the compilation results in the same Quartus Prime version, you can use the following command-line option to archive a full database:

```
quartus_sh --archive -use_file_set full_db [-revision <revision name>]<project name>
```

1.10.4. Formal Verification Support

You cannot use design partitions for incremental compilation if you are creating a netlist for a formal verification tool.

1.10.5. Signal Probe Pins and Engineering Change Orders

ECO and Signal Probe changes are performed only during ECO and Signal Probe compilations. Other compilation flows do not preserve these netlist changes.

When incremental compilation is turned on and your design contains one or more design partitions, partition boundaries are ignored while making ECO changes and Signal Probe signal settings. However, the presence of ECO and/or Signal Probe changes does not affect partition boundaries for incremental compilation. During subsequent compilations, ECO and Signal Probe changes are not preserved regardless of the **Netlist Type** or **Fitter Preservation Level** settings. To recover ECO changes and Signal Probe signals, you must use the Change Manager to re-apply the ECOs after compilation.

For partitions developed independently in separate Quartus Prime projects, the exported netlist includes all currently saved ECO changes and Signal Probe signals. If you make any ECO or Signal Probe changes that affect the interface to the lower-level partition, the software issues a warning message during the export process that this netlist does not work in the top-level design without modifying the top-level HDL code to reflect the lower-level change. After integrating the .**qxp** partition into the top-level design, the ECO changes will not appear in the Change Manager.

Related Information

- Quick Design Debugging Using Signal Probe documentation
- Engineering Change Management with the Chip Planner documentation

1.10.6. Signal Tap Logic Analyzer in Exported Partitions

You can use the Signal Tap Embedded Logic Analyzer in any project that you can compile and program into an Altera device.

When incremental compilation is turned on, debugging logic is added to your design incrementally and you can tap post-fitting nodes and modify triggers and configuration without recompiling the full design. Use the appropriate filter in the Node Finder to find your node names. Use **Signal Tap: post-fitting** if the netlist type is Post-Fit to incrementally tap node names in the post-fit netlist database. Use **Signal Tap: pre-synthesis** if the netlist type is **Source File** to make connections to the source file (pre-synthesis) node names when you synthesize the partition from the source code.

If incremental compilation is turned off, the debugging logic is added to the design during Analysis and Elaboration, and you cannot tap post-fitting nodes or modify debug settings without fully compiling the design.

For design partitions that are being developed independently in separate Quartus Prime projects and contain the logic analyzer, when you export the partition, the Quartus Prime software automatically removes the Signal Tap logic analyzer and related SLD_HUB logic. You can tap any nodes in a Quartus Prime project, including nodes within .**qxp** partitions. Therefore, you can use the logic analyzer within the full top-level design to tap signals from the .**qxp** partition.

You can also instantiate the Signal Tap IP core directly in your lower-level design (instead of using an .**stp** file) and export the entire design to the top level to include the logic analyzer in the top-level design.

Related Information

Design Debugging Using the Signal Tap Embedded Logic Analyzer documentation

1.10.7. External Logic Analyzer Interface in Exported Partitions

You can use the Logic Analyzer Interface in any project that you can compile and program into an Altera device. You cannot export a partition that uses the Logic Analyzer Interface. You must disable the Logic Analyzer Interface feature and recompile the design before you export the design as a partition.

Related Information

In-System Debugging Using External Logic Analyzers documentation

1.10.8. Assignments Made in HDL Source Code in Exported Partitions

Assignments made with I/O primitives or the `altera_attribute` HDL synthesis attribute in lower-level partitions are passed to the top-level design, but do not appear in the top-level .**qsf** file or Assignment Editor. These assignments are considered part of the source netlist files. You can override assignments made in these source files by changing the value with an assignment in the top-level design.

1.10.9. Design Partition Script Limitations

Related Information

Generating Design Partition Scripts on page 42

1.10.9.1. Warnings About Extra Clocks Due to Design Partition Scripts

The generated scripts include applicable clock information for all clock signals in the top-level design. Some of those clocks may not exist in the lower-level projects, so you may see warning messages related to clocks that do not exist in the project. You can ignore these warnings or edit your constraints so the messages are not generated.

1.10.9.2. Synopsys Design Constraint Files for the Timing Analyzer in Design Partition Scripts

After you have compiled a design using Timing Analyzer constraints, and the timing assignments option is turned on in the scripts, a separate Tcl script is generated to create an .**sdc** file for each lower-level project. This script includes only clock constraints and minimum and maximum delay settings for the Timing Analyzer.

Note: PLL settings and timing exceptions are not passed to lower-level designs in the scripts.

Related Information

Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.10.9.3. Wildcard Support in Design Partition Scripts

When applying constraints with wildcards, note that wildcards are not analyzed across hierarchical boundaries. For example, an assignment could be made to these nodes: `Top|A:inst|B:inst|*`, where A and B are lower-level partitions, and hierarchy B is a child of A, that is B is instantiated in hierarchy A. This assignment is applied to modules A, B, and all children instances of B. However, the assignment `Top|A:inst|B:inst*` is applied to hierarchy A, but is not applied to the B instances because the single level of hierarchy represented by `B:inst*` is not expanded into multiple levels of hierarchy. To avoid this issue, ensure that you apply the wildcard to the hierarchical boundary if it should represent multiple levels of hierarchy.

When using the wildcard to represent a level of hierarchy, only single wildcards are supported. This means assignments such as `Top|A:inst|*|B:inst|*` are not supported. The Quartus Prime software issues a warning in these cases.

1.10.9.4. Derived Clocks and PLLs in Design Partition Scripts

If a clock in the top level is not directly connected to a pin of a lower-level partition, the lower-level partition does not receive assignments and constraints from the top-level pin in the design partition scripts.

This issue is of particular importance for clock pins that require timing constraints and clock group settings. Problems can occur if your design uses logic or inversion to derive a new clock from a clock input pin. Make appropriate timing assignments in your lower-level Quartus Prime project to ensure that clocks are not unconstrained.

If the lower-level design uses the top-level project framework from the project lead, the design will have all the required information about the clock and PLL settings. Otherwise, if you use a PLL in your top-level design and connect it to lower-level partitions, the lower-level partitions do not have information about the multiplication or phase shift factors in the PLL. Make appropriate timing assignments in your lower-level Quartus Prime project to ensure that clocks are not unconstrained or constrained with the incorrect frequency. Alternatively, you can manually duplicate the

top-level derived clock logic or PLL in the lower-level design file to ensure that you have the correct multiplication or phase-shift factors, compensation delays and other PLL parameters for complete and accurate timing analysis. Create a design partition for the rest of the lower-level design logic for export to the top level. When the lower-level design is complete, export only the partition that contains the relevant logic.

1.10.9.5. Pin Assignments for GXB and LVDS Blocks in Design Partition Scripts

Pin assignments for high-speed GXB transceivers and hard LVDS blocks are not written in the scripts. You must add the pin assignments for these hard IP blocks in the lower-level projects manually.

1.10.9.6. Virtual Pin Timing Assignments in Design Partition Scripts

Design partition scripts use `INPUT_MAX_DELAY` and `OUTPUT_MAX_DELAY` assignments to specify inter-partition delays associated with input and output pins, which would not otherwise be visible to the project. These assignments require that the software specify the clock domain for the assignment and set this clock domain to ” * ”.

This clock domain assignment means that there may be some paths constrained and reported by the timing analysis engine that are not required.

To restrict which clock domains are included in these assignments, edit the generated scripts or change the assignments in your lower-level Quartus Prime project. In addition, because there is no known clock associated with the delay assignments, the software assumes the worst-case skew, which makes the paths seem more timing critical than they are in the top-level design. To make the paths appear less timing-critical, lower the delay values from the scripts. If required, enter negative numbers for input and output delay values.

1.10.9.7. Top-Level Ports that Feed Multiple Lower-Level Pins in Design Partition Scripts

When a single top-level I/O port drives multiple pins on a lower-level module, it unnecessarily restricts the quality of the synthesis and placement at the lower-level. This occurs because in the lower-level design, the software must maintain the hierarchical boundary and cannot use any information about pins being logically equivalent at the top level. In addition, because I/O constraints are passed from the top-level pin to each of the children, it is possible to have more pins in the lower level than at the top level. These pins use top-level I/O constraints and placement options that might make them impossible to place at the lower level. The software avoids this situation whenever possible, but it is best to avoid this design practice to avoid these potential problems. Restructure your design so that the single I/O port feeds the design partition boundary and the single connection is split into multiple signals within the lower-level partition.

1.10.10. Restrictions on IP Core Partitions

The Quartus Prime software does not support partitions for IP core instantiations. If you use the parameter editor to customize an IP core variation, the IP core generated wrapper file instantiates the IP core. You can create a partition for the IP core generated wrapper file.

The Quartus Prime software does not support creating a partition for inferred IP cores (that is, where the software infers an IP core to implement logic in your design). If you have a module or entity for the logic that is inferred, you can create a partition for that hierarchy level in the design.

The Quartus Prime software does not support creating a partition for any Quartus Prime internal hierarchy that is dynamically generated during compilation to implement the contents of an IP core.

1.10.11. Restrictions on Arria® 10 Transceiver

The Quartus Prime software does not support partitions for Arria® 10 Transceiver PHY or Transceiver PLL. This restriction applies to creating partitions, exporting and importing partitions through Quartus Prime Exported Partition File (`.qxp`). If your design block contains Arria 10 Transceiver PHY or Transceiver PLL, you must exclude the transceivers before creating partition for the design block.

Related Information

Knowledge Base

1.10.12. Register Packing and Partition Boundaries

The Quartus Prime software performs register packing during compilation automatically. However, when incremental compilation is enabled, logic in different partitions cannot be packed together because partition boundaries might prevent cross-boundary optimization. This restriction applies to all types of register packing, including I/O cells, DSP blocks, sequential logic, and unrelated logic. Similarly, logic from two partitions cannot be packed into the same ALM.

1.10.13. I/O Register Packing

Cross-partition register packing of I/O registers is allowed in certain cases where your input and output pins exist in the top-level hierarchy (and the Top partition), but the corresponding I/O registers exist in other partitions.

The following specific circumstances are required for input pin cross-partition register packing:

- The input pin feeds exactly one register.
- The path between the input pin and register includes only input ports of partitions that have one fan-out each.

The following specific circumstances are required for output register cross-partition register packing:

- The register feeds exactly one output pin.
- The output pin is fed by only one signal.
- The path between the register and output pin includes only output ports of partitions that have one fan-out each.

Output pins with an output enable signal cannot be packed into the device I/O cell if the output enable logic is part of a different partition from the output register. To allow register packing for output pins with an output enable signal, structure your HDL code or design partition assignments so that the register and tri-state logic are defined in the same partition.

Bidirectional pins are handled in the same way as output pins with an output enable signal. If the registers that need to be packed are in the same partition as the tri-state logic, you can perform register packing.

The restrictions on tri-state logic exist because the I/O atom (device primitive) is created as part of the partition that contains tri-state logic. If an I/O register and its tri-state logic are contained in the same partition, the register can always be packed with tri-state logic into the I/O atom. The same cross-partition register packing restrictions also apply to I/O atoms for input and output pins. The I/O atom must feed the I/O pin directly with exactly one signal. The path between the I/O atom and the I/O pin must include only ports of partitions that have one fan-out each.

Related Information

Best Practices for Incremental Compilation Partitions and Floorplan Assignments documentation on page 70

1.11. Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script or at a command-line prompt.

1.11.1. Tcl Scripting and Command-Line Examples

The `::quartus::incremental_compilation` Tcl package contains a set of functions for manipulating design partitions and settings related to the incremental compilation feature.

Related Information

- Quartus Prime Software Scripting Support website
  Scripting support information, design examples, and training
- Tcl Scripting documentation
- Command-Line Scripting documentation

1.11.1.1. Creating Design Partitions

To create a design partition to a specified hierarchy name, use the following command:

Example 1. Create Design Partition

```
create_partition [-h | -help] [-long_help] -contents
<hierarchy name> -partition <partition name>
```

Table 4. Tcl Script Command: `create_partition`

Argument	Description
`-h | -help`	Short help
`-long_help`	Long help with examples and possible return values
`-contents <hierarchy name>`	Partition contents (hierarchy assigned to Partition)
`-partition <partition name>`	Partition name

1.11.1.2. Enabling or Disabling Design Partition Assignments During Compilation

To direct the Quartus Prime Compiler to enable or disable design partition assignments during compilation, use the following command:

Example 2. Enable or Disable Partition Assignments During Compilation

```
set_global_assignment -name IGNORE_PARTITIONS <value>
```

Table 5. Tcl Script Command: `set_global_assignment`

Value	Description
OFF	The Quartus Prime software recognizes the design partitions assignments set in the current Quartus Prime project and recompiles the partition in subsequent compilations depending on their netlist status.
ON	The Quartus Prime software does not recognize design partitions assignments set in the current Quartus Prime project and performs a compilation without regard to partition boundaries or netlists.

1.11.1.3. Setting the Netlist Type

To set the partition netlist type, use the following command:

Example 3. Set Partition Netlist Type

```
set_global_assignment -name PARTITION_NETLIST_TYPE <value>
-section_id <partition name>
```

Note: The PARTITION_NETLIST_TYPE command accepts the following values: SOURCE, POST_SYNTH, POST_FIT, and EMPTY.

1.11.1.4. Setting the Fitter Preservation Level for a Post-fit or Imported Netlist

To set the Fitter Preservation Level for a post-fit or imported netlist, use the following command:

Example 4. Set Fitter Preservation Level

```
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
<value> -section_id <partition name>
```

Note: The PARTITION_FITTER_PRESERVATION command accepts the following values: NETLIST_ONLY, PLACEMENT, and PLACEMENT_AND_ROUTING.

1.11.1.5. Preserving High-Speed Optimization

To preserve high-speed optimization for tiles contained within the selected partition, use the following command:

Example 5. Preserve High-Speed Optimization

```
set_global_assignment -name PARTITION_PRESERVE_HIGH_SPEED_TILES_ON
```

1.11.1.6. Specifying the Software Should Use the Specified Netlist and Ignore Source File Changes

To specify that the software should use the specified netlist and ignore source file changes, even if the source file has changed since the netlist was created, use the following command:

Example 6. Specify Netlist and Ignore Source File Changes

```
set_global_assignment -name PARTITION_IGNORE_SOURCE_FILE_CHANGES ON
-section_id "<partition name>"
```

1.11.1.7. Reducing Opening a Project, Creating Design Partitions, andPerforming an Initial Compilation

Scenario background: You open a project called AB_project, set up two design partitions, entities A and B, and then perform an initial full compilation.

Example 7. Set Up and Compile AB_project

```
set project AB_project

load_package incremental_compilation
load_package flow
project_open $project

# Ensure that design partition assignments are not ignored
set_global_assignment -name IGNORE_PARTITIONS \ OFF

# Set up the partitions
create_partition -contents A -name "Partition_A"
create_partition -contents B -name "Partition_B"

# Set the netlist types to post-fit for subsequent
# compilations (all partitions are compiled during the
# initial compilation since there are no post-fit netlists)
set_partition -partition "Partition_A" -netlist_type POST_FIT
set_partition -partition "Partition_B" -netlist_type POST_FIT

# Run initial compilation
export_assignments
execute_flow -full_compile

project_close
```

1.11.1.8. Optimizing the Placement for a Timing-Critical Partition

Scenario background: You have run the initial compilation shown in the example script below. You would like to apply Fitter optimizations, such as physical synthesis, only to partition **A**. No changes have been made to the HDL files. To ensure the previous compilation result for partition **B** is preserved, and to ensure that Fitter optimizations are applied to the post-synthesis netlist of partition **A**, set the netlist type of **B** to **Post-Fit** (which was already done in the initial compilation, but is repeated here for safety), and the netlist type of **A** to **Post-Synthesis**, as shown in the following example:

Example 8. Fitter Optimization for AB_project

```
set project AB_project

load_package flow
load_package incremental_compilation
```

```
load_package project
project_open $project

# Turn on Physical Synthesis Optimization
set_high_effort_fmax_optimization_assignments

# For A, set the netlist type to post-synthesis
set_partition -partition "Partition_A" -netlist_type POST_SYNTH

# For B, set the netlist type to post-fit
set_partition -partition "Partition_B" -netlist_type POST_FIT

# Also set Top to post-fit
set_partition -partition "Top" -netlist_type POST_FIT

# Run incremental compilation
export_assignments
execute_flow -full_compile

project_close
```

1.11.1.9. Generating Design Partition Scripts

To generate design partition scripts, use the following script:

Example 9. Generate Partition Script

```
# load required package
load_package database_manager

# name and open the project
set project <project_path/project_name>
project_open $project

# generate the design partition scripts
generate_bottom_up_scripts <options>

#close project
project_close
```

1.11.1.10. Exporting a Partition

To open a project and load the `::quartus::incremental_compilation` package before you use the Tcl commands to export a partition to a **.qxp** that contains both a post-synthesis and post-fit netlist, with routing, use the following script:

Example 10. Export .qxp

```
# load required package
load_package incremental_compilation

# open project
project_open <project name>

# export partition to the .qxp and set preservation level
export_partition -partition <partition name>
-qxp <.qxp file name> -<options>

#close project
project_close
```

1.11.1.11. Importing a Partition into the Top-Level Design

To import a **.qxp** into a top-level design, use the following script:

Example 11. Import .qxp into Top-Level Design

```
# load required packages
load_package incremental_compilation
load_package project
load_package flow

# open project
project_open <project name>

#import partition
import_partition -partition <partition name> -qxp <.qxp file>
<-options>

#close project
project_close
```

1.11.1.12. Makefiles

For an example of how to use incremental compilation with a `makefile` as part of the team-based incremental compilation design flow, refer to the **read_me.txt** file that accompanies the `incr_comp` example located in the **/qdesigns/incr_comp_makefile** subdirectory.

When using a team-based incremental compilation design flow, the **Generate Design Partition Scripts** dialog box can write makefiles that automatically export lower-level design partitions and import them into the top-level design whenever design files change.

1.12. Document Revision History

Table 6. Document Revision History

Date	Version	Changes
2016.05.03	16.0.0	Stated limitations about deprecated physical synthesis options.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Removed Early Timing Estimate feature support.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings. • Updated DSE II content.
2014.08.18	14.0a10.0	Added restriction about smart compilation in Arria 10 devices.
June 2014	14.0.0	• Dita conversion. • Replaced MegaWizard Plug-In Manager content with IP Catalog and Parameter Editor content. • Revised functional safety section. Added export and import sections.
November 2013	13.1.0	Removed HardCopy device information. Revised information about Rapid Recompile. Added information about functional safety. Added information about flattening sub-partition hierarchies.
November 2012	12.1.0	Added Turning On Supported Cross-boundary Optimizations.
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	• Updated "Tcl Scripting and Command-Line Examples".

continued...

Date	Version	Changes
December 2010	10.1.0	• Changed to new document template. • Reorganized Tcl scripting section. Added description for new feature: **Ignore partitions assignments during compilation** option. • Reorganized "Incremental Compilation Summary" section.
July 2010	10.0.0	• Removed the explanation of the "bottom-up design flow" where designers work completely independently, and replaced with Altera's recommendations for team-based environments where partitions are developed in the same top-level project framework, plus an explanation of the bottom-up process for including independent partitions from third-party IP designers. • Expanded the Merge command explanation to explain how it now accommodates cross-partition boundary optimizations. • Restructured Altera recommendations for when to use a floorplan. • Added "Viewing the Contents of a Quartus Prime Exported Partition File (.qxp)" section. • Reorganized chapter to make design flow scenarios more visible; integrated into various sections rather than at the end of the chapter.
October 2009	9.1.0	• Redefined the bottom-up design flow as team-based and reorganized previous design flow examples to include steps on how to pass top-level design information to lower-level designers. • Moved SDC Constraints from Lower-Level Partitions section to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in volume 1 of the *Quartus Prime Handbook.* • Reorganized the "Conclusion" section. • Removed HardCopy APEX and HardCopy Stratix Devices section.
March 2009	9.0.0	• Split up netlist types table • Moved "Team-Based Incremental Compilation Design Flow" into the "Including or Integrating partitions into the Top-Level Design" section. • Added new section "Including or Integrating Partitions into the Top-Level Design". • Removed "Exporting a Lower-Level Partition that Uses a JTAG Feature" restriction • Other edits throughout chapter
November 2008	8.1.0	• Added new section "Importing SDC Constraints from Lower-Level Partitions" on page 2–44 • Removed the Incremental Synthesis Only option • Removed section "OpenCore Plus Feature for MegaCore Functions in Bottom-Up Flows" • Removed section "Compilation Time with Physical Synthesis Optimizations" • Added information about using a **.qxp** as a source design file without importing • Reorganized several sections • Updated Figure 2–10

Related Information

Documentation Archive

For previous versions of the *Quartus Prime Handbook*, search the documentation archives.

2. Best Practices for Incremental Compilation Partitions and Floorplan Assignments

2.1. About Incremental Compilation and Floorplan Assignments

This manual provides guidelines to help you partition your design to take advantage of Quartus Prime incremental compilation, and to help you create a design floorplan using Logic Lock (Standard) regions when they are recommended to support the compilation flow.

The Quartus Prime incremental compilation feature allows you to partition a design, compile partitions separately, and reuse results for unchanged partitions. Incremental compilation provides the following benefits:

- Reduces compilation times by an average of 75% for large design changes
- Preserves performance for unchanged design blocks
- Provides repeatable results and reduces the number of compilations
- Enables team-based design flows

Related Information

Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design documentation on page 7

2.2. Incremental Compilation Overview

Quartus Prime incremental compilation is an optional compilation flow that enhances the default Quartus Prime compilation. If you do not partition your design for incremental compilation, your design is compiled using the default "flat" compilation flow.

To prepare your design for incremental compilation, you first determine which logical hierarchy boundaries should be defined as separate partitions in your design, and ensure your design hierarchy and source code is set up to support this partitioning. You can then create design partition assignments in the Quartus Prime software to specify which hierarchy blocks are compiled independently as partitions (including empty partitions for missing or incomplete logic blocks).

During compilation, Quartus Prime Analysis & Synthesis and the Fitter create separate netlists for each partition. Netlists are internal post-synthesis and post-fit database representations of your design.

In subsequent compilations, you can select which netlist to preserve for each partition. You can either reuse the synthesis or fitting netlist, or instruct the Quartus Prime software to resynthesize the source files. You can also use compilation results exported from another Quartus Prime project.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

When you make changes to your design, the Quartus Prime software recompiles only the designated partitions and merges the new compilation results with existing netlists for other partitions, according to the degree of results preservation you set with the netlist for each design partition.

In some cases, Altera recommends that you create a design floorplan with placement assignments to constrain parts of the design to specific regions of the device.

You must use the partial reconfiguration (PR) feature in conjunction with incremental compilation for Stratix® V device families. Partial reconfiguration allows you to reconfigure a portion of the FPGA dynamically, while the remainder of the device continues to operate as intended.

Related Information

Introduction to Design Floorplans on page 105

2.2.1. Recommendations for the Netlist Type

For subsequent compilations, you specify which post-compilation netlist you want to use with the netlist type for each partition.

Use the following general guidelines to set the netlist type for each partition:

- **Source File—**Use this setting to resynthesize the source code (with any new assignments, and replace any previous synthesis or Fitter results).
  - If you modify the design source, the software automatically resynthesizes the partitions with the appropriate netlist type, which makes the **Source File** setting optional in this case.
  - Most assignments do not trigger an automatic recompilation, so you must set the netlist type to **Source File** to compile the source files with new assignments or constraints that affect synthesis.
- **Post-Synthesis** (default)**—**Use this setting to re-fit the design (with any new Fitter assignments), but preserve the synthesis results when the source files have not changed. If it is difficult to meet the required timing performance, you can use this setting to allow the Fitter the most flexibility in placement and routing. This setting does not reduce compilation time as much as the **Post-Fit** setting or preserve timing performance from the previous compilation.
- **Post-Fit**—Use this setting to preserve Fitter and performance results when the source files have not changed. This setting reduces compilation time the most, and preserves timing performance from the previous compilation.
- **Post-Fit with Fitter Preservation Level set to Placement**—Use the **Advanced Fitter Preservation Level** setting on the **Advanced** tab in the **Design Partition Properties** dialog box to allow more flexibility and find the best routing for all partitions given their placement.

The Quartus Prime software Rapid Recompile feature instructs the Compiler to reuse the compatible compilation results if most of the design has not changed since the last compilation. This feature reduces compilation time and preserves performance when there are small and isolated design changes within a partition, and works with all netlist type settings. With this feature, you do not have control over which parts of the design are recompiled; the Compiler determines which parts of the design must be recompiled.

2.3. Design Flows Using Incremental Compilation

The Quartus Prime incremental compilation feature supports various design flows. Your design flow affects design optimization and the amount of design planning required to obtain optimal results.

2.3.1. Using Standard Flow

In the standard incremental compilation flow, the top-level design is divided into partitions, which can be compiled and optimized together in one Quartus Prime project. If another team member or IP provider is developing source code for the top-level design, they can functionally verify their partition independently, and then simply provide the partition's source code to the project lead for integration into the top-level design. If the project lead wants to compile the top-level design when source code is not yet complete for a partition, they can create an empty placeholder for the partition until the code is ready to be added to the top-level design.

Compiling all design partitions in a single Quartus Prime project ensures that all design logic is compiled with a consistent set of assignments, and allows the software to perform global placement and routing optimizations. Compiling all design logic together is beneficial for FPGA design flows because all parts of the design must use the same shared set of device resources. Therefore, it is often easier to ensure good quality of results when partitions are developed within a single top-level Quartus Prime project.

2.3.2. Using Team-Based Flow

In the team-based incremental compilation flow, you can design and optimize partitions by accessing the top-level project from a shared source control system or creating copies of the top-level Quartus Prime project framework. As development continues, designers export their partition so that the post-synthesis netlist or post-fitting results can be integrated into the top-level design.

2.3.2.1. Using Third-Party IP Delivery Flow

If required for third-party IP delivery, or in cases where designers cannot access a shared or copied top-level project framework, you can create and compile a design partition logic in isolation and export a partition that is included in the top-level project. If this type of design flow is necessary, planning and rigorous design guidelines might be required to ensure that designers have a consistent view of project assignments and resource allocations. Therefore, developing partitions in completely separate Quartus Prime projects can be more challenging than having all source code within one project or developing design partitions within the same top-level project framework.

2.3.3. Combining Design Flows

You can also combine design flows and use exported partitions only when it is necessary to support your design environment. For example, if the top-level design includes one or more design blocks that will be optimized by remote designers or IP providers, you can integrate those blocks into the reserved partitions in the top-level design when the code is complete, but also have other partitions that will be developed within the top-level design.

If any partitions are developed independently, the project lead must ensure that top-level constraints (such as timing constraints, any relevant floorplan or pin assignments, and optimization settings) are consistent with those used by all designers.

2.3.4. Project Management in Team-Based Design Flows

If possible, each team member should work within the same top-level project framework. Using the same project framework amongst team members ensures that designers have the settings and constraints needed for their partition and allows designers to analyze how their design block interacts with other partitions in the top-level design.

2.3.4.1. Using a Source Control System

In a team-based environment where designers have access to the project through source control software, each designer can use project files as read-only and develop their partition within the source control system. As designers check in their completed partitions, other team members can see how their partitions interact.

2.3.4.2. Using a Copy of the Top-Level Project

If designers do not have access to a source control system, the project lead can provide each designer with a copy of the top-level project framework to use as they develop their partitions. In both cases, each designer exports their completed design as a partition, and then the project lead integrates the partition into the top-level design. The project lead can choose to use only the post-synthesis netlist and rerun placement and routing, or to use the post-fitting results to preserve the placement and routing results from the other designer's projects. Using post-synthesis partitions gives the Fitter the most flexibility and is likely to achieve a good result for all partitions, but if one partition has difficultly meeting timing, the designer can choose to preserve their successful fitting results.

2.3.4.3. Using a Separate Project

Alternatively, designers can use their own Quartus Prime project for their independent design block. You might use this design flow if a designer, such as a third-party IP provider, does not have access to the entire top-level project framework. In this case, each designer must create their own project with all the relevant assignments and constraints. This type of design flow requires more planning and rigorous design guidelines. If the project lead plans to incorporate the post-fitting compilation results for the partition, this design flow requires especially careful planning to avoid resource conflicts.

2.3.4.4. Using Scripts

The project lead also has the option to generate design partition scripts to manage resource and timing budgets in the top-level design when partitions are developed outside the top-level project framework. Scripts make it easier for designers of independent Quartus Prime projects to follow instructions from the project lead. The Quartus Prime design partition scripts feature creates Tcl scripts or `.tcl` files and makefiles that an independent designer can run to set up an independent Quartus Prime project.

2.3.4.5. Using Constraints

If designers create Quartus Prime assignments or timing constraints for their partitions, they must ensure that the constraints are integrated into the top-level design. If partition designers use the same top-level project framework (and design hierarchy), the constraints or Synopsys Design Constraints File (`.sdc`) can be easily copied or included in the top-level design. If partition designers use a separate Quartus Prime project with a different design hierarchy, they must ensure that constraints are applied to the appropriate level of hierarchy in the top-level design, and design the `.sdc` for easy delivery to the project lead.

Related Information

- Including SDC Constraints from Lower-Level Partitions for Third-Party IP Delivery on page 101
- Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design documentation on page 7
  Information about the different types of incremental design flows and example applications, as well as documented restrictions and limitations

2.4. Why Plan Partitions and Floorplan Assignments?

Incremental design flows typically require more planning than flat compilations, and require you to be more rigorous about following good design practices. For example, you might need to structure your source code or design hierarchy to ensure that logic is grouped correctly for optimization. It is easier to implement the correct logic grouping early in the design cycle than to restructure the code later.

Planning involves setting up the design logic for partitioning and may also involve planning placement assignments to create a floorplan. Not all design flows require floorplan assignments. If you decide to add floorplan assignments later, when the design is close to completion, well-planned partitions make floorplan creation easier. Poor partition or floorplan assignments can worsen design area utilization and performance and make timing closure more difficult.

As FPGA devices get larger and more complex, following good design practices become more important for all design flows. Adhering to recommended synchronous design practices makes designs more robust and easier to debug. Using an incremental compilation flow adds additional steps and requirements to your project, but can provide significant benefits in design productivity by preserving the performance of critical blocks and reducing compilation time.

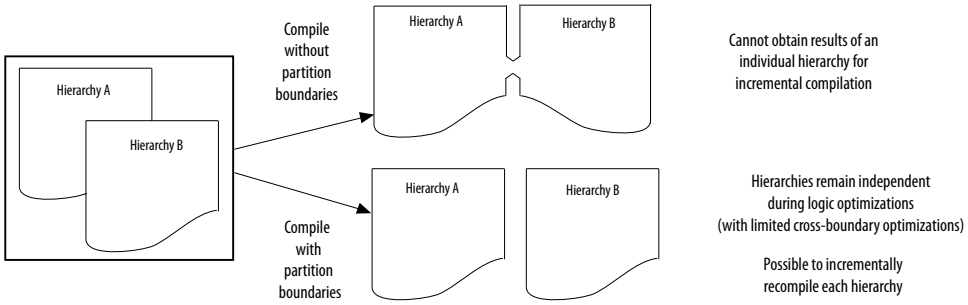
Related Information

Introduction to Design Floorplans on page 105

2.4.1. Partition Boundaries and Optimization

The logical hierarchical boundaries between partitions are treated as hard boundaries for logic optimization (except for some limited cross-boundary optimizations) to allow the software to size and place each partition independently. The figure shows the effects of partition boundaries during logic optimization.

Figure 8. Effects of Partition Boundaries During Logic Optimization



2.4.1.1. Merging Partitions

You can use the **Merge** command in the Design Partitions window to combine hierarchical partitions into a single partition, as long as they share the same immediate parent partition. Merging partitions allows additional optimizations for partition I/O ports that connect between or feed more than one of the merged hierarchical design blocks.

When partitions are placed together, the Fitter can perform placement optimizations on the design as a whole to optimize the placement of cross-boundary paths. However, the Fitter can never perform logic optimizations such as physical synthesis across the partition boundary. If partitions are fit separately in different projects, or if some partitions use previous post-fitting results, the Fitter does not place and route the entire cross-boundary path at the same time and cannot fully optimize placement across the partition boundaries. Good design partitions can be placed independently because cross-partition paths are not the critical timing paths in the design.

2.4.1.2. Resource Utilization

There are possible timing performance utilization effects due to partitioning and creating a floorplan. Not all designs encounter these issues, but you should consider these effects if a flat version of your design is very close to meeting its timing requirements, or is close to using all the device resources, before adding partition or floorplan assignments:

- Partitions can increase resource utilization due to cross-boundary optimization limitations if the design does not follow partitioning guidelines. Floorplan assignments can also increase resource utilization because regions can lead to unused logic. If your device is full with the flat version of your design, you can focus on creating partitions and floorplan assignments for timing-critical or often-changing blocks to benefit most from incremental compilation.
- Partitions and floorplan assignments might increase routing utilization compared to a flat design. If long compilation times are due to routing congestion, you might not be able to use the incremental flow to reduce compilation time. Review the Fitter messages to check how much time is spent during routing optimizations to determine the percentage of routing utilization. When routing is difficult, you can use incremental compilation to lock the routing for routing-critical blocks only (with other partitions empty), and then compile the rest of the design after the critical blocks meets their requirements.
- Partitions can reduce timing performance in some cases because of the optimization and resource effects described above, causing longer logic delays. Floorplan assignments restrict logic placement, which can make it more difficult for the Fitter to meet timing requirements. Use the guidelines in this manual to reduce any effect on your design performance.

Related Information

- Design Partition Guidelines on page 79
- Checking Floorplan Quality on page 113

2.4.1.3. Turning On Supported Cross-Boundary Optimizations

You can improve the optimizations performed between design partitions by turning on the cross-boundary optimizations feature. You can select the optimizations as individual assignments for each partition. This allows the cross-boundary optimization feature to give you more control over the optimizations that work best for your design.

You can turn on the cross-boundary optimizations for your design partitions on the **Advanced** tab of the **Design Partition Properties** dialog box. Once you change the optimization settings, the Quartus Prime software recompiles your partition from source automatically. Cross-boundary optimizations include the following: propagate constants, propagate inversions on partition inputs, merge inputs fed by a common source, merge electrically equivalent bidirectional pins, absorb internal paths, and remove logic connected to dangling outputs.

Cross-boundary optimizations are implemented top-down from the parent partition into the child partition, but not vice-versa. The cross-boundary optimization feature cannot be used with partitions with multiple personas (partial reconfiguration partitions).

Although more partitions allow for a greater reduction in compilation time, consider limiting the number of partitions to prevent degradation in the quality of results. Creating good design partitions and good floorplan location assignments helps to improve the design resource utilization and timing performance results for cross-partition paths.

2.5. Guidelines for Incremental Compilation

2.5.1. General Partitioning Guidelines

The first step in planning your design partitions is to organize your source code so that it supports good partition assignments. Although you can assign any hierarchical block of your design as a design partition or merge hierarchical blocks into the same partition, following the design guidelines presented below ensures better results.

2.5.1.1. Plan Design Hierarchy and Design Files

You begin the partitioning process by planning the design hierarchy. When you assign a hierarchical instance as a design partition, the partition includes the assigned instance and entities instantiated below that are not defined as separate partitions. You can use the **Merge** command in the Design Partitions window to combine hierarchical partitions into a single partition, as long as they have the same immediate parent partition.

- When planning your design hierarchy, keep logic in the "leaves" of the hierarchy instead of having logic at the top-level of the design so that you can isolate partitions if required.
- Create entities that can form partitions of approximately equal size. For example, do not instantiate small entities at the same hierarchy level, because it is more difficult to group them to form reasonably-sized partitions.
- Create each entity in an independent file. The Quartus Prime software uses a file checksum to detect changes, and automatically recompiles a partition if its source file changes and its netlist type is set to either post-synthesis or post-fit. If the design entities for two partitions are defined in the same file, changes to the logic in one partition initiates recompilation for both partitions.
- Design dependencies also affect which partitions are compiled when a source file changes. If two partitions rely on the same lower-level entity definition, changes in that lower-level entity affect both partitions. Commands such as VHDL `use` and Verilog HDL `include` create dependencies between files, so that changes to one file can trigger recompilations in all dependent files. Avoid these types of file dependencies if possible. The Partition Dependent Files report for each partition in the Analysis & Synthesis section of the Compilation report lists which files contribute to each partition.

2.5.1.2. Using Partitions with Third-Party Synthesis Tools

Incremental compilation works well with third-party synthesis tools in addition to Quartus Prime Integrated Synthesis. If you use a third-party synthesis tool, set up your tool to create a separate Verilog Quartus Mapping File (`.vqm`) or EDIF Input File (`.edf`) netlist for each hierarchical partition. In the Quartus Prime software, designate the top-level entity from each netlist as a design partition. The `.vqm` or `.edf` netlist file is treated as the source file for the partition in the Quartus Prime software.

Related Information

Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design documentation on page 7

2.5.1.3. Partition Design by Functionality and Block Size

Initially, you should partition your design along functional boundaries. In a top-level system block diagram, each block is often a natural design partition. Typically, each block of a system is relatively independent and has more signal interaction internally than interaction between blocks, which helps reduce optimizations between partition boundaries. Keeping functional blocks together means that synthesis and fitting can optimize related logic as a whole, which can lead to improved optimization.

- Consider how many partitions you want to maintain in your design to determine the size of each partition. Your compilation time reduction goal is also a factor, because compiling small partitions is typically faster than compiling large partitions.
- There is no minimum size for partitions; however, having too many partitions can reduce the quality of results by limiting optimization. Ensure that the design partitions are not too small. As a general guideline, each partition should contain more than approximately 2,000 logic elements (LEs) or adaptive logic modules (ALMs). If your design is incomplete when you partition the design, use previous designs to help estimate the size of each block.

2.5.1.4. Partition Design by Clock Domain and Timing Criticality

Consider which clock in your design feeds the logic in each partition. If possible, keep clock domains within one partition. When a clock signal is isolated to one partition, it reduces dependence on other partitions for timing optimization. Isolating a clock domain to one partition also allows better use of regional clock routing networks if the partition logic is constrained to one region of the design. Additionally, limiting the number of clocks within each partition simplifies the timing requirements for each partition during optimization. Use an appropriate subsystem to implement the required logic for any clock domain transfers (such as a synchronization circuit, dual-port RAM, or FIFO). You can include this logic inside the partition at one side of the transfer.

Try to isolate timing-critical logic from logic that you expect to easily meet timing requirements. Doing so allows you to preserve the satisfactory results for non-critical partitions and focus optimization iterations on only the timing-critical portions of the design to minimize compilation time.

Related Information

Analyzing and Optimizing the Design Floorplan with the Chip Planner documentation
Information about clock domains and their affect on partition design

2.5.1.5. Consider What Is Changing

When assigning partitions, you should consider what is changing in the design. Is there intellectual property (IP) or reused logic for which the source code will not change during future design iterations? If so, define the logic in its own partition so that you can compile one time and immediately preserve the results and not have to compile that part of the design again. Is logic being tuned or optimized, or are specifications changing for part of the design? If so, define changing logic in its own partition so that you can recompile only the changing part while the rest of the design remains unchanged.

As a general rule, create partitions to isolate logic that will change from logic that will not change. Partitioning a design in this way maximizes the preservation of unchanged logic and minimizes compilation time.

2.5.2. Design Partition Guidelines

Follow the design partition guidelines below when you create or modify the HDL code for each design block that you might want to assign as a design partition. You do not need to follow all the recommendations exactly to achieve a good quality of results with the incremental compilation flow, but adhering to as many as possible maximizes your chances for success.

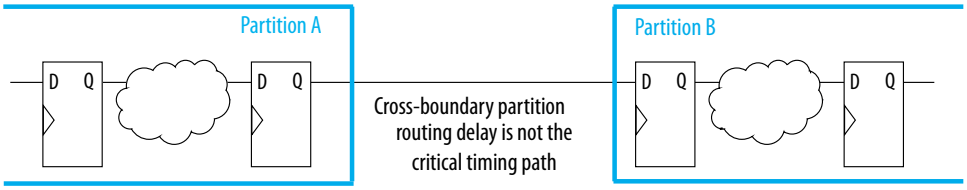
The design partition guidelines include examples of the types of optimizations that are prevented by partition boundaries, and describes how you can structure or modify your partitions to avoid these limitations.

2.5.2.1. Register Partition Inputs and Outputs

Use registers at partition input and output connections that are potentially timing-critical. Registers minimize the delays on inter-partition paths and prevent the need for cross-boundary optimizations.

If every partition boundary has a register as shown in the figure, every register-to-register timing path between partitions includes only routing delay. Therefore, the timing paths between partitions are likely not timing-critical, and the Fitter can generally place each partition independently from other partitions. This advantage makes it easier to create floorplan location assignments for each separate partition, and is especially important for flows in which partitions are placed independently in separate Quartus Prime projects. Additionally, the partition boundary does not affect combinational logic optimization because each register-to-register logic path is contained within a single partition.

Figure 9. Registering Partition I/O



If a design cannot include both input and output registers for each partition due to latency or resource utilization concerns, choose to register one end of each connection. If you register every partition output, for example, the combinational logic that occurs in each cross-partition path is included in one partition so that it can be optimized together.

It is a good synchronous design practice to include registers for every output of a design block. Registered outputs ensure that the input timing performance for each design block is controlled exclusively within the destination logic block.

Related Information

- Partition Statistics Report on page 100
- Incremental Compilation Advisor on page 97

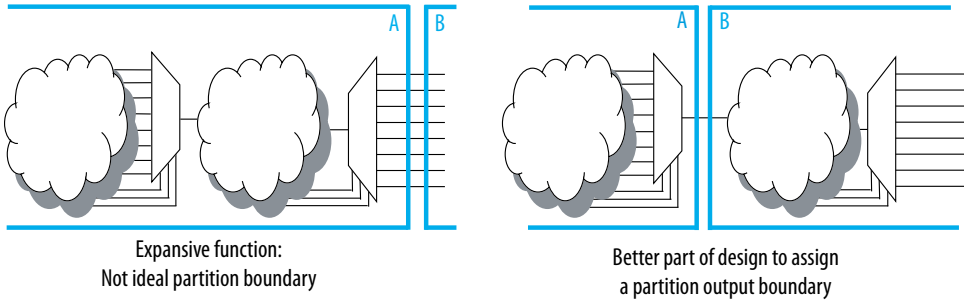
2.5.2.2. Minimize Cross-Partition-Boundary I/O

Minimize the number of I/O paths that cross between partition boundaries to keep logic paths within a single partition for optimization. Doing so makes partitions more independent for both logic and placement optimization.

This guideline is most important for timing-critical and high-speed connections between partitions, especially in cases where the input and output of each partition is not registered. Slow connections that are not timing-critical are acceptable because they should not impact the overall timing performance of the design. If there are timing-critical paths between partitions, rework or merge the partitions to avoid these inter-partition paths.

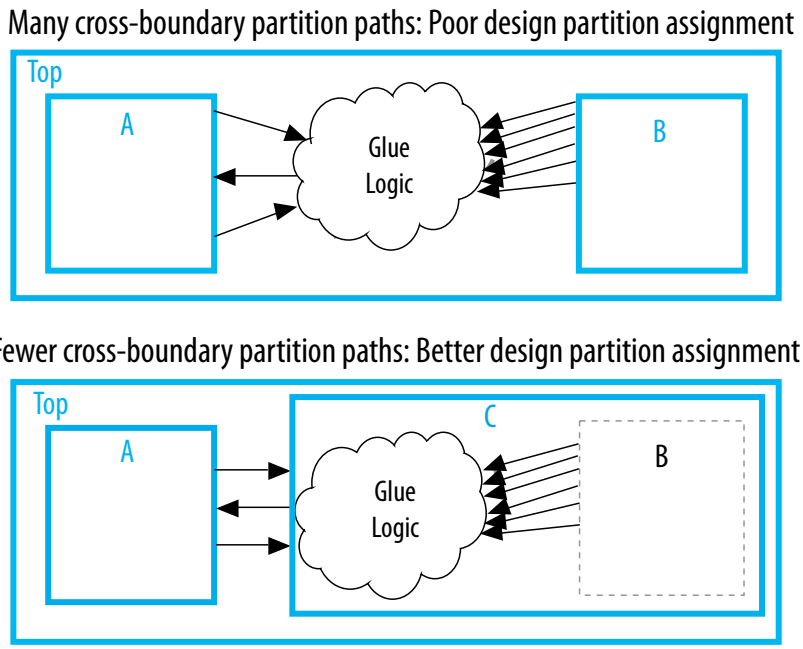
When dividing your design into partitions, consider the types of functions at the partition boundaries. The figure shows an expansive function with more outputs than inputs in the left diagram, which makes a poor partition boundary, and, on the right side, a better place to assign the partition boundary that minimizes cross-partition I/Os. Adding registers to one or both sides of the cross-partition path in this example would further improve partition quality.

Figure 10. Minimizing I/O Between Partitions by Moving the Partition Boundary



Another way to minimize connections between partitions is to avoid using combinational ”glue logic” between partitions. You can often move the logic to the partition at one end of the connection to keep more logic paths within one partition. For example, the bottom diagram includes a new level of hierarchy C defined as a partition instead of block B. Clearly, there are fewer I/O connections between partitions A and C than between partitions A and B.

Figure 11. Minimizing I/O between Partitions by Modifying Glue Logic



Related Information

- Partition Statistics Report on page 100
- Incremental Compilation Advisor on page 97

2.5.2.3. Examine the Need for Logic Optimization Across Partitions

Partition boundaries prevent logic optimizations across partitions (except for some limited cross-boundary optimizations).

In some cases, especially if part of the design is complete or comes from another designer, the designer might not have followed these guidelines when the source code was created. These guidelines are not mandatory to implement an incremental compilation flow, but can improve the quality of results. If assigning a partition affects resource utilization or timing performance of a design block as compared to the flat design, it might be due to one of the issues described in the logic optimization across partitions guidelines below. Many of the examples suggest simple changes to your partition definitions or hierarchy to move the partition boundary to improve your results.

The following guidelines ensure that your design does not require logic optimization across partition boundaries:

2.5.2.3.1. Keep Logic in the Same Partition for Optimization and Merging

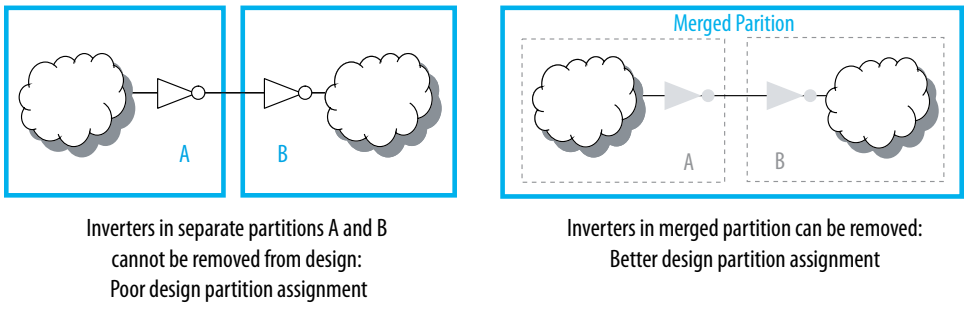
If your design logic requires logic optimization or merging to obtain optimal results, ensure that all the logic is part of the same partition because only limited cross-boundary optimizations are permitted.

Example—Combinational Logic Path

If a combinational logic path is split across two partitions, the logic cannot be optimized or merged into one logic cell in the device. This effect can result in an extra logic cell in the path, increasing the logic delay. As a very simple example, consider two inverters on the same signal in two different partitions, A and B, as shown in the left diagram of the figure. To maintain correct incremental functionality, these two inverters cannot be removed from the design during optimization because they occur in different design partitions. The Quartus Prime software cannot use information about other partitions when it compiles each partition, because each partition is allowed to change independently from the other.

On the right side of the figure, partitions A and B are merged to group the logic in blocks A and B into one partition. If the two blocks A and B are not under the same immediate parent partition, you can create a wrapper file to define a new level of hierarchy that contains both blocks, and set this new hierarchy block as the partition. With the logic contained in one partition, the software can optimize the logic and remove the two inverters (shown in gray), which reduces the delay for that logic path. Removing two inverters is not a significant reduction in resource utilization because inversion logic is readily available in Altera device architecture. However, this example is a simple demonstration of the types of logic optimization that are prevented by partition boundaries.

Figure 12. Keeping Logic in the Same Partition for Optimization



Inverters in separate partitions A and B cannot be removed from design: Poor design partition assignment

Inverters in merged partition can be removed: Better design partition assignment

Example—Fitter Merging

In a flat design, the Fitter can also merge logical instantiations into the same physical device resource. With incremental compilation, logic defined in different partitions cannot be merged to use the same physical device resource.

For example, the Fitter can merge two single-port RAMs from a design into one dedicated RAM block in the device. If the two RAMs are defined in different partitions, the Fitter cannot merge them into one dedicated device RAM block.

This limitation is a only a concern if merging is required to fit the design in the target device. Therefore, you are more likely to encounter this issue during troubleshooting rather than during planning, if your design uses more logic than is available in the device.

2.5.2.3.2. Merging PLLs and Transceivers (GXB)

Multiple instances of the ALTPLL IP core can use the same PLL resource on the device. Similarly, GXB transceiver instances can share high-speed serial interface (HSSI) resources in the same quad as other instances. The Fitter can merge multiple

instantiations of these blocks into the same device resource, even if it requires optimization across partitions. Therefore, there are no restrictions for PLLs and high-speed transceiver blocks when setting up partitions.

2.5.2.4. Keep Constants in the Same Partition as Logic

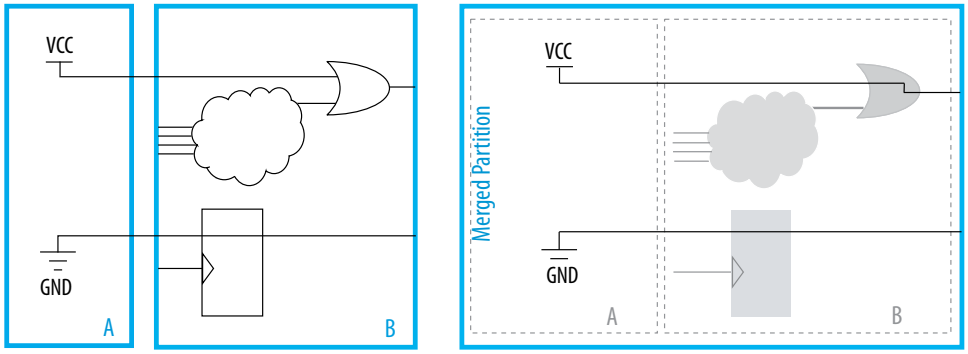
Because the Quartus Prime software cannot fully optimize across a partition boundary, constants are not propagated across partition boundaries, except from parent partition to child partition. A signal that is constant ($1/V_{CC}$ or 0/GND) in one partition cannot affect another partition.

2.5.2.4.1. Example—Constants in Merged Partitions

For example, the left diagram of the figure shows part of a design in which partition A defines some signals as constants (and assumes that the other input connections come from elsewhere in the design and are not shown in the figure). Constants such as these could appear due to parameter or generic settings or configurations with parameters, setting a bus to a specific set of values, or could result from optimizations that occur within a group of logic. Because the blocks are independent, the software cannot optimize the logic in block B based on the information from block A. The right side of the figure shows a merged partition that groups the logic in blocks A and B. If the two blocks A and B are not under the same immediate parent partition, you can create a wrapper file to define a new level of hierarchy that contains both blocks, and set this new hierarchical block as the partition.

Within the single merged partition, the Quartus Prime software can use the constants to optimize and remove much of the logic in block B (shown in gray), as shown in the figure.

Figure 13. Keeping Constants in the Same Partition as the Logic They Feed



Connections to constants in another partition: Poor design partition assignment

Constants in merged partition are used to optimize: Better design partition assignment

Related Information

- Partition Statistics Report on page 100
- Incremental Compilation Advisor on page 97

2.5.2.5. Avoid Signals That Drive Multiple Partition I/O or Connect I/O Together

Do not use the same signal to drive multiple ports of a single partition or directly connect two ports of a partition. If the same signal drives multiple ports of a partition, or if two ports of a partition are directly connected, those ports are logically

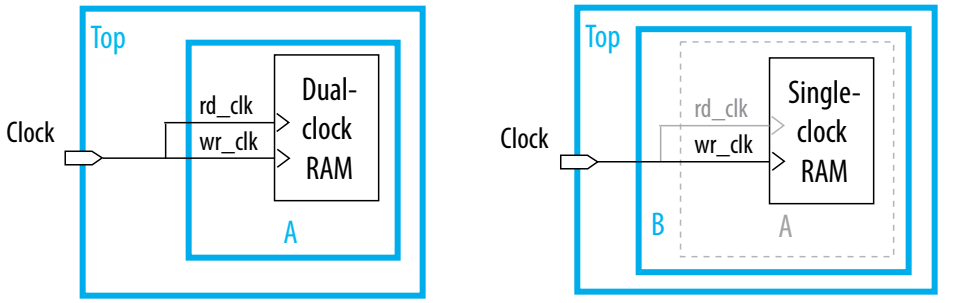
equivalent. However, the software has limited information about connections made in another partition (including the top-level partition), the compilation cannot take advantage of the equivalence. This restriction usually produces sub-optimal results.

If your design has these types of connections, redefine the partition boundaries to remove the affected ports. If one signal from a higher-level partition feeds two input ports of the same partition, feed the one signal into the partition, and then make the two connections within the partition. If an output port drives an input port of the same partition, the connection can be made internally without going through any I/O ports. If an input port drives an output port directly, the connection can likely be implemented without the ports in the lower-level partition by connecting the signals in a higher-level partition.

2.5.2.5.1. Example—Single Signal Driving More Than One Port

The figure shows an example of one signal driving more than one port. The left diagram shows a design where a single clock signal is used to drive both the read and write clocks of a RAM block. Because the RAM block is compiled as a separate partition A, the RAM block is implemented as though there are two unique clocks. If you know that the port connectivity will not change (that is, the ports will always be driven by the same signal in the top-level partition), redefine the port interface so that there is only a single port that can drive both connections inside the partition. You can create a wrapper file to define a partition that has fewer ports, as shown in the diagram on the right side. With the single clock fed into the partition, the RAM can be optimized into a single-clock RAM instead of a dual-clock RAM. Single-clock RAM can provide better performance in the device architecture. Additionally, partition A might use two global routing lines for the two copies of the clock signal. Partition B can use one global line that fans out to all destinations. Using just the single port connection prevents overuse of global routing resources.

Figure 14. Preventing One Signal from Driving Multiple Partition Inputs



Related Information

Incremental Compilation Advisor on page 97

2.5.2.6. Invert Clocks in Destination Partitions

For best results, clock inversion should be performed in the destination logic array block (LAB) because each LAB contains clock inversion circuitry in the device architecture. In a flat compilation, the Quartus Prime software can optimize a clock

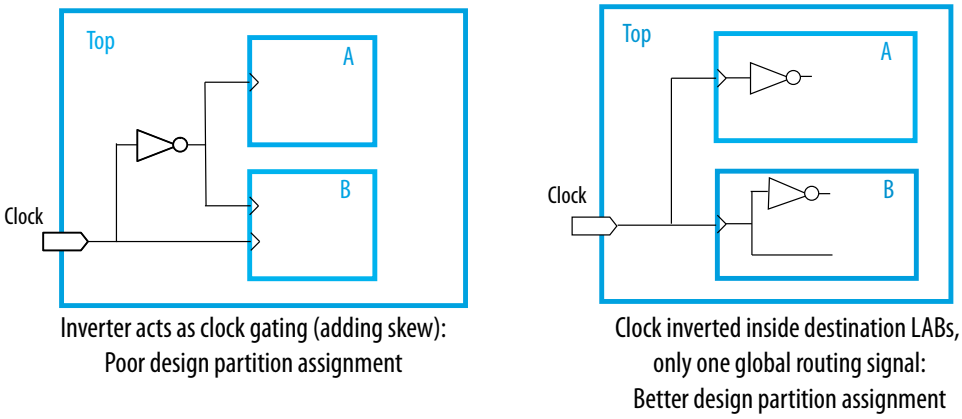
inversion to propagate it to the destination LABs regardless of where the inversion takes place in the design hierarchy. However, clock inversion cannot propagate through a partition boundary (except from a parent partition to a child partition) to take advantage of the inversion architecture in the destination LABs.

2.5.2.6.1. Example—Clock Signal Inversion

With partition boundaries as shown in the left diagram of the figure, the Quartus Prime software uses logic to invert the signal in the partition that defines the inversion (the top-level partition in this example), and then routes the signal on a global clock resource to its destinations (in partitions A and B). The inverted clock acts as a gated clock with high skew. A better solution is to invert the clock signal in the destination partitions as shown on the right side of the diagram. In this case, the correct logic and routing resources can be used, and the signal does not behave like a gated clock.

The figure shows the clock signal inversion in the destination partitions.

Figure 15. Inverting Clock Signal in Destination Partitions



Notice that this diagram also shows another example of a single pin feeding two ports of a partition boundary. In the left diagram, partition B does not have the information that the clock and inverted clock come from the same source. In the right diagram, partition B has more information to help optimize the design because the clock is connected as one port of the partition.

2.5.2.7. Connect I/O Pin Directly to I/O Register for Packing Across Partition Boundaries

The Quartus Prime software allows cross-partition register packing of I/O registers in certain cases where your input and output pins are defined in the top-level hierarchy (and the top-level partition), but the corresponding I/O registers are defined in other partitions.

Input pin cross-partition register packing requires the following specific circumstances:

- The input pin feeds exactly one register.
- The path between the input pin and register includes only input ports of partitions that have one fan-out each.

Output pin cross-partition register packing requires the following specific circumstances:

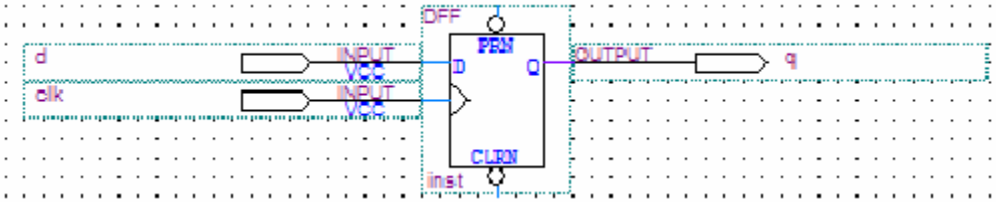
- The register feeds exactly one output pin.
- The output pin is fed by only one signal.
- The path between the register and output pin includes only output ports of partitions that have one fan-out each.

The following examples of I/O register packing illustrate this point using Block Design File (`.bdf`) schematics to describe the design logic.

2.5.2.7.1. Example 1—Output Register in Partition Feeding Multiple Output Pins

In this example, the subdesign contains a single register.

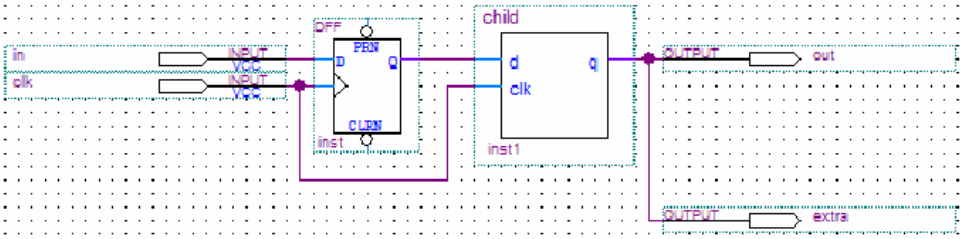
Figure 16. Subdesign with One Register, Designated as a Separate Partition



If the top-level design instantiates the subdesign with a single fan-out directly feeding an output pin, and designates the subdesign as a separate design partition, the Quartus Prime software can perform cross-partition register packing because the single partition port feeds the output pin directly.

In this example, the top-level design instantiates the subdesign as an output register with more than one fan-out signal.

Figure 17. Top-Level Design Instantiating the Subdesign with Two Output Pins



In this case, the Quartus Prime software does not perform output register packing. If there is a **Fast Output Register** assignment on pin `out`, the software issues a warning that the Fitter cannot pack the node to an I/O pin because the node and the I/O cell are connected across a design partition boundary.

This type of cross-partition register packing is not allowed because it requires modification to the interface of the subdesign partition. To perform incremental compilation, the Quartus Prime software must preserve the interface of design partitions.

To allow the Quartus Prime software to pack the register in the subdesign with the output pin out in the figure, restructure your HDL code so that output registers directly connect to output pins by making one of the following changes:

- Place the register in the same partition as the output pin. The simplest method is to move the register from the subdesign partition into the partition containing the output pin. Doing so guarantees that the Fitter can optimize the two nodes without violating partition boundaries.
- Duplicate the register in your subdesign HDL so that each register feeds only one pin, and then connect the extra output pin to the new port in the top-level design. Doing so converts the cross-partition register packing into the simplest case where each register has a single fan-out.

Figure 18. Modified Subdesign with Two Output Registers and Two Output Ports

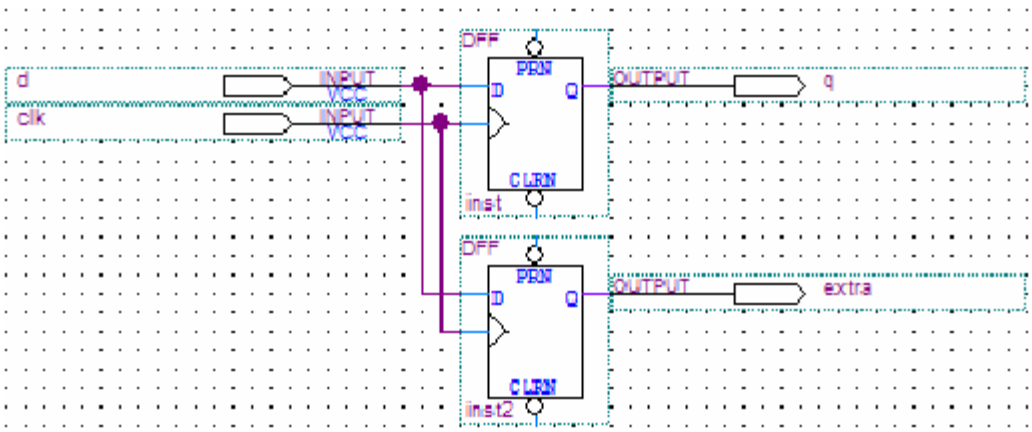
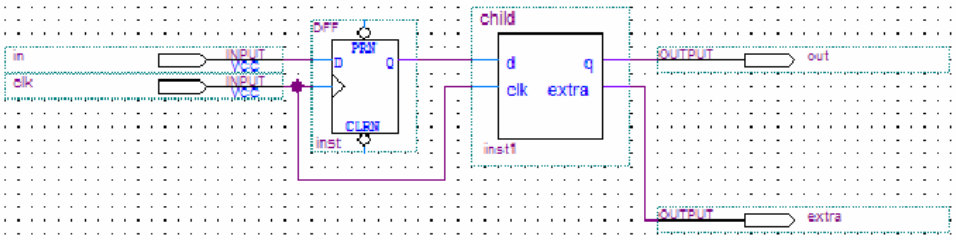


Figure 19. Modified Top-Level Design Connecting Two Output Ports to Output Pins



2.5.2.7.2. Example 2—Input Register in Partition Fed by an Inverted Input Pin or Output Register in Partition Feeding an Inverted Output Pin

In this example, a subdesign designated as a separate partition contains a register. The top-level design in the figure instantiates the subdesign as an input register with the input pin inverted. The top-level design instantiates the subdesign as an output register with the signal inverted before feeding an output pin.

Figure 20. Top-Level Design Instantiating Subdesign as an Input Register with an Inverted Input Pin

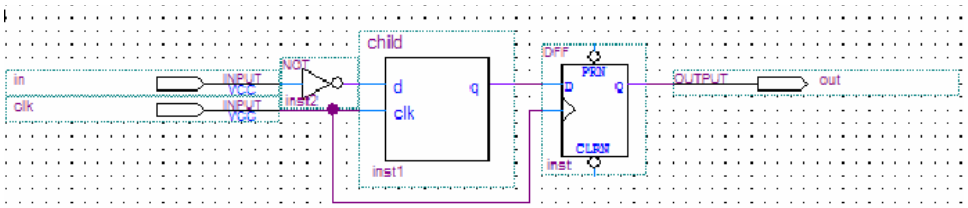
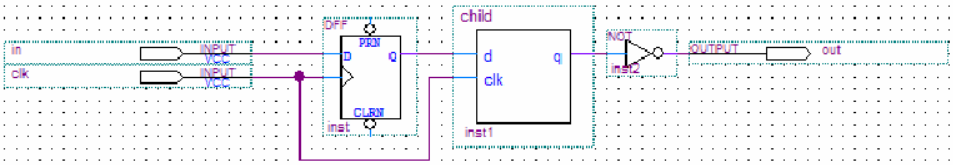


Figure 21. Top-Level Design Instantiating the Subdesign as an Output Register Feeding an Inverted Output Pin



In these cases, the Quartus Prime software does not perform register packing. If there is a **Fast Input Register** assignment on pin `in`, as shown in the top figure, or a **Fast Output Register** assignment on pin `out`, as shown in the bottom figure, the Quartus Prime software issues a warning that the Fitter cannot pack the node to an I/O pin because the node and I/O cell are connected across a design partition boundary.

This type of register packing is not allowed because it requires moving logic across a design partition boundary to place into a single I/O device atom. To perform register packing, either the register must be moved out of the subdesign partition, or the inverter must be moved into the subdesign partition to be implemented in the register.

To allow the Quartus Prime software to pack the single register in the subdesign with the input pin `in`, as shown in top figure or the output pin `out`, as shown in the bottom figure, restructure your HDL code to place the register in the same partition as the inverter by making one of the following changes:

- Move the register from the subdesign partition into the top-level partition containing the pin. Doing so ensures that the Fitter can optimize the I/O register and inverter without violating partition boundaries.
- Move the inverter from the top-level block into the subdesign, and then connect the subdesign directly to a pin in the top-level design. Doing so allows the Fitter to optimize the inverter into the register implementation, so that the register is directly connected to a pin, which enables register packing.

2.5.2.8. Do Not Use Internal Tri-States

Internal tri-state signals are not recommended for FPGAs because the device architecture does not include internal tri-state logic. If designs use internal tri-states in a flat design, the tri-state logic is usually converted to OR gates or multiplexing logic. If tri-state logic occurs on a hierarchical partition boundary, the Quartus Prime software cannot convert the logic to combinational gates because the partition could be connected to a top-level device I/O through another partition.

The figures below show a design with partitions that are not supported for incremental compilation due to the internal tri-state output logic on the partition boundaries. Instead of using internal tri-state logic for partition outputs, implement the correct logic to select between the two signals. Doing so is good practice even when there are no partitions, because such logic explicitly defines the behavior for the internal signals instead of relying on the Quartus Prime software to convert the tri-state signals into logic.

Figure 22. Unsupported Internal Tri-State Signals

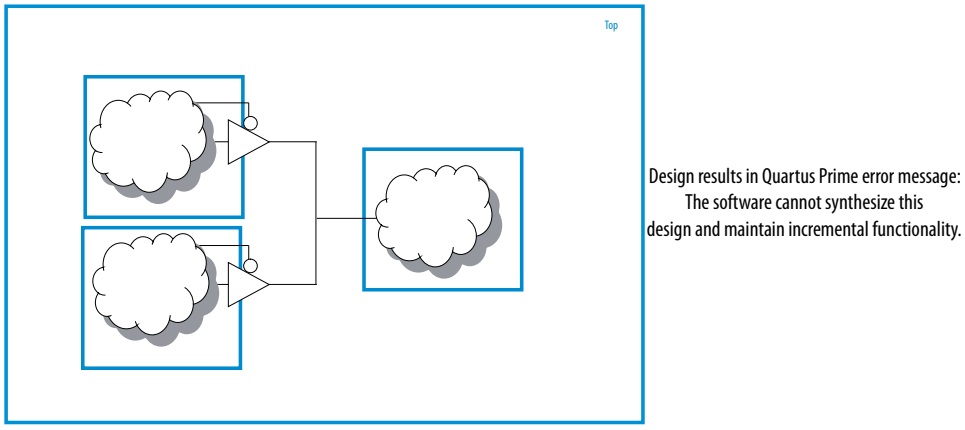
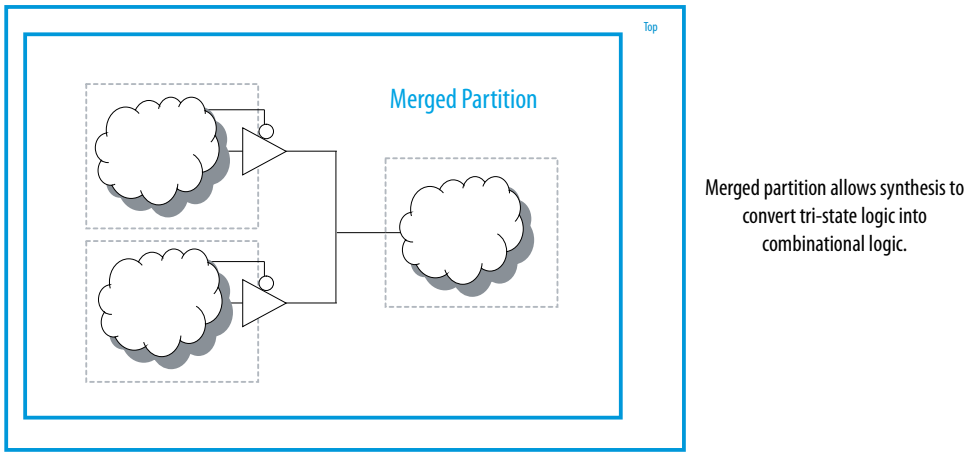


Figure 23. Merged Partition Allows Synthesis to Convert Internal Tri-State Logic to Combinational Logic



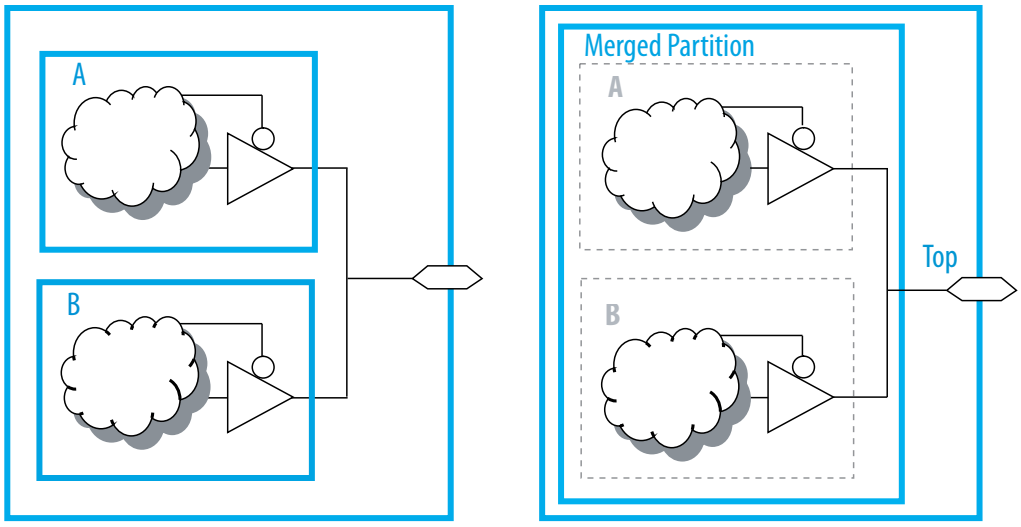
Do not use tri-state signals or bidirectional ports on hierarchical partition boundaries, unless the port is connected directly to a top-level I/O pin on the device. If you must use internal tri-state logic, ensure that all the control and destination logic is contained in the same partition, in which case the Quartus Prime software can convert the internal tri-state signals into combinational logic as in a flat design. In this example, you can also merge all three partitions into one partition, as shown in the bottom figure, to allow the Quartus Prime software to treat the logic as internal tri-state and perform the same type of optimization as a flat design. If possible, you should avoid using internal
tri-state logic in any Altera FPGA design to ensure that you get the desired implementation when the design is compiled for the target device architecture.

2.5.2.9. Include All Tri-State and Enable Logic in the Same Partition

When multiple output signals use tri-state logic to drive a device output pin, the Quartus Prime software merges the logic into one tri-state output pin. The Quartus Prime software cannot merge tri-state outputs into one output pin if any of the tri-state logic occurs on a partition boundary. Similarly, output pins with an output enable signal cannot be packed into the device I/O cell if the output enable logic is part of a different partition from the output register. To allow register packing for output pins with an output enable signal, structure your HDL code or design partition assignments so that the register and enable logic are defined in the same partition.

The figure shows a design with tri-state output signals that feed a device bidirectional I/O pin (assuming that the input connection feeds elsewhere in the design and is not shown in the figure). In the left diagram below, the tri-state output signals appear as the outputs of two separate partitions. In this case, the Quartus Prime software cannot implement the specified logic and maintain incremental functionality. In the right diagram, partitions A and B are merged to group the logic from the two blocks. With this single partition, the Quartus Prime software can merge the two tri-state output signals and implement them in the tri-state logic available in the device I/O element.

Figure 24. Including All Tri-State Output Logic in the Same Partition



2.5.2.10. Summary of Guidelines Related to Logic Optimization Across Partitions

To ensure that your design does not require logic optimization across partitions, follow the guidelines below:

- Include logic in the same partition for optimization and merging
- Include constants in the same partition as logic
- Avoid signals that drive multiple partition I/O or connect I/O together
- Invert clocks in destination partitions

- Connect I/O directly to I/O register for packing across partition boundaries
- Do not use internal tri-states
- Include all tri-state and enable logic in the same partition

Remember that these guidelines are not mandatory when implementing an incremental compilation flow, but can improve the quality of results. When creating source design code, follow these guidelines and organize your HDL code to support good partition boundaries. For designs that are complete, assess whether assigning a partition affects the resource utilization or timing performance of a design block as compared to the flat design. Make the appropriate changes to your design or hierarchy, or merge partitions as required, to improve your results.

2.5.3. Consider a Cascaded Reset Structure

Designs typically have a global asynchronous reset signal where a top-level signal feeds all partitions. To minimize skew for the high fan-out signal, the global reset signal is typically placed onto a global routing resource.

In some cases, having one global reset signal can lead to recovery and removal time problems. This issue is not specific to incremental flows; it could be applicable in any large high-speed design. In an incremental flow, the global reset signal creates a timing dependency between the top-level partition and lower-level partitions.

For incremental compilation, it is helpful to minimize the impact of global structures. To isolate each partition, consider adding reset synchronizers. Using cascaded reset structures, the intent is to reduce the inter-partition fan-out of the reset signal, thereby minimizing the effect of the global signal. Reducing the fan-out of the global reset signal also provides more flexibility in routing the cascaded signals, and might help recovery and removal times in some cases.

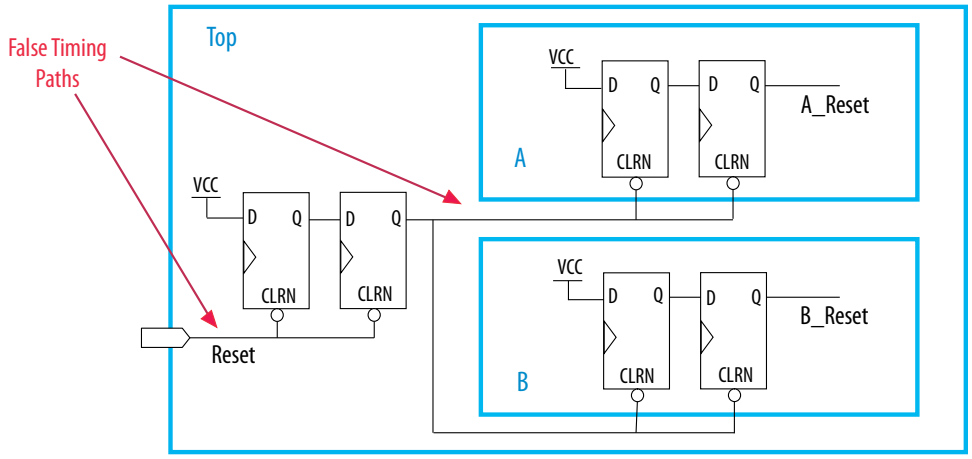
This recommendation can help in large designs, regardless of whether you are using incremental compilation. However, if one global signal can feed all the logic in its domain and meet recovery and removal times, this recommendation may not be applicable for your design. Minimizing global structures is more relevant for high-performance designs where meeting timing on the reset logic can be challenging. Isolating each partition and allowing more flexibility in global routing structures is an additional advantage in incremental flows.

If you add additional reset synchronizers to your design, latency is also added to the reset path, so ensure that this is acceptable in your design. Additionally, parts of the design may come out of the reset state in different clock cycles. You can balance the latency or add hand-shaking logic between partitions, if necessary, to accommodate these differences.

The signal is first synchronized on the chip following good synchronous design practices, meaning that the design asynchronously resets, but synchronously releases from reset to avoid any race conditions or metastability problems. Then, to minimize the impact of global structures, the circuit employs a divide-and-conquer approach for the reset structure. By implementing a cascaded reset structure, the reset paths for each partition are independent. This structure reduces the effect of inter-partition dependency because the inter-partition reset signals can now be treated as false paths for timing analysis. In some cases, the reset signal of the partition can be placed on local lines to reduce the delay added by routing to a global routing line. In other cases, the signal can be routed on a regional or quadrant clock signal.

The figure shows a cascaded reset structure.

Figure 25. Cascaded Reset Structure



This circuit design can help you achieve timing closure and partition independence for your global reset signal. Evaluate the circuit and consider how it works for your design.

2.5.4. Design Partition Guidelines for Third-Party IP Delivery

There are additional design guidelines that can improve incremental compilation flows where exported partitions are developed independently. These guidelines are not always required, but are usually recommended if the design includes partitions compiled in a separate Quartus Prime project, such as when delivering intellectual property (IP). A unique challenge of IP delivery for FPGAs is the fact that the partitions developed independently must share a common set of resources. To minimize issues that might arise from sharing a common set of resources, you can design partitions within a single Quartus Prime project, or a copy of the top-level design. A common project ensures that designers have a consistent view of the top-level design framework.

Alternatively, an IP designer can export just the post-synthesis results to be integrated in the top-level design when the post-fitting results from the IP project are not required. Using a post-synthesis netlist provides more flexibility to the Quartus Prime Fitter, so that less resource allocation is required. If a common project is not possible, especially when the project lead plans to integrate the IP's post-fitting results, it is important that the project lead and IP designer clearly communicate their requirements.

Related Information

Project Management in Team-Based Design Flows on page 73

2.5.4.1. Allocate Logic Resources

In an incremental compilation design flow in which designers, such as third-party IP providers, optimize partitions and then export them to a top-level design, the Quartus Prime software places and routes each partition separately. In some cases, partitions can use conflicting resources when combined at the top level. Allocation of logic resources requires that you decide on a set of logic resources (including I/O, LAB logic

blocks, RAM and DSP blocks) that the IP block will ”own”. This process can be interactive; the project lead and the IP designer might work together to determine what resources are required for the IP block and are available in the top-level design.

You can constrain logic utilization for the IP core using design floorplan location assignments. The design should specify I/O pin locations with pin assignments.

You can also specify limits for Quartus Prime synthesis to allocate and balance resources. This procedure can also help if device resources are overused in the individual partitions during synthesis.

In the standard synthesis flow, the Quartus Prime software can perform automated resource balancing for DSP blocks or RAM blocks and convert some of the logic into regular logic cells to prevent overuse.

You can use the Quartus Prime synthesis options to control inference of IP cores that use the DSP, or RAM blocks. You can also use the IP Catalog and Parameter Editor to customize your RAM or DSP IP cores to use regular logic instead of the dedicated hardware blocks.

Related Information

Introduction to Design Floorplans on page 105

2.5.4.2. Allocate Global Routing Signals and Clock Networks if Required

In most cases, you do not have to allocate global routing signals because the Quartus Prime software finds the best solution for the global signals. However, if your design is complex and has multiple clocks, especially for a partition developed by a third-party IP designer, you may have to allocate global routing resources between various partitions.

Global routing signals can cause conflicts when independent partitions are integrated into a top-level design. The Quartus Prime software automatically promotes high fan-out signals to use global routing resources available in the device. Third-party partitions can use the same global routing resources, thus causing conflicts in the top-level design. Additionally, LAB placement depends on whether the inputs to the logic cells within the LAB use a global clock signal. Problems can occur if a design does not use a global signal in a lower-level partition, but does use a global signal in the top-level design.

If the exported IP core is small, you can reduce the potential for problems by using constraints to promote clock and high fan-out signals to regional routing signals that cover only part of the device, instead of global routing signals. In this case, the Quartus Prime software is likely to find a routing solution in the top-level design because there are many regional routing signals available on most Altera devices, and designs do not typically overuse regional resources.

To ensure that an IP block can utilize a regional clock signal, view the resource coverage of regional clocks in the Chip Planner, and then align LogicLock regions that constrain partition placement with available global clock routing resources. For example, if the LogicLock region for a particular partition is limited to one device quadrant, that partition’s clock can use a regional clock routing type that covers only one device quadrant. When all partition logic is available, the project lead can compile the entire design at the top level with floorplan assignments to allow the use of regional clocks that span only a part of the device.

If global resources are heavily used in the overall design, or the IP designer requires global clocks for their partition, you can set up constraints to avoid signal overuse at the top-level by assigning the appropriate type of global signals or setting a maximum number of clock signals for the partition.

You can use the **Global Signal** assignment to force or prevent the use of a global routing line, making the assignment to a clock source node or signal. You can also assign certain types of global clock resources in some device families, such as regional clocks. For example, if you have an IP core, such as a memory interface that specifies the use of a dual regional clock, you can constrain the IP to part of the device covered by a regional clock and change the **Global Signal** assignment to use a regional clock. This type of assignment can reduce clocking congestion and conflicts.

Alternatively, partition designers can specify the number of clocks allowed in the project using the maximum clocks allowed options in the **Advanced Settings (Fitter)** dialog box. Specify **Maximum number of clocks of any type allowed**, or use the **Maximum number of global clocks allowed**, **Maximum number of regional clocks allowed**, and **Maximum number of periphery clocks allowed** options to restrict the number of clock resources of a particular type in your design.

If you require more control when planning a design with integrated partitions, you can assign a specific signal to use a particular clock network in newer device families by assigning the clock control block instance called CLKCTRL. You can make a point-to-point assignment from a clock source node to a destination node, or a single-point assignment to a clock source node with the **Global Clock CLKCTRL Location** logic option. Set the assignment value to the name of the clock control block: CLKCTRL_G<*global network number*> for a global routing network, or CLKCTRL_R<*regional network number*> for a dedicated regional routing network in the device.

If you want to disable the automatic global promotion performed in the Fitter to prevent other signals from being placed on global (or regional) routing networks, turn off the **Auto Global Clock** and **Auto Global Register Control Signals** options in the **Advanced Settings (Fitter)** dialog box.

If you are using design partition scripts for independent partitions, the Quartus Prime software can automatically write the commands to pass global constraints and turn off automatic options.

Alternatively, to avoid problems when integrating partitions into the top-level design, you can direct the Fitter to discard the placement and routing of the partition netlist by using the post-synthesis netlist, which forces the Fitter to reassign all the global signals for the partition when compiling the top-level design.

2.5.4.3. Assign Virtual Pins

Virtual pins map lower-level design I/Os to internal cells. If you are developing an IP block in an independent Quartus Prime project, use virtual pins when the number of I/Os on a partition exceeds the device I/O count, and to increase the timing accuracy of cross-partition paths.

You can create a virtual pin assignment in the Assignment Editor for partition I/Os that will become internal nodes in the top-level design. When you apply the Virtual Pin assignment to an input pin, the pin no longer appears as an FPGA pin, but is fixed to GND or VCC in the design. The assigned pin is not an open node. Leave the clock pins mapped to I/O pins to ensure proper routing.

You can specify locations for the virtual pins that correspond to the placement of other partitions, and also make timing assignments to the virtual pins to define a timing budget. Virtual pins are created automatically from the top-level design if you use design partition scripts. The scripts place the virtual pins to correspond with the placement of the other partitions in the top-level design.

Note: Tri-state outputs cannot be assigned as virtual pins because internal tri-state signals are not supported in Altera devices. Connect the signal in the design with regular logic, or allow the software to implement the signal as an external device I/O pin.

2.5.4.4. Perform Timing Budgeting if Required

If you optimize partitions independently and integrate them to the top-level design, or compile with empty partitions, any unregistered paths that cross between partitions are not optimized as entire paths. In these cases, the Quartus Prime software has no information about the placement of the logic that connects to the I/O ports. If the logic in one partition is placed far away from logic in another partition, the routing delay between the logic can lead to problems in meeting timing requirements. You can reduce this effect by ensuring that input and output ports of the partitions are registered whenever possible. Additionally, using the same top-level project framework helps to avoid this problem by providing the software with full information about other design partitions in the top-level design.

To ensure that the software correctly optimizes the input and output logic in any independent partitions, you might be required to perform some manual timing budgeting. For each unregistered timing path that crosses between partitions, make timing assignments on the corresponding I/O path in each partition to constrain both ends of the path to the budgeted timing delay. Assigning a timing budget for each part of the connection ensures that the software optimizes the paths appropriately.

When performing manual timing budgeting in a partition for I/O ports that become internal partition connections in a top-level design, you can assign location and timing constraints to the virtual pin that represents each connection to further improve the quality of the timing budget.

Note: If you use design partition scripts, the Quartus Prime software can write I/O timing budget constraints automatically for virtual pins.

2.5.4.5. Drive Clocks Directly

When partitions are exported from another Quartus Prime project, you should drive partition clock inputs directly with device clock input pins.

Connecting the clock signal directly avoids any timing analysis difficulties with gated clocks. Clock gating is never recommended for FPGA designs because of potential glitches and clock skew. Clock gating can be especially problematic with exported partitions because the partitions have no information about gating that takes place at the top-level design or in another partition. If a gated clock is required in a partition, perform the gating within that partition.

Direct connections to input clock pins also allows design partition scripts to send constraints from the top-level device pin to lower-level partitions.

Related Information

Invert Clocks in Destination Partitions on page 84

2.5.4.6. Recreate PLLs for Lower-Level Partitions if Required

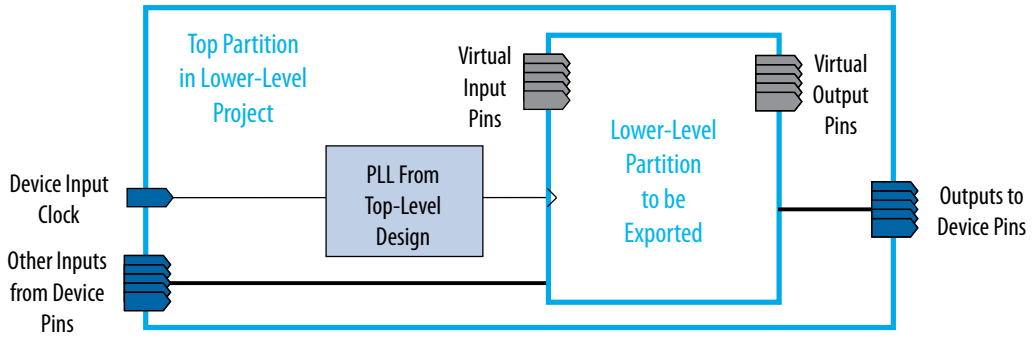
If you connect a PLL in your top-level design to partitions designed in separate Quartus Prime projects by third-party IP designers, the IP partitions do not have information about the multiplication, phase shift, or compensation delays for the PLL in the top-level design. To accommodate the PLL timing, you can make appropriate timing assignments in the projects created by IP designers to ensure that clocks are not left unconstrained or constrained with an incorrect frequency. Alternatively, you can duplicate the top-level PLL (or other derived clock logic) in the design file for the project created by the IP designer to ensure that you have the correct PLL parameters and clock delays for a complete and accurate timing analysis.

If the project lead creates a copy of the top-level project framework that includes all the settings and constraints needed for the design, this framework should include PLLs and other interface logic if this information is important to optimize partitions.

If you use a separate Quartus Prime project for an independent design block (such as when a designer or third-party IP provider does not have access to the entire design framework), include a copy of the top-level PLL in the lower-level partition as shown in figure.

In either case, the IP partition in the separate Quartus Prime project should contain just the partition logic that will be exported to the top-level design, while the full project includes more information about the top-level design. When the partition is complete, you can export just the partition without exporting the auxiliary PLL components to the top-level design. When you export a partition, the Quartus Prime software exports any hierarchy under the specified partition into the Quartus Prime Exported Partition File (`.qxp`), but does not include logic defined outside the partition (the PLL in this example).

Figure 26. Recreating a Top-Level PLL in a Lower-Level Partition



2.6. Checking Partition Quality

There are several tools you can use to create and analyze partitions in the Quartus Prime software. Take advantage of these tools to assess your partition quality, and use the information to improve your design or assignments as required to achieve the best results.

2.6.1. Incremental Compilation Advisor

You can use the Incremental Compilation Advisor to ensure that your design follows Altera's recommendations for creating design partitions and implementing the incremental compilation design flow methodology. Each recommendation in the Incremental Compilation Advisor provides an explanation, describes the effect of the recommendation, and provides the action required to make the suggested change.

Related Information

- Incremental Compilation Advisor on page 97
- Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design documentation on page 7

2.6.2. Design Partition Planner

The Design Partition Planner allows you to view design connectivity and hierarchy, and can assist you in creating effective design partitions that follow the guidelines in this manual. You can also use the Design Partition Planner to optimize design performance by isolating and resolving failing paths on a partition-by-partition basis.

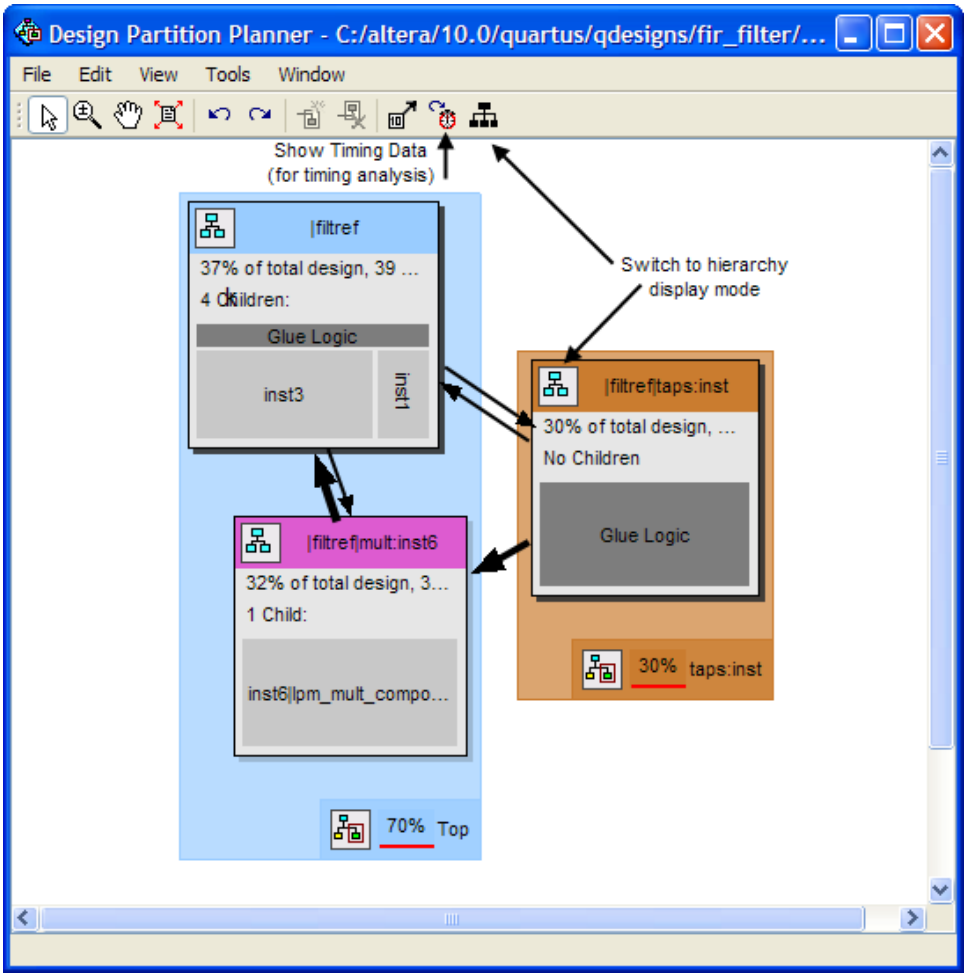
To view a design and create design partitions in the Design Partition Planner, you must first compile the design, or perform Analysis & Synthesis. In the Design Partition Planner, the design appears as a single top-level design block, with lower-level instances displayed as color-specific boxes.

In the Design Partition Planner, you can show connectivity between blocks and extract instances from the top-level design block. When you extract entities, connection bundles are drawn between entities, showing the number of connections existing between pairs of entities. When you have extracted a design block that you want to set as a design partition, right-click that design block, and then click **Create Design Partition**.

The Design Partition Planner also has an auto-partition feature that creates partitions based on the size and connectivity of the hierarchical design blocks. You can right-click the design block you want to partition (such as the top-level design hierarchy), and then click **Auto-Partition Children**. You can then analyze and adjust the partition assignments as required.

The figure shows the Design Partition Planner after making a design partition assignment to one instance and dragging another instance away from the top-level block within the same partition (two design blocks in the pale blue shaded box). The figure shows the connections between each partition and information about the size of each design instance.

Figure 27. Design Partition Planner



You can switch between connectivity display mode and hierarchical display mode, to examine the view-only hierarchy display. You can also remove the connection lines between partitions and I/O banks by turning off **Display connections to I/O banks**, or use the settings on the **Connection Counting** tab in the **Bundle Configuration** dialog box to adjust how the connections are counted in the bundles.

To optimize design performance, confine failing paths within individual design partitions so that there are no failing paths passing between partitions. In the top-level entity, child entities that contain failing paths are marked by a small red dot in the upper right corner of the entity box.

To view the critical timing paths from a timing analyzer report, first perform a timing analysis on your design, and then in the Design Partition Planner, click **Show Timing Data** on the View menu.

2.6.3. Viewing Design Partition Planner and Floorplan Side-by-Side

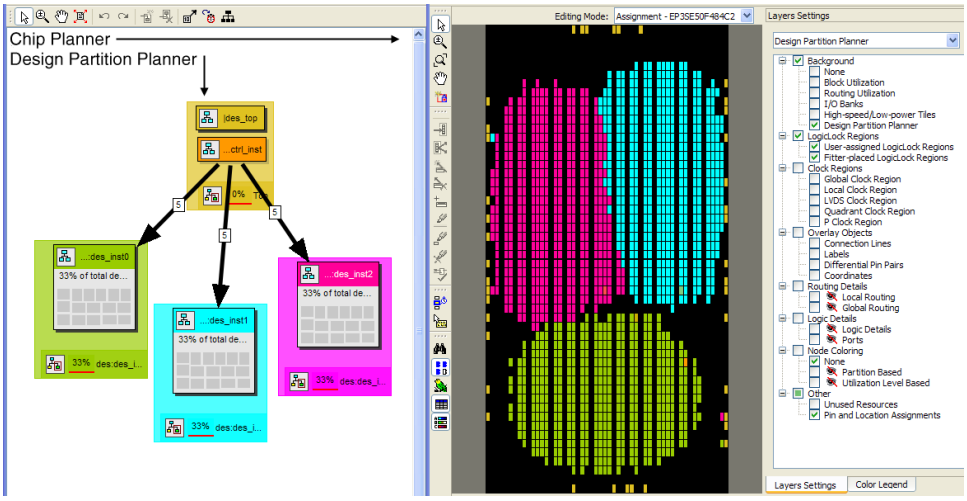
You can use the Design Partition Planner together with the Chip Planner to analyze natural placement groupings. This information can help you decide whether the design blocks should be grouped together in one partition, or whether they will make good partitions in the next compilation. It can also help determine whether the logic can easily be constrained by a LogicLock region. If logic naturally groups together when compiled without placement constraints, you can probably assign a reasonably sized LogicLock region to constrain the placement for subsequent compilations. You can experiment by extracting different design blocks in the Design Partition Planner and viewing the placement results of those design blocks from the previous compilation.

To view the Design Partition Planner and Chip Planner side-by-side, open the Design Partition Planner, and then open the Chip Planner and select the **Design Partition Planner** task. The **Design Partition Planner** task displays the physical locations of design entities with the same colors as in the Design Partition Planner.

In the Design Partition Planner, you can extract instances of interest from their parents by dragging and dropping, or with the **Extract from Parent** command. Evaluate the physical locations of instances in the Chip Planner and the connectivity between instances displayed in the Design Partition Planner. An entity is generally not suitable to be set as a separate design partition or constrained in a LogicLock region if the Chip Planner shows it physically dispersed over a noncontiguous area of the device after compilation. Use the Design Partition Planner to analyze the design connections. Child instances that are unsuitable to be set as separate design partitions or placed in LogicLock regions can be returned to their parent by dragging and dropping, or with the **Collapse to Parent** command.

The figure shows a design displayed in the Design Partition Planner and the Chip Planner with different colors for the top-level design and the three major design instances.

Figure 28. Design Partition Planner and Chip Planner



2.6.4. Partition Statistics Report

You can view statistics about design partitions in the Partition Merge Partition Statistics report and the **Statistics** tab of the **Design Partitions Properties** dialog box. These reports are useful when optimizing your design partitions, or when compiling the completed top-level design in a team-based compilation flow to ensure that partitions meet the guidelines discussed in this manual.

The Partition Merge Partition Statistics report in the Partition Merge section of the Compilation report lists statistics about each partition. The statistics for each partition (each row in the table) include the number of logic cells, as well as the number of input and output pins and how many are registered. This report also lists how many ports are unconnected, or driven by a constant V_{CC} or GND. You can use this information to assess whether you have followed the guidelines for partition boundaries.

You can also view statistics about the resource and port connections for a particular partition on the **Statistics** tab of the **Design Partition Properties** dialog box. The **Show All Partitions** button allows you to view all the partitions in the same report. The Partition Merge Partition Statistics report also shows statistics for the **Internal Congestion: Total Connections and Registered Connections**. This information represents how many signals are connected within the partition. It then lists the inter-partition connections for each partition, which helps you to see how partitions are connected to each other.

2.6.5. Report Partition Timing in the Timing Analyzer

The Report Partitions diagnostic report and the `report_partitions` SDC command in the Timing Analyzer produce a **Partition Timing Overview** and **Partition Timing Details** table, which lists the partitions, the number of failing paths, and the worst case timing slack within each partition.

You can use these reports to analyze the location of the critical timing paths in the design in relation to partitions. If a certain partition contains many failing paths, or failing inter-partition paths, you might be able to change your partitioning scheme and improve timing performance.

Related Information

Quartus Prime Timing Analyzer documentation
Information about the Timing Analyzer `report_timing` command and reports

2.6.6. Check if Partition Assignments Impact the Quality of Results

You can ensure that you limit negative effect on the quality of results by following an iterative methodology during the partitioning process. In any incremental compilation flow where you can compile the source code for every partition during the partition planning phase, Altera recommends the following iterative flow:

1. Start with a complete design that is not partitioned and has no location or LogicLock region assignments.

   To run a full compilation, use the **Start Compilation** command.
2. Record the quality of results from the Compilation report (timing slack or f_{MAX}, area and any other relevant results).
3. Create design partitions following the guidelines described in this manual.

4. Recompile the design.
5. Record the quality of results from the Compilation report. If the quality of results is significantly worse than those obtained in the previous compilation, repeat step 3 through step 5 to change your partition assignments and use a different partitioning scheme.
6. Even if the quality of results is acceptable, you can repeat step 3 through step 5 by further dividing a large partition into several smaller partitions, which can improve compilation time in subsequent incremental compilations. You can repeat these steps until you achieve a good trade-off point (that is, all critical paths are localized within partitions, the quality of results is not negatively affected, and the size of each partition is reasonable).

You can also remove or disable partition assignments defined in the top-level design at any time during the design flow to compile the design as one flat compilation and get all possible design optimizations to assess the results. To disable the partitions without deleting the assignments, use the **Ignore partition assignments during compilation** option on the **Incremental Compilation** page of the **Settings** dialog box in the Quartus Prime software. This option disables all design partition assignments in your project and runs a full compilation, ignoring all partition boundaries and netlists. This option can be useful if you are using partitions to reduce compilation time as you develop various parts of the design, but can run a long compilation near the end of the design cycle to ensure the design meets its timing requirements.

2.7. Including SDC Constraints from Lower-Level Partitions for Third-Party IP Delivery

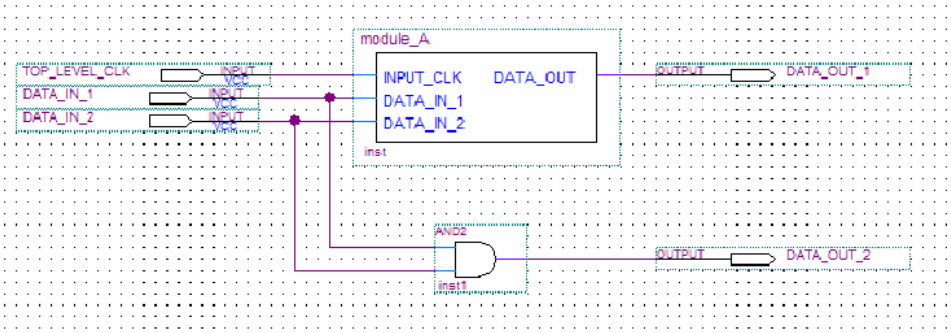
When exported partitions are compiled in a separate Quartus Prime project, such as when a third-party designer is delivering IP, the project lead must transfer the top-level project framework information and constraints to the partitions, so that each designer has a consistent view of the constraints that apply to the entire design. If the independent partition designers make any changes or add any constraints, they might have to transfer new constraints back to the project lead, so that these constraints are included in final timing sign-off of the entire design. Many assignments from the partition are carried with the partition into the top-level design; however, SDC format constraints for the Timing Analyzer are not copied into the top-level design automatically.

Passing additional timing constraints from a partition to the top-level design must be managed carefully. You can design within a single Quartus Prime project or a copy of the top-level design to simplify constraint management.

To ensure that there are no conflicts between the project lead's top-level constraints and those added by the third-party IP designer, use two `.sdc` files for each separate Quartus Prime project: an `.sdc` created by the project lead that includes project-wide constraints, and an `.sdc` created by the IP designer that includes partition-specific constraints.

The example design shown in the figure below is used to illustrate recommendations for managing the timing constraints in a third-party IP delivery flow. The top-level design instantiates a lower-level design block called `module_A` that is set as a design partition and developed by an IP designer in a separate Quartus Prime project.

Figure 29. Example Design to Illustrate SDC Constraints



In this top-level design, there is a single clock setting called `clk` associated with the FPGA input called `top_level_clk`. The top-level `.sdc` contains the following constraint for the clock:

```
create_clock -name {clk} -period 3.000 -waveform { 0.000 1.500 } \
[get_ports {TOP_LEVEL_CLK}]
```

2.7.1. Creating an .sdc File with Project-Wide Constraints

The `.sdc` with project-wide constraints for the separate Quartus Prime project should contain all constraints that are not completely localized to the partition. The `.sdc` should be maintained by the project lead. The project lead must ensure that these timing constraints are delivered to the individual partition owners and that they are syntactically correct for each of the separate Quartus Prime projects. This communication can be challenging when the design is in flux and hierarchies change. The project lead can use design partition scripts to automatically pass some of these constraints to the separate Quartus Prime projects.

The `.sdc` with project-wide constraints is used in the partition, but is not exported back to the top-level design. The partition designer should not modify this file. If changes are necessary, they should be communicated to the project lead, who can then update the SDC constraints and distribute new files to all partition designers as required.

The `.sdc` should include clock creation and clock constraints for any clock used by more than one partition. These constraints are particularly important when working with complex clocking structures, such as the following:

- Cascaded clock multiplexers
- Cascaded PLLs
- Multiple independent clocks on the same clock pin
- Redundant clocking structures required for secure applications
- Virtual clocks and generated clocks that are consistently used for source synchronous interfaces
- Clock uncertainties

Additionally, the .sdc with project-wide constraints should contain all project-wide timing exception assignments, such as the following:

- Multicycle assignments, set_multicycle_path
- False path assignments, set_false_path
- Maximum delay assignments, set_max_delay
- Minimum delay assignments, set_min_delay

The project-wide .sdc can also contain any set_input_delay or set_output_delay constraints that are used for ports in separate Quartus Prime projects, because these represent delays external to a given partition. If the partition designer wants to set these constraints within the separate Quartus Prime projects, the team must ensure that the I/O port names are identical in all projects so that the assignments can be integrated successfully without changes.

Similarly, a constraint on a path that crosses a partition boundary should be in the project-wide .sdc, because it is not completely localized in a separate Quartus Prime project.

2.7.1.1. Example Step 1—Project Lead Produces .sdc with Project-Wide Constraints for Lower-Level Partitions

The device input top_level_clk in Figure 29 on page 102 drives the input_clk port of module_A. To make sure the clock constraint is passed correctly to the partition, the project lead creates an .sdc with project-wide constraints for module_A that contains the following command:

```
create_clock -name {clk} -period 3.000 -waveform { 0.000 1.500 } [get_ports
{INPUT_CLK}]
```

The designer of module_A includes this .sdc as part of the separate Quartus Prime project.

2.7.2. Creating an .sdc with Partition-Specific Constraints

The .sdc with partition-specific constraints should contain all constraints that affect only the partition. For example, a set_false_path or set_multicycle_path constraint for a path entirely within the partition should be in the partition-specific .sdc. These constraints are required for correct compilation of the partition, but do not need to be present in any other separate Quartus Prime projects.

The partition-specific .sdc should be maintained by the partition designer; they must add any constraints required to properly compile and analyze their partition.

The partition-specific .sdc is used in the separate Quartus Prime project and must be exported back to the project lead for the top-level design. The project lead must use the partition-specific constraints to properly constrain the placement, routing, or both,

if the partition logic is fit at the top level, and to ensure that final timing sign-off is accurate. Use the following guidelines in the partition-specific `.sdc` to simplify these export and integration steps:

- Create a hierarchy variable for the partition (such as `module_A_hierarchy`) and set it to an empty string because the partition is the top-level instance in the separate Quartus Prime project. The project lead modifies this variable for the top-level hierarchy, reducing the effort of translating constraints on lower-level design hierarchies into constraints that apply in the top-level hierarchy. Use the following Tcl command first to check if the variable is already defined in the project, so that the top-level design does not use this empty hierarchy path: `if {![info exists module_A_hierarchy]}`.
- Use the hierarchy variable in the partition-specific `.sdc` as a prefix for assignments in the project. For example, instead of naming a particular instance of a register `reg:inst`, use `${module_A_hierarchy}reg:inst`. Also, use the hierarchy variable as a prefix to any wildcard characters (such as ” * ”).
- Pay attention to the location of the assignments to I/O ports of the partition. In most cases, these assignments should be specified in the `.sdc` with project-wide constraints, because the partition interface depends on the top-level design. If you want to set I/O constraints within the partition, the team must ensure that the I/O port names are identical in all projects so that the assignments can be integrated successfully without changes.
- Use caution with the `derive_clocks` and `derive_pll_clocks` commands. In most cases, the `.sdc` with project-wide constraints should call these commands. Because these commands impact the entire design, integrating them unexpectedly into the top-level design might cause problems.

If the design team follows these recommendations, the project lead should be able to include the `.sdc` with the partition-specific constraints provided by the partition designer directly in the top-level design.

2.7.2.1. Example Step 2—Partition Designer Creates .sdc with Partition-Specific Constraints

The partition designer compiles the design with the `.sdc` with project-wide constraints and might want to add some additional constraints. In this example, the designer realizes that he or she must specify a false path between the register called `reg_in_1` and all destinations in this design block with the wildcard character (such as ” * ”). This constraint applies entirely within the partition and must be exported to the top-level design, so it qualifies for inclusion in the `.sdc` with partition-specific constraints. The designer first defines the `module_A_hierarchy` variable and uses it when writing the constraint as follows:

```
if {![info exists module_A_hierarchy]} {
    set module_A_hierarchy ""
}
set_false_path -from [get_registers ${module_A_hierarchy}reg_in_1] \
-to [get_registers ${module_A_hierarchy}*]
```

2.7.3. Consolidating the .sdc in the Top-Level Design

When the partition designers complete their designs, they export the results to the project lead. The project lead receives the exported `.qxp` files and a copy of the `.sdc` with partition-specific constraints.

To set up the top-level .sdc constraint file to accept the .sdc files from the separate Quartus Prime projects, the top-level .sdc should define the hierarchy variables specified in the partition .sdc files. List the variable for each partition and set it to the hierarchy path, up to and including the instantiation of the partition in the top-level design, including the final hierarchy character "|".

To ensure that the .sdc files are used in the correct order, the project lead can use the Tcl Source command to load each .sdc.

2.7.3.1. Example Step 3—Project Lead Performs Final Timing Analysis and Sign-off

With these commands, the top-level .sdc file looks like the following example:

```
create_clock -name {clk} -period 3.000 -waveform { 0.000 1.500 } \
[get_ports {TOP_LEVEL_CLK}]
# Include the lower-level SDC file
set module_A_hierarchy "module_A:inst|"  # Note the final '|' character
source <partition-specific constraint file such as ..\module_A
\module_A_constraints>.sdc
```

When the project lead performs top-level timing analysis, the false path assignment from the lower-level module_A project expands to the following:

```
set_false_path -from module_A:inst|reg_in_1 -to module_A:inst|*
```

Adding the hierarchy path as a prefix to the SDC command makes the constraint legal in the top-level design, and ensures that the wildcard does not affect any nodes outside the partition that it was intended to target.

2.8. Introduction to Design Floorplans

A floorplan represents the layout of the physical resources on the device. Creating a design floorplan, or floorplanning, describes the process of mapping the logical design hierarchy onto physical regions in the device.

In the Quartus Prime software, LogicLock regions can be used to constrain blocks of a design to a particular region of the device. LogicLock regions represent an area on the device with a user-defined or Fitter-defined size and location in the device layout.

Related Information

Analyzing and Optimizing the Design Floorplan with the Chip Planner documentation

2.8.1. The Difference between Logical Partitions and Physical Regions

Design partitions are logical entities based on the design hierarchy. LogicLock regions are physical placement assignments that constrain logic to a particular region on the device.

A common misconception is that logic from a design partition is always grouped together on the device when you use incremental compilation. Actually, logic from a partition can be placed anywhere in the device if it is not constrained to a LogicLock region, although the Fitter can pack related logic together to improve timing performance. A logical design partition does not refer to any physical area on the device and does not directly control where instances are placed on the device.

If you want to control the placement of logic from a design partition and isolate it to a particular part of the device, you can assign the logical design partition to a physical region in the device floorplan with a LogicLock region assignment. Altera recommends creating a design floorplan by assigning design partitions to LogicLock regions to improve the quality of results and avoid placement conflicts in some situations for incremental compilation.

Another misconception is that LogicLock assignments are used to preserve placement results for incremental compilation. Actually, LogicLock regions only constrain logic to a physical region on the device. Incremental compilation does not use LogicLock assignments or any location assignments to preserve the placement results; it simply reuses the results stored in the database netlist from a previous compilation.

2.8.2. Why Create a Floorplan?

Creating a design floorplan is usually required if you want to preserve placement for partitions that will be exported, to avoid resource conflicts between partitions in the top-level design. Floorplan location planning can be important for a design that uses incremental compilation, for the following reasons:

- To avoid resource conflicts between partitions, predominantly when integrating partitions exported from another Quartus Prime project.
- To ensure good quality of results when recompiling individual timing-critical partitions.

Location assignments for each partition ensure that there are no placement conflicts between partitions. If there are no LogicLock region assignments, or if LogicLock regions are set to auto-size or floating location, no device resources are specifically allocated for the logic associated with the region. If you do not clearly define resource allocation, logic placement can conflict when you integrate the partitions in the top-level design if you reuse the placement information from the exported netlist.

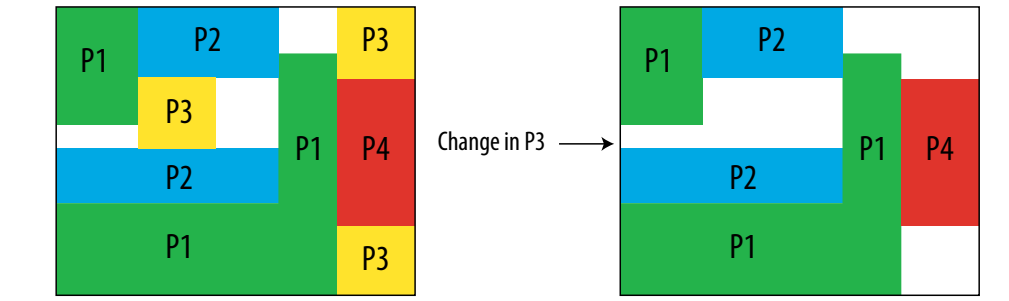
Creating a floorplan is also recommended for timing-critical partitions that have little timing margin to maintain good quality of results when the design changes.

Floorplan assignments are not required for non-critical partitions compiled in the same Quartus Prime project. The logic for partitions that are not timing-critical can be placed anywhere in the device on each recompilation if that is best for your design.

Design floorplan assignments prevent the situation in which the Fitter must place a partition in an area of the device where most resources are used by other partitions. A LogicLock region provides a reasonable region to re-place logic after a change, so the Fitter does not have to scatter logic throughout the available space in the device.

The figure illustrates the problems that may be associated with refitting designs that do not have floorplan location assignments. The left floorplan shows the initial placement of a four-partition design (P1-P4) without any floorplan location assignments. The right floorplan shows the device if a change occurs to P3. After removing the logic for the changed partition, the Fitter must re-place and reroute the new logic for P3 in the scattered white space. The placement of the post-fit netlists for other partitions forces the Fitter to implement P3 with the device resources that have not been used.

Figure 30. Representation of Device Floorplan without Location Assignments



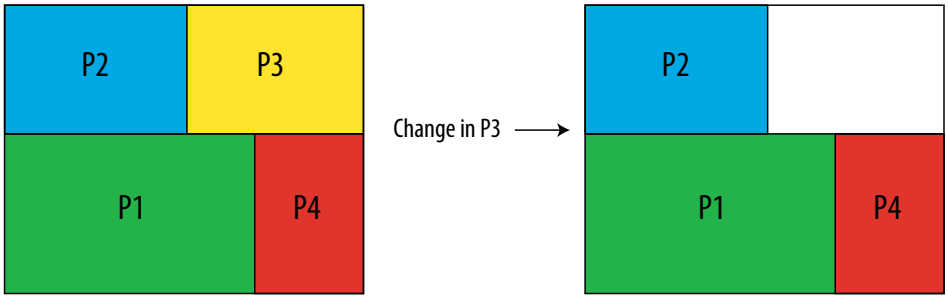
No floorplan assignments: Device has 4 partitions and the logic is placed throughout

Device after removing changed partition P3: New P3 must be placed in empty areas

The Fitter has a more difficult task because of more difficult physical constraints, and as a result, compilation time often increases. The Fitter might not be able to find any legal placement for the logic in partition P3, even if it could in the initial compilation. Additionally, if the Fitter can find a legal placement, the quality of results often decreases in these cases, sometimes dramatically, because the new partition is now scattered throughout the device.

The figure below shows the initial placement of a four-partition design with floorplan location assignments. Each partition is assigned to a LogicLock region. The second part of the figure shows the device after partition P3 is removed. This placement presents a much more reasonable task to the Fitter and yields better results.

Figure 31. Representation of Device Floorplan with Location Assignments



With floorplan location assignments: Device has 4 partitions placed in 4 LogicLock regions

Device after removing changed partition P3: Much easier to place new P3 partition in empty area

Altera recommends that you create a LogicLock floorplan assignment for timing-critical blocks with little timing margin that will be recompiled as you make changes to the design.

2.8.3. When to Create a Floorplan

It is important that you plan early to incorporate partitions into the design, and ensure that each partition follows partitioning guidelines. You can create floorplan assignments at different stages of the design flow, early or late in the flow. These guidelines help ensure better results as you begin creating floorplan location assignments.

2.8.3.1. Early Floorplan

An early floorplan is created before the design stage. You can plan an early floorplan at the top level of a design to allocate each partition a portion of the device resources. Doing so allows the designer for each block to create the logic for their design partition without conflicting with other logic. Each partition can be optimized in a separate Quartus Prime project if required, and the design can still be easily integrated in the top-level design. Even within one Quartus Prime project, each partition can be locked down with a post-fit netlist, and you can be sure there is space in the device floorplan for other partitions.

When you have compiled your complete design, or after you have integrated the first versions of partitions developed in separate Quartus Prime projects, you can use the design information and Quartus Prime features to tune and improve the floorplan .

2.8.3.2. Late Floorplan

A late floorplan is created or modified after the design is created, when the code is close to complete and the design structure is likely to remain stable. Creating a late floorplan is typically necessary only if you are starting to use incremental compilation late in the design flow, or need to reserve space for a logic block that becomes timing-critical but still has HDL changes to be integrated. When the design is complete, you can take advantage of the Quartus Prime analysis features to check the floorplan quality. To adjust the floorplan, you can perform iterative compilations as required and assess the results of different assignments.

Note: It may not be possible to create a good-quality late floorplan if you do not create partitions in the early stages of the design.

2.9. Design Floorplan Placement Guidelines

The following guidelines are key to creating a good design floorplan:

- Capture correct resources in each region.
- Use good region placement to maintain design performance compared to flat compilation.

A common misconception is that creating a floorplan enhances timing performance, as compared to a flat compilation with no location assignments. The Fitter does not usually require guidance to get optimal results for a full design.

Floorplan assignments can help maintain good performance when designs change incrementally. However, poor placement assignments in an incremental compilation can often adversely affect performance results, as compared to a flat compilation, because the assignments limit the options for the Fitter. Investing time to find good region placement is required to match the performance of a full flat compilation.

2.9.1. Flow for Creating a Floorplan

Use the following general procedure to create a floorplan:

1. Divide the design into partitions.
2. Assign the partitions to LogicLock regions.
3. Compile the design.
4. Analyze the results.
5. Modify the placement and size of regions, as required.

You might have to perform these steps several times to find the best combination of design partitions and LogicLock regions that meet the resource and timing goals of the design.

Related Information

Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design documentation on page 7

2.9.2. Assigning Partitions to LogicLock Regions

Before compiling a design with new LogicLock assignments, ensure that the partition netlist type is set to **Post-Synthesis** or **Source File**, so that the Fitter does not reuse previous placement results.

In most cases, you should include logic from one partition in each LogicLock region. This organization helps to prevent resource conflicts when partitions are exported and can lead to better performance preservation when locking down parts of a design in a single project.

The Quartus Prime software is flexible and allows exceptions to this rule. For example, you can place more than one partition in the same LogicLock region if the partitions are tightly connected, but you do not want to merge the partitions into one larger partition. For best results, ensure that you recompile all partitions in the LogicLock region every time the logic in one partition changes. Additionally, if a partition contains multiple lower-level entities, you can place those entities in different areas of the device with multiple LogicLock regions, even if they are defined in the same partition.

You can use the **Reserved** LogicLock option to ensure that you avoid conflicts with other logic that is not locked into a LogicLock region. This option prevents other logic from being placed in the region, and is useful if you have empty partitions at any point during your design flow, so that you can reserve space in the floorplan. Do not make reserved regions too large to prevent unused area because no other logic can be placed in a region with the **Reserved** LogicLock option.

Related Information

LogicLock Region Properties Dialog Box online help

2.9.3. How to Size and Place Regions

In an early floorplan, assign physical locations based on design specifications. Use information about the connections between partitions, the partition size, and the type of device resources required.

In a late floorplan, when the design is complete, you can use locations or regions chosen by the Fitter as a guideline. If you have compiled the full design, you can view the location of the partition logic in the Chip Planner. You can use the natural grouping

of each unconstrained partition as a starting point for a LogicLock region constraint. View the placement for each partition that requires a floorplan constraint, and create a new LogicLock region by drawing a box around the area on the floorplan, and then assigning the partition to the region to constrain the partition placement.

Instead of creating regions based on the previous compilation results, you can start with the Fitter results for a default auto size and floating origin location for each new region when the design logic is complete. After compilation, lock the size and origin location.

Alternatively, if the design logic is complete with auto-sized or floating location regions, you can specify the size based on the synthesis results and use the locations chosen by the Fitter with the **Set to Estimated Size** command. Like the previous option, start with floating origin location. After compilation, lock the origin location. You can also enable the **Fast Synthesis Effort** setting to reduce synthesis time.

After a compilation, save the Fitter size and origin location of the Fitter with the **Set Size and Origin to Previous Fitter Results** command.

Note: It is important that you use the Fitter-chosen locations only as a starting point to give the regions a good fixed size and location. Ensure that all LogicLock regions in the design have a fixed size and have their origin locked to a specific location on the device. On average, regions with fixed size and location yield better timing performance than auto-sized regions.

Related Information

Checking Partition Quality on page 96

2.9.4. Modifying Region Size and Origin

After saving the Fitter results from an initial compilation for a late floorplan, modify the regions using your knowledge of the design to set a specific size and location. If you have a good understanding of how the design fits together, you can often improve upon the regions placed in the initial compilation. In an early floorplan, when the design has not yet been created, you can use the guidelines in this section to set the size and origin, even though there is no initial Fitter placement.

The easiest way to move and resize regions is to drag the region location and borders in the Chip Planner. Make sure that you select the **User-Defined** region in the floorplan (as opposed to the **Fitter-Placed** region from the last compilation) so that you can change the region.

Generally, you can keep the Fitter-determined relative placement of the regions, but make adjustments if required to meet timing performance. Performing a full compilation ensures that the Fitter can optimize for a full placement and routing.

If two LogicLock regions have several connections between them, ensure they are placed near each other to improve timing performance. By placing connected regions near each other, the Fitter has more opportunity to optimize inter-region paths when both partitions are recompiled. Reducing the criticality of inter-region paths also allows the Fitter more flexibility when placing other logic in each region.

If resource utilization is low in the overall device, enlarge the regions. Doing so usually improves the final results because it gives the Fitter more freedom to place additional or modified logic added to the partition during subsequent incremental compilations. It also allows room for optimizations such as pipelining and logic duplication.

Try to have each region evenly full, with the same ”fullness” that the complete design would have without LogicLock regions; Altera recommends approximately 75% full.

Allow more area for regions that are densely populated, because overly congested regions can lead to poor results. Allow more empty space for timing-critical partitions to improve results. However, do not make regions too large for their logic. Regions that are too large can result in wasted resources and also lead to suboptimal results.

Ideally, almost the entire device should be covered by LogicLock regions if all partitions are assigned to regions.

Regions should not overlap in the device floorplan. If two partitions are allocated on an overlapping portion of the chip, each may independently claim common resources in this region. This leads to resource conflicts when integrating results into a top-level design. In a single project, overlapping regions give more difficult constraints to the Fitter and can lead to reduced quality of results.

You can create hierarchical LogicLock regions to ensure that the logic in a child partition is physically placed inside the LogicLock region for its parent partition. This can be useful when the parent partition does not contain registers at the boundary with the lower-level child partition and has a lot of signal connectivity. To create a hierarchical relationship between regions in the LogicLock Regions window, drag and drop the child region to the parent region.

2.9.5. I/O Connections

Consider I/O timing when placing regions. Using I/O registers can minimize I/O timing problems, and using boundary registers on partitions can minimize problems connecting regions or partitions. However, I/O timing might still be a concern. It is most important for flows where each partition is compiled independently, because the Fitter can optimize the placement for paths between partitions if the partitions are compiled at the same time.

Place regions close to the appropriate I/O, if necessary. For example, DDR memory interfaces have very strict placement rules to meet timing requirements. Incorporate any specific placement requirements into your floorplan as required. You should create LogicLock regions for internal logic only, and provide pin location assignments for external device I/O pins (instead of including the I/O cells in a LogicLock region to control placement).

2.9.6. LogicLock Resource Exclusions

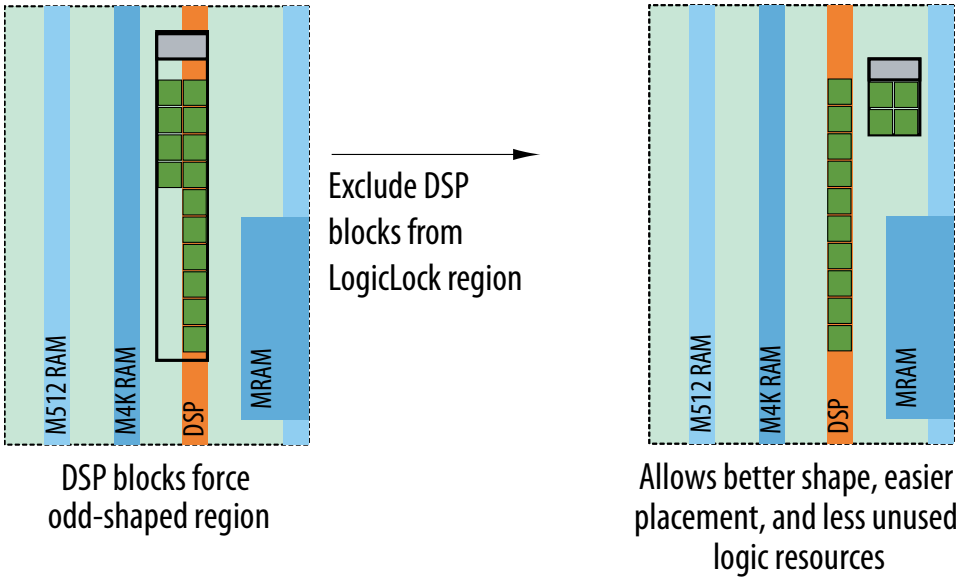
You can exclude certain resource types from a LogicLock region to manage the ratio of logic to dedicated DSP and RAM resources in the region.

If your design contains memory or Digital Signal Processing (DSP) elements, you may want to exclude these elements from the LogicLock region. LogicLock resource exceptions prevent certain types of elements from being assigned to a region. Therefore, those elements are not required to be placed inside the region boundaries. The option does not prevent them from being placed inside the region boundaries unless the **Reserved** property of the region is turned on.

Resource exceptions are useful in cases where it is difficult to place rectangular regions for design blocks that contain memory and DSP elements, due to their placement in columns throughout the device floorplan. Exclude RAMs, DSPs, or logic cells to give the Fitter more flexibility with region sizing and placement. Excluding RAM

or DSP elements can help to resolve no-fit errors that are caused by regions spanning too many resources, especially for designs that are memory-intensive, DSP-intensive, or both. The figure shows an example of a design with an odd-shaped region to accommodate DSP blocks for a region that does not contain very much logic. The right side of the figure shows the result after excluding DSP blocks from the region. The region can be placed more easily without wasting logic resources.

Figure 32. LogicLock Resource Exclusion Example



To view any resource exceptions, right-click in the LogicLock Regions window, and then click **LogicLock Regions Properties**. In the **LogicLock Regions Properties** dialog box, select the design element (module or entity) in the **Members** box, and then click **Edit**. In the **Edit Node** dialog box, to set up a resource exception, click the **Edit** button next to the **Excluded element types** box, and then turn on the design element types to be excluded from the region. You can choose to exclude combinational logic or registers from logic cells, or any of the sizes of TriMatrix memory blocks, or DSP blocks.

If the excluded logic is in its own lower-level design entity (even if it is within the same design partition), you can assign the entity to a separate LogicLock region to constrain its placement in the device.

You can also use this feature with the LogicLock **Reserved** property to reserve specific resources for logic that will be added to the design.

2.9.6.1. Creating Floorplan Location Assignments With Tcl Commands—Excluding or Filtering Certain Device Elements (Such as RAM or DSP Blocks)

To assign a code block to a LogicLock region, with exclusions, use the following command:

```
set_logiclock_contents -region <LogicLock region name> \
-to <block> -exceptions \"<keyword>:<keyword>"
```

- *<LogicLock region name>*—The name of the LogicLock region to which the code block is assigned.
- *<block>*—A code block in a Quartus Prime project hierarchy, which can also be a design partition.
- *<keyword>*—The list of exceptions made during assignment. For example, if DSP was in the keyword list, the named block of code would be assigned to the LogicLock region, except for any DSP block within the code block. You can include the following exceptions in the `set_logiclock_contents` command:

Keyword variables:

- *REGISTER*—Any registers in the logic cells.
- *COMBINATIONAL*—Any combinational elements in the logic cells.
- *SMALL_MEM*—Small TriMatrix memory blocks (M512 or MLAB).
- *MEDIUMEM_MEM*—Medium TriMatrix memory blocks (M4K or M9K).
- *LARGE_MEM*—Large TriMatrix memory blocks (M-RAM or M144K).
- *DSP*—Any DSP blocks.
- *VIRTUAL_PIN*—Any virtual pins.

Note: Resource filtering uses the optional Tcl argument `-exclude_resources` in the `set_logiclock_contents` function. If left unspecified, no resource filter is created. In the `.qsf`, resource filtering uses an extra LogicLock membership assignment called `LL_MEMBER_RESOURCE_EXCLUDE`. For example, the following line in the `.qsf` is used to specify a resource filter for the `alu:alu_unit` entity assigned to the ALU region.

```
set_instance_assignment -name LL_MEMBER_RESOURCE_EXCLUDE \
"DSP:SMALL_MEM" -to "alu:alu_unit" -section_id ALU
```

2.9.7. Creating Non-Rectangular Regions

To constrain placement to non-rectangular or non-contiguous areas of the device, you can connect multiple rectangular regions together using the **Merge** command.

For devices that do not support the **Merge** command (MAX™ II devices), you can limit entity placement to a sub-area of a LogicLock region to create non-rectangular constraints. In these devices, construct a LogicLock hierarchy by creating child regions inside of parent regions, and then use the **Reserved** option to control which logic can be placed inside these child regions. Setting the **Reserved** option for the region prevents the Fitter from placing nodes that are not assigned to the region inside the boundary of the region.

2.10. Checking Floorplan Quality

The Quartus Prime software has several tools to help you create a floorplan. You can use these tools to assess your floorplan quality and use the information to improve your design or assignments as required to achieve the best results.

2.10.1. Incremental Compilation Advisor

You can use the Incremental Compilation Advisor to check that your design follows the recommendations for creating floorplan location assignments that are presented in this manual.

2.10.2. LogicLock Region Resource Estimates

You can view resource estimates for a LogicLock region to determine the region's resource coverage, and use this estimate before compilation to check region size. Using this estimate helps to ensure adequate resources when you are sizing or moving regions.

2.10.3. LogicLock Region Properties Statistics Report

LogicLock region statistics are similar to design partition properties, but also include resource usage details after compilation.

The statistics report the number of resources used and the total resources covered by the region, and also lists the number of I/O connections and how many I/Os are registered (good), as well as the number of internal connections and the number of inter-region connections (bad).

2.10.4. Locate the Quartus Prime Timing Analyzer Path in the Chip Planner

In the Timing Analyzer user interface, you can locate a specific path in the Chip Planner to view its placement and perform a report timing operation (for example, report timing for all paths with less than 0 ns slack).

2.10.5. Inter-Region Connection Bundles

The Chip Planner can display bundles of connections between LogicLock regions, with filtering options that allow you to choose the relevant data for display. These bundles can help you to visualize how many connections there are between each LogicLock region to improve floorplan assignments or to change partition assignments, if required.

2.10.6. Routing Utilization

The Chip Planner includes a feature to display a color map of routing congestion. This display helps identify areas of the chip that are too tightly packed.

In the Chip Planner, red LAB blocks indicate higher routing congestion. You can position the mouse pointer over a LAB to display a tooltip that reports the logic and routing utilization information.

2.10.7. Ensure Floorplan Assignments Do Not Significantly Impact Quality of Results

The end results of design partitioning and floorplan creation differ from design to design. However, it is important to evaluate your results to ensure that your scheme is successful. Compare your before and after results, and consider using another scheme if any of the following guidelines are not met:

- You should see only minor degradation in f_{MAX} after the design is partitioned and floorplan location assignments are created. There is some performance cost associated with setting up a design for incremental compilation; approximately 3% is typical.
- The area increase should be no more than 5% after the design is partitioned and floorplan location assignments are created.
- The time spent in the routing stage should not significantly increase.

The amount of compilation time spent in the routing stage is reported in the Messages window with an Info message that indicates the elapsed time for Fitter routing operations. If you notice a dramatic increase in routing time, the floorplan location assignments may be creating substantial routing congestion. In this case, decrease the number of LogicLock regions, which typically reduces the compilation time in subsequent incremental compilations and may also improve design performance.

2.11. Recommended Design Flows and Application Examples

Listed below are application examples with design flows for partitioning and creating a design floorplan during common timing closure and team-based design scenarios. Each flow describes the situation in which it should be used, and provides a step-by-step description of the commands required to implement the flow.

2.11.1. Create a Floorplan for Major Design Blocks

Use this incremental compilation flow for designs when you want to assign a floorplan location for each major block in your design. A full floorplan ensures that partitions do not interact as they are changed and recompiled— each partition has its own area of the device floorplan.

To create a floorplan for major design blocks, follow this general methodology:

1. In the Design Partitions window, ensure that all partitions have their netlist type set to **Source File** or **Post-Synthesis**. If the netlist type is set to **Post-Fit**, floorplan location assignments are not used when recompiling the design.
2. Create a LogicLock region for each partition (including the top-level entity, which is set as a partition by default).
3. Run a full compilation of your design to view the initial Fitter-chosen placement of the LogicLock regions as a guideline.
4. In the Chip Planner, view the placement results of each partition and LogicLock region on the device.
5. If required, modify the size and location of the LogicLock regions in the Chip Planner. For example, enlarge the regions to fill up the device and allow for future logic changes.You can also, if needed, create a new LogicLock region by drawing a box around an area on the floorplan.
6. Run the Compiler with the **Start Compilation** command to determine the timing performance of your design with the modified or new LogicLock regions.
7. Repeat steps 5 and 6 until you are satisfied with the quality of results for your design floorplan. Once you are satisfied with your results, run a full compilation of your design.

2.11.2. Create a Floorplan Assignment for One Design Block with Difficult Timing

Use this flow when you have one timing-critical design block that requires more optimization than the rest of your design. You can take advantage of incremental compilation to reduce your compilation time without creating a full design floorplan.

In this scenario, you do not want to create floorplan assignments for the entire design. Instead, you can create a region to constrain the location of your critical design block, and allow the rest of the logic to be placed anywhere on the device. To create a region for critical design block, follow these steps:

1. Divide up your design into partitions. Ensure that you isolate the timing-critical logic in a separate partition.
2. Define a LogicLock region for the timing-critical partition. Ensure that you capture the correct amount of device resources in the region. Turn on the **Reserved** property to prevent any other logic from being placed in the region.
   - If the design block is not complete, reserve space in the design floorplan based on your knowledge of the design specifications, connectivity between design blocks, and estimates of the size of the partition based on any initial implementation numbers.
   - If the critical design block has initial source code ready, compile the design to place the LogicLock region. Save the Fitter-determined size and origin, and then enlarge the region to provide more flexibility and allow for future design changes.

   As the rest of the design is completed, and the device fills up, the timing-critical region reserves an area of the floorplan. When you make changes to the design block, the logic will be re-placed in the same part of the device, which helps ensure good quality of results.

Related Information

Design Partition Guidelines on page 79

2.11.3. Create a Floorplan as the Project Lead in a Team-Based Flow

Use this approach when you have several designs that will be implemented in separate Quartus Prime projects by different designers, or third-party IP designers who want to optimize their designs independently and pass the results to the project lead.

As the project lead in this scenario, follow these steps to prepare the top-level design for a successful team-based design methodology with early floorplan planning:

1. Create a new Quartus Prime project that will ultimately contain the full implementation of the entire design.
2. Create a "skeleton" or framework of the design that defines the hierarchy for the subdesigns that will be implemented by separate designers. Consider the partitioning guidelines in this manual when determining the design hierarchy.
3. Make project-wide settings. Select the device, make global assignments for clocks and device I/O ports, and make any global signal constraints to specify which signals can use global routing resources.

4. Make design partition assignments for each major subdesign. Set the netlist type for each partition that will be implemented in a separate Quartus Prime project and later exported and integrated with the top-level design set to **Empty**.
5. Create LogicLock regions for each partition to create a design floorplan. This floorplan should consider the connectivity between partitions and estimates of the size of each partition based on any initial implementation numbers and knowledge of the design specifications. Use the guidelines described in this chapter to choose a size and location for each LogicLock region.
6. Provide the constraints from the top-level design to partition designers using one of the following procedures:
   a. Create a copy of the top-level Quartus Prime project framework by checking out the appropriate files from a source control system, using the **Copy Project** command, or creating a project archive. Provide each partition designer with the copy of the project.
   b. Provide the constraints with documentation or scripts.

2.12. Document Revision History

Table 7. Document Revision History

Date	Version	Changes
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	Removed support for early timing estimate feature.
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings. • Updated description of Virtual Pin assingment to clarify that assigned pins are no longer free as input pins.
June 2014	14.0.0	• Dita conversion. • Removed obsolete devices content for Arria GX, Cyclone, Cyclone II, Cyclone III, Stratix, Stratix GX, Stratix II, Stratix II GX, • Replace Megafunction content with IP Catalog and Parameter Editor content.
November 2013	13.1.0	Removed HardCopy device information.
November 2012	12.1.0	Added Turning On Supported Cross-Boundary Optimizations.
June 2012	12.0.0	Removed survey link.
November 2011	11.0.1	Template update.
May 2011	11.0.0	Updated links.
December 2010	10.1.0	• Changed to new document template. • Moved "Creating Floorplan Location Assignments With Tcl Commands—Excluding or Filtering Certain Device Elements (Such as RAM or DSP Blocks)" from the Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design chapter in volume 1 of the *Quartus Prime Handbook.* • Consolidated Design Partition Planner and Incremental Compilation Advisor information between the Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design and Best Practices for Incremental Compilation Partitions and Floorplan Assignments handbook chapters.

continued...

Date	Version	Changes
July 2010	10.0.0	• Removed the explanation of the "bottom-up design flow" where designers work completely independently, and replaced with Altera's recommendations for team-based environments where partitions are developed in the same top-level project framework, plus an explanation of the bottom-up process for including independent partitions from third-party IP designers. • Expanded the **Merge** command explanation to explain how it now accommodates cross-partition boundary optimizations. • Restructured Altera recommendations for when to use a floorplan.
October 2009	9.1.0	• Redefined the bottom-up design flow as team-based and reorganized previous design flow examples to include steps on how to pass top-level design information to lower-level projects. • Added "Including SDC Constraints from Lower-Level Partitions for Third-Party IP Delivery" from the Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design chapter in volume 1 of the *Quartus Prime Handbook.* • Reorganized the "Recommended Design Flows and Application Examples" section. • Removed HardCopy APEX and HardCopy Stratix Devices section.
March 2009	9.0.0	• Added I/O register packing examples from *Incremental Compilation for Hierarchical and Team-Based Designs* chapter • Moved "Incremental Compilation Advisor" section • Added "Viewing Design Partition Planner and Floorplan Side-by-Side" section • Updated Figure 15-22 • Chapter 8 was previously Chapter 7 in software release 8.1.
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size. No change to content.
May 2007	8.0.0	• Initial release.

Related Information

Documentation Archive

For previous versions of the *Quartus Prime Handbook*, search the documentation archives.

3. Quartus Prime Integrated Synthesis

As programmable logic designs become more complex and require increased performance, advanced synthesis becomes an important part of a design flow. The Altera® Quartus® II software includes advanced Integrated Synthesis that fully supports VHDL, Verilog HDL, and Altera-specific design entry languages, and provides options to control the synthesis process. With this synthesis support, the Quartus Prime software provides a complete, easy-to-use solution.

Related Information

Designing With Low-Level Primitives User Guide
For more information about coding with primitives that describe specific low-level functions in Altera devices

3.1. Design Flow

The Quartus Prime Analysis & Synthesis stage of the compilation flow runs Integrated Synthesis, which fully supports Verilog HDL, VHDL, and Altera-specific languages, and major features of the SystemVerilog language.

In the synthesis stage of the compilation flow, the Quartus Prime software performs logic synthesis to optimize design logic and performs technology mapping to implement the design logic in device resources such as logic elements (LEs) or adaptive logic modules (ALMs), and other dedicated logic blocks. The synthesis stage generates a single project database that integrates all your design files in a project (including any netlists from third-party synthesis tools).

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

You can use Analysis & Synthesis to perform the following compilation processes:

Table 8. Compilation Process

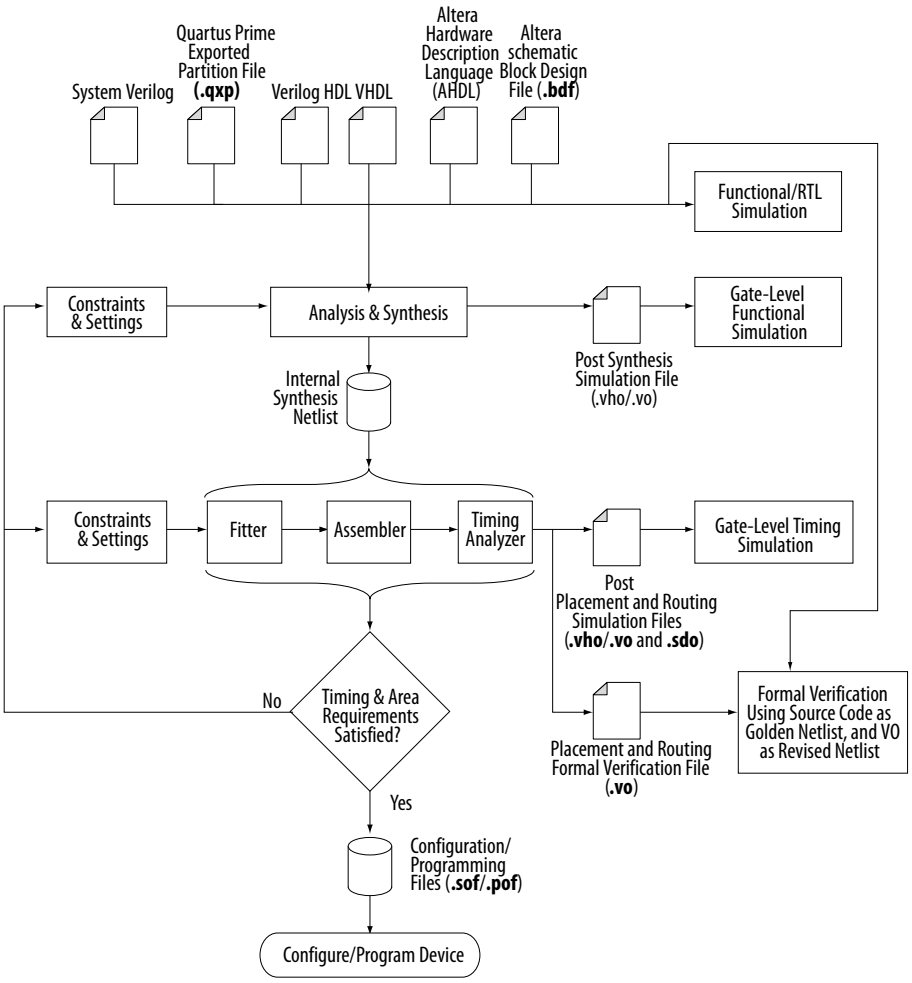
Compilation Process	Description
Analyze Current File	Parses your current design source file to check for syntax errors. This command does not report many semantic errors that require further design synthesis. To perform this analysis, on the Processing menu, click **Analyze Current File**.
Analysis & Elaboration	Checks your design for syntax and semantic errors and performs elaboration to identify your design hierarchy. To perform Analysis & Elaboration, on the Processing menu, point to **Start,** and then click **Start Analysis & Elaboration**.
Hierarchy Elaboration	Parses HDL designs and generates a skeleton of hierarchies. Hierarchy Elaboration is similar to the Analysis & Elaboration flow, but without any elaborated logic, thus making it much faster to generate.
Analysis & Synthesis	Performs complete Analysis & Synthesis on a design, including technology mapping. To perform Analysis & Synthesis, on the Processing menu, point to **Start,** and then click **Start Analysis & Synthesis**.

Related Information

Language Support on page 122

3.1.1. Quartus Prime Integrated Synthesis Design and Compilation Flow

Figure 33. Basic Design Flow Using Quartus Prime Integrated Synthesis



The Quartus Prime Integrated Synthesis design and compilation flow consists of the following steps:

1. Create a project in the Quartus Prime software and specify the general project information, including the top-level design entity name.
2. Create design files in the Quartus Prime software or with a text editor.
3. On the Project menu, click **Add/Remove Files in Project** and add all design files to your Quartus Prime project using the **Files** page of the **Settings** dialog box.
4. Specify Compiler settings that control the compilation and optimization of your design during synthesis and fitting.

5. Add timing constraints to specify the timing requirements.
6. Compile your design. To synthesize your design, on the Processing menu, point to **Start**, and then click **Start Analysis & Synthesis**. To run a complete compilation flow including placement, routing, creation of a programming file, and timing analysis, click **Start Compilation** on the Processing menu.
7. After obtaining synthesis and placement and routing results that meet your requirements, program or configure your Altera device.

Integrated Synthesis generates netlists that enable you to perform functional simulation or gate-level timing simulation, timing analysis, and formal verification.

Related Information

- Quartus Prime Synthesis Options on page 138
  For more information about synthesis settings
- Incremental Compilation on page 136
  For more information about partitioning your design to reduce compilation time
- Quartus Prime Exported Partition File as Source on page 138
  For more information about using **.qxp** as a design source file
- Introduction to the Quartus Prime Software
  For an overall summary of features in the Quartus Prime software

3.1.1.1. 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.

3.2. Language Support

Quartus Prime Integrated Synthesis supports HDL. You can specify the Verilog HDL or VHDL language version in your design.

To ensure that the Quartus Prime software reads all associated project files, add each file to your Quartus Prime project by clicking **Add/Remove Files in Project** on the Project menu. You can add design files to your project. You can mix all supported languages and netlists generated by third-party synthesis tools in a single Quartus Prime project.

Related Information

- Design Libraries on page 129
  Describes how to compile and reference design units in custom libraries
- Using Parameters/Generics on page 132
  Describes how to use parameters or generics and pass them between languages

3.2.1. 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)

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.

3.2.1.1. 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.

3.2.1.1.1. Configuration Syntax

A Verilog HDL configuration contains the following statements:

```
config config_identifier;
design [library_identifier.]cell_identifier;
config_rule_statement;
endconfig
```

- `config`—the keyword that begins the configuration.
- `config_identifier`—the name you enter for the configuration.
- `design`—the keyword that starts a design statement for specifying the top of the design.
- `[library_identifier.]cell_identifier`—specifies the top-level module (or top-level modules) in the design and the logical library for this module (modules).
- `config_rule_statement`—one or more of the following clauses: default, instance, or cell. For more information, refer to Table 9 on page 124.
- `endconfig`—the keyword that ends a configuration.

Table 9. Type of Clauses for the config_rule_statement Keyword

Clause Type	Description
`default`	Specifies the logical libraries to search to resolve a default cell instance. A default cell instance is an instance in the design that is not specified in a subsequent instance or cell clause in the configuration. You specify these libraries with the `liblist` keyword. The following is an example of a default clause: `default liblist lib1 lib2;` Also specifies resolving default instances in the logical libraries (`lib1` and `lib2`). Because libraries are inherited, some simulators (for example, VCS) also search the default (or current) library as well after the searching the logical libraries (`lib1` and `lib2`).

`instance`	Specifies a specific instance. The specified instance clause depends on the use of the following keywords:
	— `liblist`—specifies the logical libraries to search to resolve the instance.
	— `use`—specifies that the instance is an instance of the specified cell in the specified logical library.
	The following are examples of `instance` clauses:
	`instance top.dev1 liblist lib1 lib2;`
	This instance clause specifies to resolve `instance top.dev1` with the cells assigned to logical libraries `lib1` and `lib2`;
	`instance top.dev1.gm1 use lib2.gizmult;`
	This instance clause specifies that `top.dev1.gm1` is an instance of the cell named `gizmult` in logical library `lib2`.

`cell`	A `cell` clause is similar to an `instance` clause, except that the `cell` clause specifies all instances of a cell definition instead of specifying a particular instance. What it specifies depends on the use of the `liblist` or `use` keywords: — `liblist`—specifies the logical libraries to search to resolve all instances of the cell. — `use`—the specified cell's definition is in the specified library.

3.2.1.1.2. 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 12. 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 13. 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.

3.2.1.1.3. Suffix :config

To distinguish between a module by the same name, use the optional extension `:config` to refer to configuration names. For example, you can always refer to a `cfg2` configuration as `cfg2:config` (even if the `cfg2` module does not exist).

3.2.1.2. SystemVerilog Support

The Quartus Prime software supports the SystemVerilog constructs.

Note: Designs written to support the Verilog-2001 standard might not compile with the SystemVerilog setting because the SystemVerilog standard has several new reserved keywords.

3.2.1.3. 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 14. 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 15. Text File Format: Initializing RAM with the readmemb Command

```
@0
00000000
@1
00000001
@2
00000010
…
@e
00001110
@f
00001111
```

3.2.1.4. 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
```

3.2.1.4.1. Setting a Verilog HDL Macro Default Value in the Quartus Prime Software

To specify a macro in the Quartus Prime software, follow these steps:

1. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Verilog HDL Input**
2. Under **Verilog HDL macro**, type the macro name in the **Name** box and the value in the **Setting** box.
3. Click **Add**.

3.2.1.4.2. Setting a Verilog HDL Macro Default Value on the Command Line

To set a default value for a Verilog HDL macro on the command line, use the `--verilog_macro` option:

```
quartus_map <Design name> --verilog_macro= "<Macro name>=<Macro setting>"
```

The command in this example has the same effect as specifying `` `define a 2 `` in the Verilog HDL source code:

```
quartus_map my_design --verilog_macro="a=2"
```

To specify multiple macros, you can repeat the option more than once.

```
quartus_map my_design --verilog_macro="a=2" --verilog_macro="b=3"
```

3.2.2. 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)

The Quartus Prime Compiler uses the VHDL 1993 standard by default for files that have the extension `.vhdl` or `.vhd`.

Note: The VHDL code samples follow the VHDL 1993 standard.

3.2.2.1. Quartus Prime Support for VHDL 2008

The Quartus Prime software contains support for VHDL 2008 with the following constructs defined in the IEEE Standard 1076-2008 version of the *IEEE Standard for VHDL Language Reference Manual*:

- Section 5.3.2—Unconstrained elements in arrays
- Section 9.2.3—Matching equality/inequality operators
- Section 9.2.9—Condition operator
- Section 10.9—Matching case statement
- Section 11.3—Simplified sensitivity lists
- Section 11.8—Extensions to the `generate` statement (`elsif` and `else` constructs in `if generate` statements and support for `case generate` statements)
- Section 15.8—Enhanced Bit-string literals
- Section 15.9—Block comments

The Quartus Prime software provides a library of HDL design templates, including VHDL 2008 constructs, that you can access through the **Insert Template** dialog box.

3.2.2.2. 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.

3.2.2.3. 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 16. 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;
```

3.2.3. AHDL Support

The Quartus Prime Compiler's Analysis & Synthesis module fully supports the Altera Hardware Description Language (AHDL).

AHDL designs use Text Design Files (**.tdf**). You can import AHDL Include Files (**.inc**) into a **.tdf** with an AHDL `include` statement. Altera provides **.inc** files for all IP cores shipped with the Quartus Prime software.

Note: The AHDL language does not support the synthesis directives or attributes.

3.2.4. Schematic Design Entry Support

The Quartus Prime Compiler's Analysis & Synthesis module fully supports **.bdf** for schematic design entry.

Note: Schematic entry methods do not support the synthesis directives or attributes.

3.2.5. State Machine Editor

The Quartus Prime Pro Edition software supports graphical state machine entry. To create a new finite state machine (FSM) design:

1. Click **File ➤ New**.
2. In the **New** dialog box, expand the **Design Files** list, and then select **State Machine File**.

3.2.6. 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.

Note: If you do not specify a design library, if a file refers to a library that does not exist, or if the referenced library does not contain a referenced design unit, the Quartus Prime software searches the work library. This behavior allows the Quartus Prime software to compile most designs with minimal setup, but you have the option of creating separate custom design libraries.

Related Information

Mapping a VHDL Instance to an Entity in a Specific Library on page 131

3.2.6.1. Specifying a Destination Library Name in the Settings Dialog Box

To specify a library name for one of your design files, follow these steps:

1. On the Assignments menu, click **Settings**.
2. In the **Category** list, select **Files**.
3. Select the file in the **File Name** list.
4. Click **Properties**.

5. In the **File Properties** dialog box, select the type of design file from the **Type** list.
6. Type the library name in the **Library** field.
7. Click **OK**.

3.2.6.2. Specifying a Destination Library Name in the Quartus Prime Settings File or with Tcl

You can specify the library name with the `-library` option to the <language type>_FILE assignment in the Quartus Prime Settings File (**.qsf**) or with Tcl commands.

For example, the following assignments specify that the Quartus Prime software analyzes the **my_file.vhd** and stores its contents (design units) in the VHDL library **my_lib**, and then analyzes the Verilog HDL file **my_header_file.h** and stores its contents in a library called **another_lib**.

```
set_global_assignment –name VHDL_FILE my_file.vhd –library my_lib
set_global_assignment –name VERILOG_FILE my_header_file.h –library another_lib
```

Related Information

Scripting Support on page 189
For more information about Tcl scripting

3.2.6.3. Specifying a Destination Library Name in a VHDL File

You can use the `library` synthesis directive to specify a library name in your VHDL source file. This directive takes the name of the destination library as a single string argument. Specify the `library` directive in a VHDL comment before the context clause for a primary design unit (that is, a package declaration, an entity declaration, or a configuration), with one of the supported keywords for synthesis directives, that is, `altera`, `synthesis`, `pragma`, `synopsys`, or `exemplar`.

The `library` directive overrides the default library destination **work**, the library setting specified for the current file in the **Settings** dialog box, any existing **.qsf** setting, any setting made through the Tcl interface, or any prior `library` directive in the current file. The directive remains effective until the end of the file or the next `library` synthesis directive.

The following example uses the `library` synthesis directive to create a library called **my_lib** containing the `my_entity` design unit:

```
-- synthesis library my_lib
library ieee;
use ieee.std_logic_1164.all;
entity my_entity(...)
end entity my_entity;
```

Note: You can specify a single destination library for all your design units in a given source file by specifying the library name in the **Settings** dialog box, editing the **.qsf**, or using the Tcl interface. To organize your design units in a single file into different libraries rather than just a single library, you can use the `library` directive to change the destination VHDL library in a source file.

The Quartus Prime software generates an error if you use the library directive in a design unit.

Related Information

Synthesis Directives on page 141
For more information about specifying synthesis directives

3.2.6.4. Mapping a VHDL Instance to an Entity in a Specific Library

The VHDL language provides several ways to map or bind an instance to an entity in a specific library.

3.2.6.4.1. Direct Entity Instantiation

In the direct entity instantiation method, the instantiation refers to an entity in a specific library.

The following shows the direct entity instantiation method for VHDL:

```
entity entity1 is
port(...);
end entity entity1;
architecture arch of entity1 is
begin
inst: entity lib1.foo
port map(...);
end architecture arch;
```

3.2.6.4.2. Component Instantiation—Explicit Binding Instantiation

You can bind a component to an entity in several mechanisms. In an explicit binding indication, you bind a component instance to a specific entity.

The following shows the binding instantiation method for VHDL:

```
entity entity1 is
port(...);
end entity entity1;
package components is
component entity1 is
port map (...);
end component entity1;
end package components;
entity top_entity is
port(...);
end entity top_entity;
use lib1.components.all;
architecture arch of top_entity is
-- Explicitly bind instance I1 to entity1 from lib1
for I1: entity1 use entity lib1.entity1
port map(...);
end for;
begin
I1: entity1 port map(...);
end architecture arch;
```

3.2.6.4.3. Component Instantiation—Default Binding

If you do not provide an explicit binding indication, the Quartus Prime software binds a component instance to the nearest visible entity with the same name. If no such entity is visible in the current scope, the Quartus Prime software binds the instance to

the entity in the library in which you declare the component. For example, if you declare the component in a package in the MY_LIB library, an instance of the component binds to the entity in the MY_LIB library.

The code examples in the following examples show this instantiation method:

Example 17. VHDL Code: Default Binding to the Entity in the Same Library as the Component Declaration

```
use mylib.pkg.foo; -- import component declaration from package “pkg” in

                           -- library “mylib”
architecture rtl of top
...
begin
-- This instance will be bound to entity “foo” in library “mylib”
inst: foo
port map(...);
end architecture rtl;
```

Example 18. VHDL Code: Default Binding to the Directly Visible Entity

```
use mylib.foo; -- make entity “foo” in library “mylib” directly visible
architecture rtl of top
component foo is
generic (...)
port (...);
end component;
begin
-- This instance will be bound to entity “foo” in library “mylib”
inst: foo
port map(...);
end architecture rtl;
```

3.2.7. Using Parameters/Generics

The Quartus Prime software supports parameters (known as generics in VHDL) and you can pass these parameters between design languages.

Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Default Parameters** to enter default parameter values for your design. In AHDL, the Quartus Prime software inherits parameters, so any default parameters apply to all AHDL instances in your design. You can also specify parameters for instantiated modules in a **.bdf**. To specify parameters in a **.bdf** instance, double-click the parameter value box for the instance symbol, or right-click the symbol and click **Properties**, and then click the **Parameters** tab.

You can specify parameters for instantiated modules in your design source files with the provided syntax for your chosen language. Some designs instantiate entities in a different language; for example, they might instantiate a VHDL entity from a Verilog HDL design file. You can pass parameters or generics between VHDL, Verilog HDL, AHDL, and BDF schematic entry, and from EDIF or VQM to any of these languages. You do not require an additional procedure to pass parameters from one language to another. However, sometimes you must specify the type of parameter you are passing. In those cases, you must follow certain guidelines to ensure that the Quartus Prime software correctly interprets the parameter value.

Related Information

- Setting Default Parameter Values and BDF Instance Parameter Values on page 133
  For more information about the GUI-based entry methods, the interpretation of parameter values, and format recommendations
- Passing Parameters Between Two Design Languages on page 134
  For more information about parameter type rules

3.2.7.1. Setting Default Parameter Values and BDF Instance Parameter Values

Default parameter values and BDF instance parameter values do not have an explicitly declared type. Usually, the Quartus Prime software can correctly infer the type from the value without ambiguity. For example, the Quartus Prime software interprets "ABC" as a string, 123 as an integer, and 15.4 as a floating-point value. In other cases, such as when the instantiated subdesign language is VHDL, the Quartus Prime software uses the type of the parameter, generic, or both in the instantiated entity to determine how to interpret the value, so that the Quartus Prime software interprets a value of 123 as a string if the VHDL parameter is of a type string. In addition, you can set the parameter value in a format that is legal in the language of the instantiated entity. For example, to pass an unsized bit literal value from **.bdf** to Verilog HDL, you can use `'1` as the parameter value, and to pass a 4-bit binary vector from **.bdf** to Verilog HDL, you can use `4'b1111` as the parameter value.

In a few cases, the Quartus Prime software cannot infer the correct type of parameter value. To avoid ambiguity, specify the parameter value in a type-encoded format in which the first or first and second characters of the parameter indicate the type of the parameter, and the rest of the string indicates the value in a quoted sub-string. For example, to pass a binary string `1001` from **.bdf** to Verilog HDL, you cannot use the value `1001`, because the Quartus Prime software interprets it as a decimal value. You also cannot use the string `"1001"` because the Quartus Prime software interprets it as an ASCII string. You must use the type-encoded string `B"1001"` for the Quartus Prime software to correctly interpret the parameter value.

This table lists valid parameter strings and how the Quartus Prime software interprets the parameter strings. Use the type-encoded format only when necessary to resolve ambiguity.

Table 10. Valid Parameter Strings and Interpretations

Parameter String	Quartus Prime Parameter Type, Format, and Value
`S"abc"`, `s"abc"`	String value abc
`"abc123"`, `"123abc"`	String value abc123 or 123abc
`F"12.3"`, `f"12.3"`	Floating point number 12.3
`-5.4`	Floating point number -5.4
`D"123"`, `d"123"`	Decimal number 123
`123`, `-123`	Decimal number 123, -123
`X"ff"`, `H"ff"`	Hexadecimal value FF
`Q"77"`, `O"77"`	Octal value 77
`B"1010"`, `b"1010"`	Unsigned binary value 1010

continued...

Parameter String	Quartus Prime Parameter Type, Format, and Value
SB"1010", sb"1010"	Signed binary value 1010
R"1", R"0", R"X", R"Z", r"1", r"0", r"X", r"Z"	Unsized bit literal
E"apple", e"apple"	Enumeration type, value name is apple
P"1 unit"	Physical literal, the value is (1, unit)
A(...), a(...)	Array type or record type. The string (...) determines the array type or record type content

You can select the parameter type for global parameters or global constants with the pull-down list in the **Parameter** tab of the **Symbol Properties** dialog box. If you do not specify the parameter type, the Quartus Prime software interprets the parameter value and defines the parameter type. You must specify parameter type with the pull-down list to avoid ambiguity.

Note: If you open a **.bdf** in the Quartus Prime software, the software automatically updates the parameter types of old symbol blocks by interpreting the parameter value based on the language-independent format. If the Quartus Prime software does not recognize the parameter value type, the software sets the parameter type as **untyped**.

The Quartus Prime software supports the following parameter types:

- **Unsigned Integer**
- **Signed Integer**
- **Unsigned Binary**
- **Signed Binary**
- **Octal**
- **Hexadecimal**
- **Float**
- **Enum**
- **String**
- **Boolean**
- **Char**
- **Untyped/Auto**

3.2.7.2. Passing Parameters Between Two Design Languages

When passing a parameter between two different languages, a design block that is higher in the design hierarchy instantiates a lower-level subdesign block and provides parameter information. The subdesign language (the design entity that you instantiate) must correctly interpret the parameter. Based on the information provided by the higher-level design and the value format, and sometimes by the parameter type of the subdesign entity, the Quartus Prime software interprets the type and value of the passed parameter.

When passing a parameter whose value is an enumerated type value or literal from a language that does not support enumerated types to one that does (for example, from Verilog HDL to VHDL), you must ensure that the enumeration literal is in the correct

spelling in the language of the higher-level design block (block that is higher in the hierarchy). The Quartus Prime software passes the parameter value as a string literal, and the language of the lower-level design correctly convert the string literal into the correct enumeration literal.

If the language of the lower-level entity is SystemVerilog, you must ensure that the enum value is in the correct case. In SystemVerilog, two enumeration literals differ in more than just case. For example, `enum {item, ITEM}` is not a good choice of item names because these names can create confusion and is more difficult to pass parameters from case-insensitive HDLs, such as VHDL.

Arrays have different support in different design languages. For details about the array parameter format, refer to the **Parameter** section in the Analysis & Synthesis Report of a design that contains array parameters or generics.

The following code shows examples of passing parameters from one design entry language to a subdesign written in another language.

Table 11. VHDL Parameterized Subdesign Entity

This table shows a VHDL subdesign that you instantiate in a top-level Verilog HDL design in Table 12 on page 135.

HDL	Code

VHDL	`type fruit is (apple, orange, grape);`
	`entity vhdl_sub is`
	`generic (`
	`name : string := "default",`
	`width : integer := 8,`
	`number_string : string := "123",`
	`f : fruit := apple,`
	`binary_vector : std_logic_vector(3 downto 0) := "0101",`
	`signed_vector : signed (3 downto 0) := "1111");`

Table 12. Verilog HDL Top-Level Design Instantiating and Passing Parameters to VHDL Entity

This table shows a Verilog HDL Top-Level Design Instantiating and Passing Parameters to VHDL Entity from Table 11 on page 135.

HDL	Code

Verilog HDL	`vhdl_sub inst (...);`
	`defparam inst.name = "lower";`
	`defparam inst.width = 3;`
	`defparam inst.num_string = "321";`
	`defparam inst.f = "grape"; // Must exactly match enum value`
	`defparam inst.binary_vector = 4'b1010;`
	`defparam inst.signed_vector = 4'sb1010;`

Table 13. Verilog HDL Parameterized Subdesign Module

This table shows a Verilog HDL subdesign that you instantiate in a top-level VHDL design in Table 14 on page 136.

HDL	Code

Verilog HDL	`module veri_sub (...)`
	`parameter name = "default";`
	`parameter width = 8;`
	`parameter number_string = "123";`
	`parameter binary_vector = 4'b0101;`
	`parameter signed_vector = 4'sb1111;`

Table 14. VHDL Top-Level Design Instantiating and Passing Parameters to the Verilog HDL Module

This table shows a VHDL Top-Level Design Instantiating and Passing Parameters to the Verilog HDL Module from Table 13 on page 135.

HDL	Code

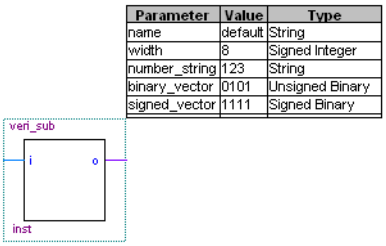
VHDL	`inst:veri_sub`
	`generic map (`
	`name => "lower",`
	`width => 3,`
	`number_string => "321"`
	`binary_vector = "1010"`
	`signed_vector = "1010")`

To use an HDL subdesign such as the one shown in Table 13 on page 135 in a top-level **.bdf** design, you must generate a symbol for the HDL file, as shown in Figure 34 on page 136. Open the HDL file in the Quartus Prime software, and then, on the File menu, point to **Create/Update,** and then click **Create Symbol Files for Current File**.

To specify parameters on a **.bdf** instance, double-click the parameter value box for the instance symbol, or right-click the symbol and click **Properties**, and then click the **Parameters** tab. Right-click the symbol and click **Update Design File from Selected Block** to pass the updated parameter to the HDL file.

Figure 34. BDF Top-Level Design Instantiating and Passing Parameters to the Verilog HDL Module

This figure shows BDF Top-Level Design Instantiating and Passing Parameters to the Verilog HDL Module from Table 13 on page 135



3.3. Incremental Compilation

Incremental compilation manages a design hierarchy for incremental design by allowing you to divide your design into multiple partitions. Incremental compilation ensures that the Quartus Prime software resynthesizes only the updated partitions of your design during compilation, to reduce the compilation time and the runtime memory usage. The feature maintains node names during synthesis for all registered and combinational nodes in unchanged partitions. You can perform incremental synthesis by setting the netlist type for all design partitions to **Post-Synthesis**.

You can also preserve the placement and routing information for unchanged partitions. This feature allows you to preserve performance of unchanged blocks in your design and reduces the time required for placement and routing, which significantly reduces your design compilation time.

Related Information

Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design on page 7
For more information about incremental compilation for hierarchical and team-based design

3.3.1. Partitions for Preserving Hierarchical Boundaries

A design partition represents a portion of your design that you want to synthesize and fit incrementally.

If you want to preserve the **Optimization Technique** and **Restructure Multiplexers** logic options in any entity, you must create new partitions for the entity instead of using the **Preserve Hierarchical Boundary** logic option. If you have settings applied to specific existing design hierarchies, particularly those created in the Quartus Prime software versions before 9.0, you must create a design partition for the design hierarchy so that synthesis can optimize the design instance independently and preserve the hierarchical boundaries.

Note: The **Preserve Hierarchical Boundary** logic option is available only in Quartus Prime software versions 8.1 and earlier. Altera recommends using design partitions if you want to preserve hierarchical boundaries through the synthesis and fitting process, because incremental compilation maintains the hierarchical boundaries of design partitions.

3.3.2. Parallel Synthesis

The **Parallel Synthesis** logic option reduces compilation time for synthesis. The option enables the Quartus Prime software to use multiple processors to synthesize multiple partitions in parallel.

This option is available when you perform the following tasks:

- Specify the maximum number of processors allowed under **Parallel Compilation** options in the **Compilation Process Settings** page of the **Settings** dialog box.
- Enable the incremental compilation feature.
- Use two or more partitions in your design.
- Turn on the **Parallel Synthesis** option.

By default, the Quartus Prime software enables the **Parallel Synthesis** option. To disable parallel synthesis, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis) ➤ Parallel Synthesis**.

You can also set the **Parallel Synthesis** option with the following Tcl command:

```
set_global_assignment -name parallel_synthesis off
```

If you use the command line, you can differentiate among the interleaved messages by turning on the **Show partition that generated the message** option in the Messages page. This option shows the partition ID in parenthesis for each message.

You can view all the interleaved messages from different partitions in the Messages window. The **Partition** column in the Messages window displays the partition ID of the partition referred to in the message. After compilation, you can sort the messages by partition.

Related Information

About the Messages Window
For more information about displaying the Partition column

3.3.3. Quartus Prime Exported Partition File as Source

You can use a **.qxp** as a source file in incremental compilation. The **.qxp** contains the precompiled design netlist exported as a partition from another Quartus Prime project, and fully defines the entity. Project team members or intellectual property (IP) providers can use a **.qxp** to send their design to the project lead, instead of sending the original HDL source code. The **.qxp** preserves the compilation results and instance-specific assignments. Not all global assignments can function in a different Quartus Prime project. You can override the assignments for the entity in the **.qxp** by applying assignments in the top-level design.

Related Information

- Quartus Prime Exported Partition File .qxp
  For more information about **.qxp**
- Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design on page 7
  For more information about exporting design partitions and using **.qxp** files

3.4. Quartus Prime Synthesis Options

The Quartus Prime software offers several options to help you control the synthesis process and achieve optimal results for your design.

Note: When you apply a Quartus Prime Synthesis option globally or to an entity, the option affects all lower-level entities in the hierarchy path, including entities instantiated with Altera and third-party IP.

Related Information

Setting Synthesis Options on page 138
Describes the **Compiler Settings** page of the **Settings** dialog box, in which you can set the most common global settings and options, and defines the following types of synthesis options: Quartus Prime logic options, synthesis attributes, and synthesis directives.

3.4.1. Setting Synthesis Options

You can set synthesis options in the **Settings** dialog box, or with logic options in the Quartus Prime software, or you can use synthesis attributes and directives in your HDL source code.

The **Compiler Settings** page of the **Settings** dialog box allows you to set global synthesis options that apply to the entire project. You can also use a corresponding Tcl command.

You can set some of the advanced synthesis settings in the **Advanced Settings** dialog box on the **Compiler Settings** page.

Related Information

Netlist Optimizations and Physical Synthesis
For more information about Physical Synthesis options

3.4.1.1. Quartus Prime Logic Options

The Quartus Prime logic options control many aspects of the synthesis and placement and routing process. To set logic options in the Quartus Prime software, on the Assignments menu, click **Assignment Editor**. You can also use a corresponding Tcl command to set global assignments. The Quartus Prime logic options enable you to set instance or node-specific assignments without editing the source HDL code.

3.4.1.2. Synthesis Attributes

The Quartus Prime software supports synthesis attributes for Verilog HDL and VHDL, also commonly called pragmas. These attributes are not standard Verilog HDL or VHDL commands. Synthesis tools use attributes to control the synthesis process. The Quartus Prime software applies the attributes in the HDL source code, and attributes always apply to a specific design element. Some synthesis attributes are also available as Quartus Prime logic options via the Quartus Prime software or scripting. Each attribute description indicates a corresponding setting or a logic option that you can set in the Quartus Prime software. You can specify only some attributes with HDL synthesis attributes.

Attributes specified in your HDL code are not visible in the Assignment Editor or in the **.qsf**. Assignments or settings made with the Quartus Prime software, the **.qsf**, or the Tcl interface take precedence over assignments or settings made with synthesis attributes in your HDL code. The Quartus Prime software generates warning messages if the software finds invalid attributes, but does not generate an error or stop the compilation. This behavior is necessary because attributes are specific to various design tools, and attributes not recognized in the Quartus Prime software might be for a different EDA tool. The Quartus Prime software lists the attributes specified in your HDL code in the Source assignments table of the Analysis & Synthesis report.

The Verilog-2001, SystemVerilog, and VHDL language definitions provide specific syntax for specifying attributes, but in Verilog-1995, you must embed attribute assignments in comments. You can enter attributes in your code using the syntax in Specifying Synthesis Attributes in Verilog-1995 on page 140 through Synthesis Attributes in VHDL on page 141, in which <*attribute*>, <*attribute type*>, <*value*>, <*object*>, and <*object type*> are variables, and the entry in brackets is optional. These examples demonstrate each syntax form.

Note: Verilog HDL is case sensitive; therefore, synthesis attributes in Verilog HDL files are also case sensitive.

In addition to the `synthesis` keyword shown above, the Quartus Prime software supports the `pragma`, `synopsys`, and `exemplar` keywords for compatibility with other synthesis tools. The software also supports the `altera` keyword, which allows you to add synthesis attributes that the Quartus Prime Integrated Synthesis feature recognizes and not by other tools that recognize the same synthesis attribute.

Note: Because formal verification tools do not recognize the `exemplar`, `pragma`, and `altera` keywords, avoid using these attribute keywords when using formal verification.

Related Information

- Maximum Fan-Out on page 158
  For more information about maximum fan-out attribute
- Preserve Registers on page 153
  For more information about `preserve` attribute

3.4.1.2.1. Synthesis Attributes in Verilog-1995

You must use Verilog-1995 comment-embedded attributes as a suffix to the declaration of an item and must appear before a semicolon, when a semicolon is necessary.

Note: You cannot use the open one-line comment in Verilog HDL when a semicolon is necessary after the line, because it is not clear to which HDL element that the attribute applies. For example, you cannot make an attribute assignment such as `reg r; // synthesis <attribute>` because the Quartus Prime software could read the attribute as part of the next line.

Specifying Synthesis Attributes in Verilog-1995

The following show an example of specifying synthesis attributes in Verilog-1995:

```
// synthesis <attribute> [ = <value> ]
or
/* synthesis <attribute> [ = <value> ] */
```

Applying Multiple Attributes to the Same Instance in Verilog-1995

To apply multiple attributes to the same instance in Verilog-1995, separate the attributes with spaces.

```
//synthesis <attribute1> [ = <value> ] <attribute2> [ = <value> ]
```

For example, to set the `maxfan` attribute to 16 and set the `preserve` attribute on a register called `my_reg`, use the following syntax:

```
reg my_reg /* synthesis maxfan = 16 preserve */;
```

Related Information

- Maximum Fan-Out on page 158
  For more information about maximum fan-out attribute
- Preserve Registers on page 153
  For more information about `preserve` attribute

3.4.1.2.2. Synthesis Attributes in Verilog-2001

You must use Verilog-2001 attributes as a prefix to a declaration, module item, statement, or port connection, and as a suffix to an operator or a Verilog HDL function name in an expression.

Note: Formal verification does not support the Verilog-2001 attribute syntax because the tools do not recognize the syntax.

Specifying Synthesis Attributes in Verilog-2001 and SystemVerilog

```
(* <attribute> [ = <value> ] *)
```

Applying Multiple Attributes

To apply multiple attributes to the same instance in Verilog-2001 or SystemVerilog, separate the attributes with commas.

```
(* <attribute1> [ = <value1>], <attribute2> [ = <value2> ] *)
```

For example, to set the `maxfan` attribute to 16 and set the `preserve` attribute on a register called `my_reg`, use the following syntax:

```
(* maxfan = 16, preserve *) reg my_reg;
```

Related Information

- Maximum Fan-Out on page 158
  For more information about maximum fan-out attribute
- Preserve Registers on page 153
  For more information about `preserve` attribute

3.4.1.2.3. Synthesis Attributes in VHDL

VHDL attributes declare and apply the attribute type to the object you specify.

Synthesis Attributes in VHDL

The following shows the synthesis attributes example in VHDL:

```
attribute <attribute> : <attribute type> ;
attribute <attribute> of <object> : <object type> is <value>;
```

altera_syn_attributes

The Quartus Prime software defines and applies each attribute separately to a given node. For VHDL designs, the software declares all supported synthesis attributes in the `altera_syn_attributes` package in the Altera library. You can call this library from your VHDL code to declare the synthesis attributes:

```
LIBRARY altera;
USE altera.altera_syn_attributes.all;
```

3.4.1.3. Synthesis Directives

The Quartus Prime software supports synthesis directives, also commonly called compiler directives or pragmas. You can include synthesis directives in Verilog HDL or VHDL code as comments. These directives are not standard Verilog HDL or VHDL commands. Synthesis tools use directives to control the synthesis process. Directives do not apply to a specific design node, but change the behavior of the synthesis tool from the point in which they occur in the HDL source code. Other tools, such as simulators, ignore these directives and treat them as comments.

Table 15. Specifying Synthesis Directives

You can enter synthesis directives in your code using the syntax in the following table, in which <*directive*> and <*value*> are variables, and the entry in brackets are optional. For synthesis directives, no equal sign before the value is necessary; this is different than the Verilog syntax for synthesis attributes. The examples demonstrate each syntax form.

Language	Syntax Example

Verilog HDL[4]	`// synthesis <directive> [ <value> ]`
	or
	`/* synthesis <directive> [ <value> ] */`

VHDL	`-- synthesis <directive> [ <value> ]`
VHDL-2008	`/* synthesis <directive> [<value>] */`

In addition to the `synthesis` keyword shown above, the software supports the `pragma`, `synopsys`, and `exemplar` keywords in Verilog HDL and VHDL for compatibility with other synthesis tools. The Quartus Prime software also supports the keyword `altera`, which allows you to add synthesis directives that only Quartus Prime Integrated Synthesis feature recognizes, and not by other tools that recognize the same synthesis directives.

Note: Because formal verification tools ignore the `exemplar`, `pragma`, and `altera` keywords, Altera recommends that you avoid using these directive keywords when you use formal verification to prevent mismatches with the Quartus Prime results.

3.4.2. Optimization Technique

The **Optimization Technique** logic option specifies the goal for logic optimization during compilation; that is, whether to attempt to achieve maximum speed performance or minimum area usage, or a balance between the two.

Related Information

Optimization Technique logic option
For more information about the Optimization Technique logic option

3.4.3. Auto Gated Clock Conversion

Clock gating is a common optimization technique in ASIC designs to minimize power consumption. You can use the **Auto Gated Clock Conversion** logic option to optimize your prototype ASIC designs by converting gated clocks into clock enables when you use FPGAs in your ASIC prototyping. The automatic conversion of gated clocks to clock enables is more efficient than manually modifying source code. The **Auto Gated Clock Conversion** logic option automatically converts qualified gated clocks (base clocks as defined in the Synopsys Design Constraints [SDC]) to clock enables. Click **AssignmentsSettingsCompiler SettingsAdvanced Settings (Synthesis)** to enable **Auto Gated Clock Conversion**.

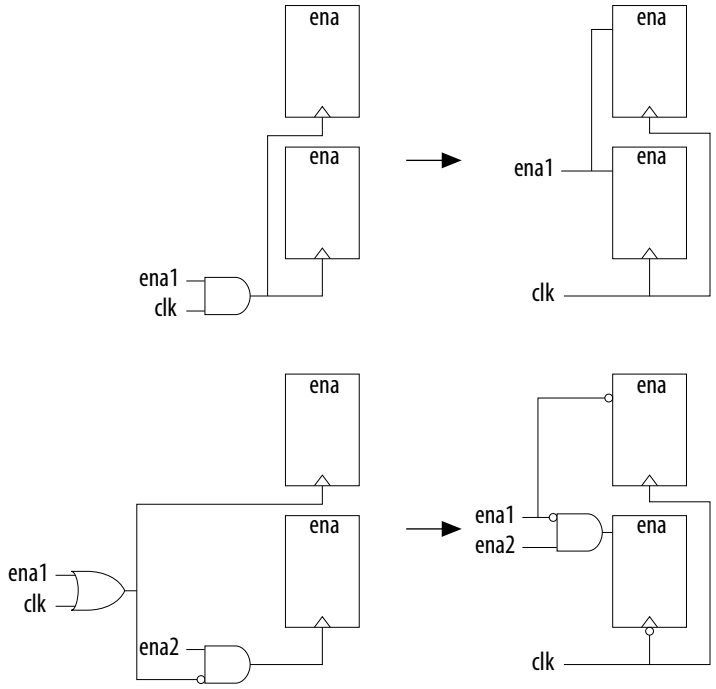
[4] Verilog HDL is case sensitive; therefore, all synthesis directives are also case sensitive.

The gated clock conversion occurs when all these conditions are met:

- Only one base clock drives a gated-clock
- For one set of gating input values, the value output of the gated clock remains constant and does not change as the base clock changes
- For one value of the base clock, changes in the gating inputs do not change the value output for the gated clock

The option supports combinational gates in clock gating network.

Figure 35. Example Gated Clock Conversion



Note: This option does not support registers in RAM, DSP blocks, or I/O related WYSIWYG primitives. Because the gated-clock conversion cannot trace the base clock from the gated clock, the gated clock conversion does not support multiple design partitions from incremental compilation in which the gated clock and base clock are not in the same hierarchical partition. A gated clock tree, instead of every gated clock, is the basis of each conversion. Therefore, if you cannot convert a gated clock from a root gated clock of a multiple cascaded gated clock, the conversion of the entire gated clock tree fails.

The **Info** tab in the Messages window lists all the converted gated clocks. You can view a list of converted and nonconverted gated clocks from the Compilation Report under the **Optimization Results** of the Analysis & Synthesis Report. The **Gated Clock Conversion Details** table lists the reasons for nonconverted gated clocks.

Related Information

Auto Gated Clock Conversion logic option
For more information about Auto Gated Clock Conversion logic option and a list of supported devices

3.4.4. Enabling Timing-Driven Synthesis

Timing-driven synthesis directs the Compiler to account for your timing constraints during synthesis. Timing-driven synthesis runs initial timing analysis to obtain netlist timing information. Synthesis then focuses performance efforts on timing-critical design elements, while optimizing non-timing-critical portions for area.

Timing-driven synthesis preserves timing constraints, and does not perform optimizations that conflict with timing constraints. Timing-driven synthesis may increase the number of required device resources. Specifically, the number of adaptive look-up tables (ALUTs) and registers may increase. The overall area can increase or decrease. Runtime and peak memory use increases slightly.

Timing-Driven Synthesis prevents registers with incompatible timing constraints from merging for any **Optimization Technique** setting. If your design contains multiple partitions, you can select **Timing-Driven Synthesis** options for each partition. If you use a `.qxp` as a source file, or if your design uses partitions developed in separate Quartus Prime projects, the software cannot properly compute timing of paths that cross the partition boundaries.

3.4.5. SDC Constraint Protection

The **SDC Constraint Protection** option specifies whether Analysis & Synthesis should protect registers from merging when they have incompatible timing constraints. For example, when you turn on this option, the software does not merge two registers that are duplicates of each other but have different multicycle constraints on them. When you turn on the **Timing-Driven Synthesis** option, the software detects registers with incompatible constraints, and you do not need to turn on **SDC Constraint Protection**. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** to enable the **SDC constraint protection** option.

3.4.6. PowerPlay Power Optimization

The **PowerPlay Power Optimization** logic option controls the power-driven compilation setting of Analysis & Synthesis and determines how aggressively Analysis & Synthesis optimizes your design for power.

Related Information

- PowerPlay Power Optimization logic option
  For more information about the available settings for the PowerPlay power optimization logic option and a list of supported devices
- Power Optimization
  For more information about optimizing your design for power utilization
- PowerPlay Power Analysis
  For information about analyzing your power results

3.4.7. Limiting Resource Usage in Partitions

Resource balancing is important when performing Analysis & Synthesis. During resource balancing, Quartus Prime Integrated Synthesis considers the amount of used and available DSP and RAM blocks in the device, and tries to balance these resources to prevent no-fit errors.

For DSP blocks, Resource balancing is important when performing Analysis & Synthesis. During resource balancing, Quartus Prime Integrated Synthesis considers the amount of used and available DSP and RAM blocks in the device, and tries to balance these resources to prevent no-fit errors. resource balancing converts the remaining DSP blocks to equivalent logic if there are more DSP blocks in your design that the software can place in the device. For RAM blocks, resource balancing converts RAM blocks to different types of RAM blocks if there are not enough blocks of a certain type available in the device; however, Quartus Prime Integrated Synthesis does not convert RAM blocks to logic.

Note: The RAM balancing feature does not support Stratix V devices because Stratix V has only M20K memory blocks.

By default, Quartus Prime Integrated Synthesis considers the information in the targeted device to identify the number of available DSP or RAM blocks. However, in incremental compilation, each partition considers the information in the device independently and consequently assumes that the partition has all the DSP and RAM blocks in the device available for use, resulting in over allocation of DSP or RAM blocks in your design, which means that the total number of DSP or RAM blocks used by all the partitions is greater than the number of DSP or RAM blocks available in the device, leading to a no-fit error during the fitting process.

Related Information

- Creating LogicLock Regions on page 145
  For more information about preventing a no-fit error during the fitting process
- Using Assignments to Limit the Number of RAM and DSP Blocks on page 146
  For more information about preventing a no-fit error during the fitting process

3.4.7.1. Creating LogicLock Regions

The floorplan-aware synthesis feature allows you to use LogicLock regions to define resource allocation for DSP blocks and RAM blocks. For example, if you assign a certain partition to a certain LogicLock region, resource balancing takes into account that all the DSP and RAM blocks in that partition need to fit in this LogicLock region. Resource balancing then balances the DSP and RAM blocks accordingly.

Because floorplan-aware balancing step considers only one partition at a time, it does not know that nodes from another partition may be using the same resources. When using this feature, Altera recommends that you do not manually assign nodes from different partitions to the same LogicLock region.

If you do not want the software to consider the LogicLock floorplan constraints when performing DSP and RAM balancing, you can turn off the floorplan-aware synthesis feature. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** to disable **Use LogicLock Constraints During Resource Balancing** option.

Related Information

Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design on page 7
For more information about using LogicLock regions to create a floorplan for incremental compilation

3.4.7.2. Using Assignments to Limit the Number of RAM and DSP Blocks

For DSP and RAM block balancing, you can use assignments to limit the maximum number of blocks that the balancer allows. You can set these assignments globally or on individual partitions. For DSP block balancing, the **Maximum DSP Block Usage** logic option allows you to specify the maximum number of DSP blocks that the DSP block balancer assumes are available for the current partition. For RAM blocks, the floorplan-aware logic option allows you to specify maximum resources for different RAM types, such as **Maximum Number of M4K/M9K/M20K/M10K Memory Blocks**, **Maximum Number of M512 Memory Blocks**, **Maximum Number of M-RAM/M144K Memory Blocks**, or **Maximum Number of LABs**.

The partition-specific assignment overrides the global assignment, if any. However, each partition that does not have a partition-specific assignment uses the value set by the global assignment, or the value derived from the device size if no global assignment exists. This action can also lead to over allocation. Therefore, Altera recommends that you always set the assignment on each partition individually.

To select the **Maximum Number <block type> Memory Blocks** option or the **Maximum DSP Block Usage** option globally, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. You can use the Assignment Editor to set this assignment on a partition by selecting the assignment, and setting it on the root entity of a partition. You can set any positive integer as the value of this assignment. If you set this assignment on a name other than a partition root, Analysis & Synthesis gives an error.

Related Information

- Maximum DSP Block Usage logic option on page 0
  For more information about the **Maximum DSP Block Usage** logic option, including a list of supported device families
- Maximum Number of M4K/M9K/M20K/M10K Memory Blocks logic option on page 0
  For more information about the **Maximum Number of M4K/M9K/M20K/M10K Memory Blocks** logic option, including a list of supported device families

3.4.8. Restructure Multiplexers

The **Restructure Multiplexers logic** option restructures multiplexers to create more efficient use of area, allowing you to implement multiplexers with a reduced number of LEs or ALMs.

When multiplexers from one part of your design feed multiplexers in another part of your design, trees of multiplexers form. Multiplexers may arise in different parts of your design through Verilog HDL or VHDL constructs such as the "`if`," "`case`," or "`?:`" statements. Multiplexer buses occur most often as a result of multiplexing together arrays in Verilog HDL, or `STD_LOGIC_VECTOR` signals in VHDL. The **Restructure Multiplexers** logic option identifies buses of multiplexer trees that have a similar structure. This logic option optimizes the structure of each multiplexer bus for the target device to reduce the overall amount of logic in your design.

Results of the multiplexer optimizations are design dependent, but area reductions as high as 20% are possible. The option can negatively affect your design's f_{MAX}.

Related Information

- Analysis Synthesis Optimization Results Reports
  For more information about the Multiplexer Restructuring Statistics report table for each bus of multiplexers
- Restructure Multiplexers logic option
  For more information about the Restructure Multiplexers logic option, including the settings and a list of supported device families

3.4.9. Synthesis Effort

The **Synthesis Effort** logic option specifies the overall synthesis effort level in the Quartus Prime software.

Related Information

Synthesis Effort logic option
For more information about Synthesis Effort logic option, including a list of supported device families

3.4.10. Fitter Intial Placement Seed

Specifies the starting value the Fitter uses when randomly determining the initial placement for the current design. The value can be any non-negative integer value. Changing the starting value may or may not produce better fitting. Specify a starting value only if the Fitter is not meeting timing requirements by a small amount. Use the Design Space Explorer to sweep many seed values easily and find the best value for your project. Modifying the design or Quartus settings even slightly usually changes which seed is best for the design.

To set the **Synthesis Seed** option, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**. The default value is **1**. You can specify a positive integer value.

3.4.11. State Machine Processing

The **State Machine Processing** logic option specifies the processing style to synthesize a state machine.

The default state machine encoding, **Auto**, uses one-hot encoding for FPGA devices and minimal-bits encoding for CPLDs. These settings achieve the best results on average, but another encoding style might be more appropriate for your design, so this option allows you to control the state machine encoding.

For one-hot encoding, the Quartus Prime software does not guarantee that each state has one bit set to one and all other bits set to zero. Quartus Prime Integrated Synthesis creates one-hot register encoding with standard one-hot encoding and then inverts the first bit. This results in an initial state with all zero values, and the remaining states have two 1 values. Quartus Prime Integrated Synthesis encodes the initial state with all zeros for the state machine power-up because all device registers power up to a low value. This encoding has the same properties as true one-hot

encoding: the software recognizes each state by the value of one bit. For example, in a one-hot-encoded state machine with five states, including an initial or reset state, the software uses the following register encoding:

```
State 0    0 0 0 0 0
State 1    0 0 0 1 1
State 2    0 0 1 0 1
State 3    0 1 0 0 1
State 4    1 0 0 0 1
```

If you set the **State Machine Processing** logic option to **User-Encoded** in a Verilog HDL design, the software starts with the original design values for the state constants. For example, a Verilog HDL design can contain the following declaration:

```
parameter S0 = 4'b1010, S1 = 4'b0101, ...
```

If the software infers the states `S0, S1, . . .` the software uses the encoding `4'b1010, 4'b0101, . . .` . If necessary, the software inverts bits in a user-encoded state machine to ensure that all bits of the reset state of the state machine are zero.

Note: You can view the state machine encoding from the Compilation Report under the State Machines of the Analysis & Synthesis Report. The State Machine Viewer displays only a graphical representation of the state machines as interpreted from your design.

To assign your own state encoding with the **User-Encoded** setting of the **State Machine Processing** option in a VHDL design, you must apply specific binary encoding to the elements of an enumerated type because enumeration literals have no numeric values in VHDL. Use the `syn_encoding` synthesis attribute to apply your encoding values.

Related Information

- Manually Specifying State Assignments Using the syn_encoding Attribute on page 148
- State Machine Processing logic option

3.4.11.1. Manually Specifying State Assignments Using the syn_encoding Attribute

The Quartus Prime software infers state machines from enumerated types and automatically assigns state encoding based on State Machine Processing on page 147.

With this logic option, you can choose the value **User-Encoded** to use the encoding from your HDL code. However, in standard VHDL code, you cannot specify user encoding in the state machine description because enumeration literals have no numeric values in VHDL.

To assign your own state encoding for the **User-Encoded State Machine Processing** setting, use the `syn_encoding` synthesis attribute to apply specific binary encodings to the elements of an enumerated type or to specify an encoding style. The Quartus Prime software can implement Enumeration Types with different encoding styles, as listed in this table.

Table 16. syn_encoding Attribute Values

Attribute Value	Enumeration Types
"default"	Use an encoding based on the number of enumeration literals in the Enumeration Type. If the number of literals is less than five, use the "sequential" encoding. If the number of literals is more than five, but fewer than 50, use a "one-hot" encoding. Otherwise, use a "gray" encoding.
"sequential"	Use a binary encoding in which the first enumeration literal in the Enumeration Type has encoding 0 and the second 1.
"gray"	Use an encoding in which the encodings for adjacent enumeration literals differ by exactly one bit. An N-bit gray code can represent 2^N values.
"johnson"	Use an encoding similar to a gray code. An N-bit Johnson code can represent at most 2^N states, but requires less logic than a gray encoding.
"one-hot"	The default encoding style requiring N bits, in which N is the number of enumeration literals in the Enumeration Type.
"compact"	Use an encoding with the fewest bits.
"user"	Encode each state using its value in the Verilog source. By changing the values of your state constants, you can change the encoding of your state machine.

The syn_encoding attribute must follow the enumeration type definition, but precede its use.

Related Information

State Machine Processing on page 147

3.4.11.2. Manually Specifying Enumerated Types Using the enum_encoding Attribute

By default, the Quartus Prime software one-hot encodes all enumerated types you defined. With the enum_encoding attribute, you can specify the logic encoding for an enumerated type and override the default one-hot encoding to improve the logic efficiency.

Note: If an enumerated type represents the states of a state machine, using the enum_encoding attribute to specify a manual state encoding prevents the Compiler from recognizing state machines based on the enumerated type. Instead, the Compiler processes these state machines as regular logic with the encoding specified by the attribute, and the Report window for your project does not list these states machines as state machines. If you want to control the encoding for a recognized state machine, use the **State Machine Processing** logic option and the syn_encoding synthesis attribute.

To use the enum_encoding attribute in a VHDL design file, associate the attribute with the enumeration type whose encoding you want to control. The enum_encoding attribute must follow the enumeration type definition, but precede its use. In addition, the attribute value should be a string literal that specifies either an arbitrary user encoding or an encoding style of "default", "sequential", "gray", "johnson", or "one-hot".

An arbitrary user encoding consists of a space-delimited list of encodings. The list must contain as many encodings as the number of enumeration literals in your enumeration type. In addition, the encodings should have the same length, and each encoding must consist solely of values from the `std_ulogic` type declared by the `std_logic_1164` package in the IEEE library.

In this example, the `enum_encoding` attribute specifies an arbitrary user encoding for the enumeration type `fruit`.

Example 19. Specifying an Arbitrary User Encoding for Enumerated Type

```
type fruit is (apple, orange, pear, mango);
attribute enum_encoding : string;
attribute enum_encoding of fruit : type is "11 01 10 00";
```

This example shows the encoded enumeration literals:

Example 20. Encoded Enumeration Literals

```
apple   = "11"
orange  = "01"
pear    = "10"
mango   = "00"
```

Altera recommends that you specify an encoding style, rather than a manual user encoding, especially when the enumeration type has a large number of enumeration literals. The Quartus Prime software can implement Enumeration Types with the different encoding styles, as shown in this table.

Table 17. enum_encoding Attribute Values

Attribute Value	Enumeration Types
`"default"`	Use an encoding based on the number of enumeration literals in the enumeration type. If the number of literals are fewer than five, use the `"sequential"` encoding. If the number of literals are more than five, but fewer than 50 literals, use a `"one-hot"` encoding. Otherwise, use a `"gray"` encoding.
`"sequential"`	Use a binary encoding in which the first enumeration literal in the enumeration type has encoding `0` and the second `1`.
`"gray"`	Use an encoding in which the encodings for adjacent enumeration literals differ by exactly one bit. An N-bit gray code can represent 2^N values.
`"johnson"`	Use an encoding similar to a gray code. An N-bit Johnson code can represent at most 2^N states, but requires less logic than a gray encoding.
`"one-hot"`	The default encoding style requiring N bits, in which N is the number of enumeration literals in the enumeration type.

In Specifying an Arbitrary User Encoding for Enumerated Type on page 149, the `enum_encoding` attribute manually specified a gray encoding for the enumeration type `fruit`. You can also concisely write this example by specifying the `"gray"` encoding style instead of a manual encoding, as shown in the following example:

Example 21. Specifying the "gray" Encoding Style or Enumeration Type

```
type fruit is (apple, orange, pear, mango);
attribute enum_encoding : string;
attribute enum_encoding of fruit : type is "gray";
```

3.4.12. Safe State Machine

The **Safe State Machine** logic option and corresponding `syn_encoding` attribute value `safe` specify that the software must insert extra logic to detect an illegal state, and force the transition of the state machine to the reset state.

A finite state machine can enter an illegal state—meaning the state registers contain a value that does not correspond to any defined state. By default, the behavior of the state machine that enters an illegal state is undefined. However, you can set the `syn_encoding` attribute to `safe` or use the **Safe State Machine** logic option if you want the state machine to recover deterministically from an illegal state. The software inserts extra logic to detect an illegal state, and forces the transition of the state machine to the reset state. You can use this logic option when the state machine enters an illegal state. The most common cause of an illegal state is a state machine that has control inputs that come from another clock domain, such as the control logic for a clock-crossing FIFO, because the state machine must have inputs from another clock domain. This option protects only state machines (and not other registers) by forcing them into the reset state. You can use this option if your design has asynchronous inputs. However, Altera recommends using a synchronization register chain instead of relying on the safe state machine option.

The `safe` state machine value does not use any user-defined default logic from your HDL code that corresponds to unreachable states. Verilog HDL and VHDL enable you to specify a behavior for all states in the state machine explicitly, including unreachable states. However, synthesis tools detect if state machine logic is unreachable and minimize or remove the logic. Synthesis tools also remove any flag signals or logic that indicate such an illegal state. If the software implements the state machine as safe, the recovery logic added by Quartus Prime Integrated Synthesis forces its transition from an illegal state to the reset state.

You can set the **Safe State Machine** logic option globally, or on individual state machines. To set this logic option, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.

Table 18. Setting the `syn_encoding safe` attribute on a State Machine in HDL

HDL	Code
Verilog HDL	`reg [2:0] my_fsm /* synthesis syn_encoding = "safe" */;`
Verilog-2001 and SystemVerilog	`(* syn_encoding = "safe" *) reg [2:0] my_fsm;`
VHDL	`ATTRIBUTE syn_encoding OF my_fsm : TYPE IS "safe";`

If you specify an encoding style, separate the encoding style value in the quotation marks with the `safe` value with a comma, as follows: `"safe, one-hot"` or `"safe, gray"`.

Safe state machine implementation can result in a noticeable area increase for your design. Therefore, Altera recommends that you set this option only on the critical state machines in your design in which the safe mode is necessary, such as a state machine that uses inputs from asynchronous clock domains. You may not need to use this option if you correctly synchronize inputs coming from other clock domains.

Note: If you create the `safe` state machine assignment on an instance that the software fails to recognize as a state machine, or an entity that contains a state machine, the software takes no action. You must restructure the code, so that the software recognizes and infers the instance as a state machine.

Related Information

- Manually Specifying State Assignments Using the syn_encoding Attribute on page 148
- Safe State Machine logic option
  For more information about the Safe State Machine logic option

3.4.13. Power-Up Level

This logic option causes a register (flipflop) to power up with the specified logic level, either high (`1`) or low (`0`). The registers in the core hardware power up to `0` in all Altera devices. For the register to power up with a logic level high, the Compiler performs an optimization referred to as NOT-gate push back on the register. 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 to be high and the device operates as expected. The register itself still powers up to `0`, but the register output inverts so the signal arriving at all destinations is `1`.

The **Power-Up Level** option supports wildcard characters, and you can apply this option to any register, registered logic cell WYSIWYG primitive, or to a design entity containing registers, if you want to set the power level for all registers in your design entity. If you assign this option to a registered logic cell WYSIWYG primitive, such as an atom primitive from a third-party synthesis tool, you must turn on the **Perform WYSIWYG Primitive Resynthesis** logic option for the option to take effect. You can also apply the option to a pin with the logic configurations described in the following list:

- If you turn on this option for an input pin, the option transfers to the register that the pin drives, if all these conditions are present:
  - No logic, other than inversion, between the pin and the register.
  - The input pin drives the data input of the register.
  - The input pin does not fan-out to any other logic.
- If you turn on this option for an output or bidirectional pin, the option transfers to the register that feeds the pin, if all these conditions are present:
  - No logic, other than inversion, between the register and the pin.
  - The register does not fan out to any other logic.

Related Information

Power-Up Level logic option
For more information about the Power-Up Level logic option, including information on the supported device families

3.4.13.1. Inferred Power-Up Levels

Quartus Prime Integrated Synthesis reads default values for registered signals defined in Verilog HDL and VHDL code, and converts the default values into **Power-Up Level** settings. The software also synthesizes variables with assigned values in Verilog HDL

initial blocks into power-up conditions. Synthesis of these default and initial constructs allows synthesized behavior of your design to match, as closely as possible, the power-up state of the HDL code during a functional simulation.

The following register declarations all set a power-up level of V_{CC} or a logic value "1", as shown in this example:

```
signal q : std_logic = '1';  -- power-up to VCC

reg q = 1'b1;  // power-up to VCC

reg q;
initial begin q = 1'b1; end  // power-up to VCC
```

3.4.14. Power-Up Don't Care

This logic option allows the Compiler to optimize registers in your design that do not have a defined power-up condition.

For example, your design might have a register with its D input tied to V_{CC}, and with no clear signal or other secondary signals. If you turn on this option, the Compiler can choose for the register to power up to V_{CC}. Therefore, the output of the register is always V_{CC}. The Compiler can remove the register and connect its output to V_{CC}. If you turn this option off or if you set a **Power-Up Level** assignment of **Low** for this register, the register transitions from GND to V_{CC} when your design starts up on the first clock signal. Thus, the register is at V_{CC} and you cannot remove the register. Similarly, if the register has a clear signal, the Compiler cannot remove the register because after asserting the clear signal, the register transitions again to GND and back to V_{CC}.

If the Compiler performs a **Power-Up Don't Care** optimization that allows it to remove a register, it issues a message to indicate that it is doing so.

This project-wide option does not apply to registers that have the **Power-Up Level** logic option set to either **High** or **Low**.

Related Information

Power-Up Don't Care logic option
For more information about Power-Up Don't Care logic option and a list of supported devices

3.4.15. Remove Duplicate Registers

The **Remove Duplicate Registers** logic option removes registers that are identical to other registers.

Related Information

Remove Duplicate Registers logic option
For more information about Remove Duplicate Registers logic option and the supported devices

3.4.16. Preserve Registers

This attribute and logic option directs the Compiler not to minimize or remove a specified register during synthesis optimizations or register netlist optimizations. Optimizations can eliminate redundant registers and registers with constant drivers;

this option prevents the software from reducing a register to a constant or merging with a duplicate register. This option can preserve a register so you can observe the register during simulation or with the Signal Tap. Additionally, this option can preserve registers if you create a preliminary version of your design in which you have not specified the secondary signals. You can also use the attribute to preserve a duplicate of an I/O register so that you can place one copy of the I/O register in an I/O cell and the second in the core.

Note: This option cannot preserve registers that have no fan-out.

The **Preserve Registers** logic option prevents the software from inferring a register as a state machine.

You can set the **Preserve Registers** logic option in the Quartus Prime software, or you can set the `preserve` attribute in your HDL code. In these examples, the Quartus Prime software preserves the `my_reg` register.

Table 19. Setting the syn_preserve attribute in HDL Code

HDL	Code[5]
Verilog HDL	`reg my_reg /* synthesis syn_preserve = 1 */;`
Verilog-2001	`(* syn_preserve = 1 *) reg my_reg;`

Table 20. Setting the preserve attribute in HDL Code

In addition to `preserve`, the Quartus Prime software supports the `syn_preserve` attribute name for compatibility with other synthesis tools.

HDL	Code

VHDL	`signal my_reg : stdlogic;`
	`attribute preserve : boolean;`
	`attribute preserve of my_reg : signal is true;`

Related Information

- Preserve Registers logic option
  For more information about the Preserve Registers logic option and the supported devices
- Noprune Synthesis Attribute/Preserve Fan-out Free Register Node on page 155
  For more information about preventing the removal of registers with no fan-out

3.4.17. Disable Register Merging/Don't Merge Register

This logic option and attribute prevents the specified register from merging with other registers and prevents other registers from merging with the specified register. When applied to a design entity, it applies to all registers in the entity.

(5) The = 1 after the `preserve` are optional, because the assignment uses a default value of 1 when you specify the assignment.

You can set the **Disable Register Merging** logic option in the Quartus Prime software, or you can set the `dont_merge` attribute in your HDL code, as shown in these examples. In these examples, the logic option or the attribute prevents the `my_reg` register from merging.

Table 21. Setting the `dont_merge` attribute in HDL code

HDL	Code
Verilog HD	`reg my_reg /* synthesis dont_merge */;`
Verilog-2001 and SystemVerilog	`(* dont_merge *) reg my_reg;`

VHDL	`signal my_reg : stdlogic;`
	`attribute dont_merge : boolean;`
	`attribute dont_merge of my_reg : signal is true;`

Related Information

Disable Register Merging logic option
For more information about the **Disable Register Merging** logic option and the supported devices

3.4.18. Noprune Synthesis Attribute/Preserve Fan-out Free Register Node

This synthesis attribute and corresponding logic option direct the Compiler to preserve a fan-out-free register through the entire compilation flow. This option is different from the **Preserve Registers** option, which prevents the Quartus Prime software from reducing a register to a constant or merging with a duplicate register. Standard synthesis optimizations remove nodes that do not directly or indirectly feed a top-level output pin. This option can retain a register so you can observe the register in the Simulator or the Signal TapAdditionally, this option can retain registers if you create a preliminary version of your design in which you have not specified the fan-out logic of the register.

You can set the **Preserve Fan-out Free Register Node** logic option in the Quartus Prime software, or you can set the `noprune` attribute in your HDL code, as shown in these examples. In these examples, the logic option or the attribute preserves the `my_reg` register.

Note: You must use the `noprune` attribute instead of the logic option if the register has no immediate fan-out in its module or entity. If you do not use the synthesis attribute, the software removes (or "prunes") registers with no fan-out during Analysis & Elaboration before the logic synthesis stage applies any logic options. If the register has no fan-out in the full design, but has fan-out in its module or entity, you can use the logic option to retain the register through compilation.

The software supports the attribute name `syn_noprune` for compatibility with other synthesis tools.

Table 22. Setting the `noprune` attribute in HDL code

HDL	Code
Verilog HD	`reg my_reg /* synthesis syn_noprune */;`

continued...

HDL	Code
Verilog-2001 and SystemVerilog	`(* noprune *) reg my_reg;`

VHDL	`signal my_reg : stdlogic;`
	`attribute noprune: boolean;`
	`attribute noprune of my_reg : signal is true;`

Related Information

Preserve Fan-out Free Register logic option
For more information about **Preserve Fan-out Free Register Node** logic option and a list of supported devices

3.4.19. Keep Combinational Node/Implement as Output of Logic Cell

This synthesis attribute and corresponding logic option direct the Compiler to keep a wire or combinational node through logic synthesis minimizations and netlist optimizations. A wire that has a `keep` attribute or a node that has the **Implement as Output of Logic Cell** logic option applied becomes the output of a logic cell in the final synthesis netlist, and the name of the logic cell remains the same as the name of the wire or node. You can use this directive to make combinational nodes visible to the Signal Tap.

Note: The option cannot keep nodes that have no fan-out. You cannot maintain node names for wires with tri-state drivers, or if the signal feeds a top-level pin of the same name (the software changes the node name to a name such as <net name>~`buf0`).

You can use the **Ignore LCELL Buffers** logic option to direct Analysis & Synthesis to ignore logic cell buffers that the **Implement as Output of Logic Cell** logic option or the LCELL primitive created. If you apply this logic option to an entity, it affects all lower-level entities in the hierarchy path.

Note: To avoid unintended design optimizations, ensure that any entity instantiated with Altera or third-party IP that relies on logic cell buffers for correct behavior does not inherit the **Ignore LCELL Buffers** logic option. For example, if an IP core uses logic cell buffers to manage high fan-out signals and inherits the **Ignore LCELL Buffers** logic option, the target device may no longer function properly.

You can turn off the **Ignore LCELL Buffers** logic option for a specific entity to override any assignments inherited from higher-level entities in the hierarchy path if logic cell buffers created by the **Implement as Output of Logic Cell** logic option or the LCELL primitive are required for correct behavior.

You can set the **Implement as Output of Logic Cell** logic option in the Quartus Prime software, or you can set the `keep` attribute in your HDL code, as shown in these tables. In these tables, the Compiler maintains the node name `my_wire`.

Table 23. Setting the keep Attribute in HDL code

HDL	Code
Verilog HD	`wire my_wire /* synthesis keep = 1 */;`
Verilog-2001	`(* keep = 1 *) wire my_wire;`

Table 24. Setting the syn_keep Attribute in HDL Code

In addition to keep, the Quartus Prime software supports the syn_keep attribute name for compatibility with other synthesis tools.

HDL	Code

VHDL	`signal my_wire: bit;`
	`attribute syn_keep: boolean;`
	`attribute syn_keep of my_wire: signal is true;`

Related Information

Implement as Output of Logic Cell logic option
For more information about the **Implement as Output of Logic Cell** logic option and the supported devices

3.4.20. Disabling Synthesis Netlist Optimizations with dont_retime Attribute

This attribute disables synthesis retiming optimizations on the register you specify. When applied to a design entity, it applies to all registers in the entity.

You can turn off retiming optimizations with this option and prevent node name changes, so that the Compiler can correctly use your timing constraints for the register.

You can set the **Netlist Optimizations** logic option to **Never Allow** in the Quartus Prime software to disable retiming along with other synthesis netlist optimizations, or you can set the `dont_retime` attribute in your HDL code, as shown in the following table. In the following table, the code prevents `my_reg` register from being retimed.

Table 25. Setting the dont_retime Attribute in HDL Code

HDL	Code
Verilog HDL	`reg my_reg /* synthesis dont_retime */;`
Verilog-2001 and SystemVerilo	`(* dont_retime *) reg my_reg;`

VHD	`signal my_reg : std_logic;`
	`attribute dont_retime : boolean;`
	`attribute dont_retime of my_reg : signal is true;`

Note: For compatibility with third-party synthesis tools, Quartus Prime Integrated Synthesis also supports the attribute `syn_allow_retiming`. To disable retiming, set `syn_allow_retiming` to `0` (Verilog HDL) or `false` (VHDL). This attribute does not have any effect when you set the attribute to `1` or `true`.

3.4.21. Disabling Synthesis Netlist Optimizations with dont_replicate Attribute

This attribute disables synthesis replication optimizations on the register you specify. When applied to a design entity, it applies to all registers in the entity.

You can turn off register replication (or duplication) optimizations with this option, so that the Compiler uses your timing constraints for the register.

You can set the **Netlist Optimizations** logic option to **Never Allow** in the Quartus Prime software to disable replication along with other synthesis netlist optimizations, or you can set the `dont_replicate` attribute in your HDL code, as shown in these examples. In these examples, the code prevents the replication of the `my_reg` register.

Table 26. Setting the `dont_replicate` attribute in HDL Code

HDL	Code
Verilog HD	`reg my_reg /* synthesis dont_replicate  */;`
Verilog-2001 and SystemVerilog	`(* dont_replicate *) reg my_reg;`

VHDL	`signal my_reg : std_logic;`
	`attribute dont_replicate : boolean;`
	`attribute dont_replicate of my_reg : signal is true;`

Note: For compatibility with third-party synthesis tools, Quartus Prime Integrated Synthesis also supports the attribute `syn_replicate`. To disable replication, set `syn_replicate` to `0` (Verilog HDL) or `false` (VHDL). This attribute does not have any effect when you set the attribute to `1` or `true`.

3.4.22. Maximum Fan-Out

This **Maximum Fan-Out** attribute and logic option direct the Compiler to control the number of destinations that a node feeds. The Compiler duplicates a node and splits its fan-out until the individual fan-out of each copy falls below the maximum fan-out restriction. You can apply this option to a register or a logic cell buffer, or to a design entity that contains these elements. You can use this option to reduce the load of critical signals, which can improve performance. You can use the option to instruct the Compiler to duplicate a register that feeds nodes in different locations on the target device. Duplicating the register can enable the Fitter to place these new registers closer to their destination logic to minimize routing delay.

To turn off the option for a given node if you set the option at a higher level of the design hierarchy, in the **Netlist Optimizations** logic option, select **Never Allow**. If not disabled by the **Netlist Optimizations** option, the Compiler acknowledges the maximum fan-out constraint as long as the following conditions are met:

- The node is not part of a cascade, carry, or register cascade chain.
- The node does not feed itself.
- The node feeds other logic cells, DSP blocks, RAM blocks, and pins through data, address, clock enable, and other ports, but not through any asynchronous control ports (such as asynchronous clear).

The Compiler does not create duplicate nodes in these cases, because there is no clear way to duplicate the node, or to avoid the small differences in timing which could produce functional differences in the implementation (in the third condition above in which asynchronous control signals are involved). If you cannot apply the constraint because you do not meet one of these conditions, the Compiler issues a message to indicate that the Compiler ignores the maximum fan-out assignment. To instruct the Compiler not to check node destinations for possible problems such as the third condition, you can set the **Netlist Optimizations** logic option to **Always Allow** for a given node.

Note: If you have enabled any of the Quartus Prime netlist optimizations that affect registers, add the `preserve` attribute to any registers to which you have set a `maxfan` attribute. The `preserve` attribute ensures that the netlist optimization algorithms, such as register retiming, do not affect the registers.

You can set the **Maximum Fan-Out** logic option in the Quartus Prime software. This option supports wildcard characters. You can also set the `maxfan` attribute in your HDL code, as shown in these examples. In these examples, the Compiler duplicates the `clk_gen` register, so its fan-out is not greater than 50.

Table 27. Setting the `maxfan` attribute in HDL Code

HDL	Code
Verilog HDL	`reg clk_gen /* synthesis syn_maxfan = 50 */;`
Verilog-2001	`(* maxfan = 50 *) reg clk_gen;`

Table 28. Setting the `syn_maxfan` attribute in HDL Code

The Quartus Prime software supports the `syn_maxfan` attribute for compatibility with other synthesis tools.

HDL	Code

VHDL	`signal clk_gen : stdlogic;`
	`attribute maxfan : signal ;`
	`attribute maxfan of clk_gen : signal is 50;`

Related Information

- Netlist Optimizations and Physical Synthesis
  For details about netlist optimizations
- Maximum Fan-Out logic option
  For more information about the Maximum Fan-Out logic option and the supported devices

3.4.23. Controlling Clock Enable Signals with Auto Clock Enable Replacement and direct_enable

The **Auto Clock Enable Replacement** logic option allows the software to find logic that feeds a register and move the logic to the register's clock enable input port. To solve fitting or performance issues with designs that have many clock enables, you can turn off this option for individual registers or design entities. Turning the option off prevents the software from using the register's clock enable port. The software implements the clock enable functionality using multiplexers in logic cells.

If the software does not move the specific logic to a clock enable input with the **Auto Clock Enable Replacement** logic option, you can instruct the software to use a direct clock enable signal. The attribute ensures that the signal drives the clock enable port, and the software does not optimize or combine the signal with other logic.

These tables show how to set this attribute to ensure that the attribute preserves the signal and uses the signal as a clock enable.

Table 29. Setting the `direct_enable` in HDL Code

HDL	Code
Verilog HDL	`wire my_enable /* synthesis direct_enable = 1 */ ;`
VHDL	`attribute direct_enable: boolean;` `attribute direct_enable of my_enable: signal is true;`

Table 30. Setting the `syn_direct_enable` in HDL Code

The Quartus Prime software supports the `syn_direct_enable` attribute name for compatibility with other synthesis tools.

HDL	Code
Verilog-2001 and SystemVerilog	`(* syn_direct_enable *) wire my_enable;`

Related Information

Auto Clock Enable Replacement logic option
For more information about the **Auto Clock Enable Replacement** logic option and the supported devices

3.5. Inferring Multiplier, DSP, and Memory Functions from HDL Code

The Quartus Prime Compiler automatically recognizes multipliers, multiply-accumulators, multiply-adders, or memory functions described in HDL code, and either converts the HDL code into respective IP core or maps them directly to device atoms or memory atoms. If the software converts the HDL code into an IP core, the software uses the Altera IP core code when you compile your design, even when you do not specifically instantiate the IP core. The software infers IP cores to take advantage of logic that you optimize for Altera devices. The area and performance of such logic can be better than the results from inferring generic logic from the same HDL code.

Additionally, you must use IP cores to access certain architecture-specific features, such as RAM, DSP blocks, and shift registers that provide improved performance compared with basic logic cells.

The Quartus Prime software provides options to control the inference of certain types of IP cores.

3.5.1. Multiply-Accumulators and Multiply-Adders

Use the **Auto DSP Block Replacement** logic option to control DSP block inference for multiply-accumulations and multiply-adders. To disable inference, turn off this option for the entire project on the **Advanced Analysis & Synthesis** dialog box of the **Compiler Settings** page.

Related Information

Auto DSP Block Replacement logic option
For more information about the Auto DSP Block Replacement logic option and the supported devices

3.5.2. Shift Registers

Use the **Auto Shift Register Replacement** logic option to control shift register inference. This option has three settings: **Off**, **Auto** and **Always**. **Auto** is the default setting in which Quartus Prime Integrated Synthesis decides which shift registers to replace or leave in registers. Placing shift registers in memory saves logic area, but can have a negative effect on f_{max}. Quartus Prime Integrated Synthesis uses the optimization technique setting, logic and RAM utilization of your design, and timing information from **Timing-Driven Synthesis** to determine which shift registers are located in memory and which are located in registers. To disable inference, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. You can also disable the option for a specific block with the Assignment Editor. Even if you set the logic option to **On** or **Auto**, the software might not infer small shift registers because small shift registers do not benefit from implementation in dedicated memory. However, you can use the **Allow Any Shift Register Size for Recognition** logic option to instruct synthesis to infer a shift register even when its size is too small.

You can use the **Allow Shift Register Merging across Hierarchies** option to prevent the Compiler from merging shift registers in different hierarchies into one larger shift register. The option has three settings: **On**, **Off**, and **Auto**. The **Auto** setting is the default setting, and the Compiler decides whether or not to merge shift registers across hierarchies. When you turn on this option, the Compiler allows all shift registers to merge across hierarchies, and when you turn off this option, the Compiler does not allow any shift registers to merge across hierarchies. You can set this option globally or on entities or individual nodes.

Note: The registers that the software maps to the RAM-based Shift Register IP core and places in RAM are not available in the Simulator because their node names do not exist after synthesis.

The Compiler turns off the **Auto Shift Register Replacement** logic option when you select a formal verification tool on the **EDA Tool Settings** page. If you do not select a formal verification tool, the Compiler issues a warning and the compilation report lists shift registers that the logic option might infer. To enable an IP core for the shift register in the formal verification flow, you can either instantiate a shift register explicitly with the IP catalog or make the shift register into a black box in a separate entity or module.

Related Information

- Auto Shift Register Replacement logic option
  For more information about the Auto Shift Register Replacement logic option and the supported devices
- RAM-Based Shift Register (ALTSHIFT_TAPS) User Guide
  For more information about the RAM-based Shift Register IP core

3.5.3. RAM and ROM

Use the **Auto RAM Replacement** and **Auto ROM Replacement** logic options to control RAM and ROM inference, respectively. To disable the inference, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.

Note: Although the software implements inferred shift registers in RAM blocks, you cannot turn off the **Auto RAM Replacement** option to disable shift register replacement. Use the **Auto Shift Register Replacement** option.

The software might not infer very small RAM or ROM blocks because you can implement very small memory blocks with the registers in the logic. However, you can use the **Allow Any RAM Size for Recognition** and **Allow Any ROM Size for Recognition** logic options to instruct synthesis to infer a memory block even when its size is too small.

Note: The software turns off the **Auto ROM Replacement** logic option when you select a formal verification tool in the **EDA Tool Settings** page. If you do not select a formal verification tool, the software issues a warning and a report panel provides a list of ROMs that the logic option might infer. To enable an IP core for the shift register in the formal verification flow, you can either instantiate a ROM explicitly using the IP Catalog or create a black box for the ROM in a separate entity or in a separate module.

Although formal verification tools do not support inferred RAM blocks, due to the importance of inferring RAM in many designs, the software turns on the **Auto RAM Replacement** logic option when you select a formal verification tool in the **EDA Tool Settings** page. The software automatically performs black box instance for any module or entity that contains an inferred RAM block. The software issues a warning and lists the black box created in the compilation report. This black box allows formal verification tools to proceed; however, the formal verification tool cannot verify the entire module or entire entity that contains the RAM. Altera recommends that you explicitly instantiate RAM blocks in separate modules or in separate entities so that the formal verification tool can verify as much logic as possible.

Related Information

- Shift Registers on page 161
- Auto RAM Replacement logic option
  For more information about the Auto RAM Replacement logic option and its supported devices
- Auto ROM Replacement logic option
  For more information about the Auto ROM Replacement logic option and its supported devices

3.5.4. Resource Aware RAM, ROM, and Shift-Register Inference

The Quartus Prime Integrated Synthesis considers resource usage when inferring RAM, ROM, and shift registers. During RAM, ROM, and shift register inferencing, synthesis looks at the number of memories available in the current device and does not infer more memory than is available to avoid a no-fit error. Synthesis tries to select the memories that are not inferred in a way that aims at the smallest increase in logic and registers.

Resource aware RAM, ROM and shift register inference is controlled by the **Resource Aware Inference for Block RAM** option. To disable this option for the entire project, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.

When you select the **Auto** setting, resource aware RAM, ROM, and shift register inference use the resource counts from the largest device.

For designs with multiple partitions, Quartus Prime Integrated Synthesis considers one partition at a time. Therefore, for each partition, it assumes that all RAM blocks are available to that partition. If this causes a no-fit error, you can limit the number of RAM blocks available per partition with the **Maximum Number of M512 Memory Blocks**, **Maximum Number of M4K/M9K/M20K/M10K Memory Blocks**, **Maximum Number of M-RAM/M144K Memory Blocks and Maximum Number of LABs** settings in the Assignment Editor. The balancer also uses these options.

3.5.5. Auto RAM to Logic Cell Conversion

The **Auto RAM to Logic Cell Conversion** logic option allows Quartus Prime Integrated Synthesis to convert small RAM blocks to logic cells if the logic cell implementation gives better quality of results. The software converts only single-port or simple-dual port RAMs with no initialization files to logic cells. You can set this option globally or apply it to individual RAM nodes. You can enable this option by turning on the appropriate option for the entire project in the **Advanced Analysis & Synthesis Settings** dialog box.

For Arria GX and Stratix family of devices, the software uses the following rules to determine the placement of a RAM, either in logic cells or a dedicated RAM block:

- If the number of words is less than 16, use a RAM block if the total number of bits is greater than or equal to 64.
- If the number of words is greater than or equal to 16, use a RAM block if the total number of bits is greater than or equal to 32.
- Otherwise, implement the RAM in logic cells.

For the Cyclone family of devices, the software uses the following rules:

- If the number of words is greater than or equal to 64, use a RAM block.
- If the number of words is greater than or equal to 16 and less than 64, use a RAM block if the total number of bits is greater than or equal to 128.
- Otherwise, implement the RAM in logic cells.

Related Information

Auto RAM to Logic Cell Conversion logic option
For more information about the Auto RAM to Logic Cell Conversion logic options and the supported devices

3.5.6. RAM Style and ROM Style—for Inferred Memory

These attributes specify the implementation for an inferred RAM or ROM block. You can specify the type of TriMatrix embedded memory block, or specify the use of standard logic cells (LEs or ALMs). The Quartus Prime software supports the attributes only for device families with TriMatrix embedded memory blocks.

The `ramstyle` and `romstyle` attributes take a single string value. The `M512`, `M4K`, `M-RAM`, `MLAB`, `M9K`, `M144K`, `M20K`, and `M10K` values (as applicable for the target device family) indicate the type of memory block to use for the inferred RAM or ROM. If you set the attribute to a block type that does not exist in the target device family, the software generates a warning and ignores the assignment. The `logic` value indicates that the Quartus Prime software implements the RAM or ROM in regular logic rather than dedicated memory blocks. You can set the attribute on a module or entity, in which case it specifies the default implementation style for all inferred memory

blocks in the immediate hierarchy. You can also set the attribute on a specific signal (VHDL) or variable (Verilog HDL) declaration, in which case it specifies the preferred implementation style for that specific memory, overriding the default implementation style.

Note: If you specify a `logic` value, the memory appears as a RAM or ROM block in the RTL Viewer, but Integrated Synthesis converts the memory to regular logic during synthesis.

In addition to `ramstyle` and `romstyle`, the Quartus Prime software supports the `syn_ramstyle` attribute name for compatibility with other synthesis tools.

These tables specify that you must implement all memory in the module or the `my_memory_blocks` entity with a specific type of block.

Table 31. Applying a `romstyle` Attribute to a Module Declaration

HDL	Code
Verilog-1995	`module my_memory_blocks (...) /* synthesis romstyle = "M4K" */;`

Table 32. Applying a ramstyle Attribute to a Module Declaration

HDL	Code
Verilog-2001 and SystemVerilog	`(* ramstyle = "M512" *) module my_memory_blocks (...);`

Table 33. Applying a romstyle Attribute to an Architecture

HDL	Code

VHDL	`architecture rtl of my_ my_memory_blocks is`
	`attribute romstyle : string;`
	`attribute romstyle of rtl : architecture is "M-RAM";`
	`begin`

These tables specify that you must implement the inferred `my_ram` or `my_rom` memory with regular logic instead of a TriMatrix memory block.

Table 34. Applying a `syn_ramstyle` Attribute to a Variable Declaration

HDL	Code
Verilog-1995	`reg [0:7] my_ram[0:63] /* synthesis syn_ramstyle = "logic" */;`

Table 35. Applying a `romstyle` Attribute to a Variable Declaration

HDL	Code
Verilog-2001 and SystemVerilog	`(* romstyle = "logic" *) reg [0:7] my_rom[0:63];`

Table 36. Applying a `ramstyle` Attribute to a Signal Declaration

HDL	Code

VHDL	`type memory_t is array (0 to 63) of std_logic_vector (0 to 7);`
	`signal my_ram : memory_t;`
	`attribute ramstyle : string;`
	`attribute ramstyle of my_ram : signal is "logic";`

You can control the depth of an inferred memory block and optimize its usage with the `max_depth` attribute. You can also optimize the usage of the memory block with this attribute.

These tables specify the depth of the inferred memory `mem` using the `max_depth` synthesis attribute.

Table 37. Applying a max_depth Attribute to a Variable Declaration

HDL	Code
Verilog-1995	`reg [7:0] mem [127:0] /* synthesis max_depth = 2048 */`

Table 38. Applying a max_depth Attribute to a Variable Declaration

HDL	Code
Verilog-2001 and SystemVerilog	`(* max_depth = 2048*) reg [7:0] mem [127:0];`

Table 39. Applying a max_depth Attribute to a Variable Declaration

HDL	Code

VHDL	`type ram_block is array (0 to 31) of std_logic_vector (2 downto 0);`
	`signal mem : ram_block;`
	`attribute max_depth : natural;`
	`attribute max_depth OF mem : signal is 2048;`

The syntax for setting these attributes in HDL is the same as the syntax for other synthesis attributes, as shown in Synthesis Attributes on page 139.

Related Information

Synthesis Attributes on page 139

3.5.7. RAM Style Attribute—For Shift Registers Inference

The RAM style attribute for shift register allows you to use the RAM style attribute for shift registers, just as you use them for RAM or ROMs. The Quartus Prime Synthesis uses the RAM style attribute during shift register inference. If synthesis infers the shift register to RAM, it will be sent to the requested RAM block type. Shift registers are merged only if the RAM style attributes are compatible. If the RAM style is set to logic, a shift register does not get inferred to RAM.

Table 40. Setting the RAM Style Attribute for Shift Registers

HDL	Code
Verilog	`(* ramstyle = "mlab" *)reg [N-1:0] sr;`

continued...

HDL	Code
VHDL	`attribute ramstyle : string;attribute ramstyle of sr : signal is "M20K";`

3.5.8. Disabling Add Pass-Through Logic to Inferred RAMs no_rw_check Attribute

Use the `no_rw_check` value for the `ramstyle` attribute, or disable the `add_pass_through_logic_to_inferred_rams option` logic option assignment to indicate that your design does not depend on the behavior of the inferred RAM, when there are reads and writes to the same address in the same clock cycle. If you specify the attribute or disbale the logic option, the Quartus Prime software chooses a read-during-write behavior instead of the read-during-write behavior of your HDL source code.

You disable or edit the attributes of this option by modifying the `add_pass_through_logic_to_inferred_rams option` in the Quartus Prime Settings File (`.qsf`). There is no corresponding GUI setting for this option.

Sometimes, you must map an inferred RAM into regular logic cells because the inferred RAM has a read-during-write behavior that the TriMatrix memory blocks in your target device do not support. In other cases, the Quartus Prime software must insert extra logic to mimic read-during-write behavior of the HDL source to increase the area of your design and potentially reduce its performance. In some of these cases, you can use the attribute to specify that the software can implement the RAM directly in a TriMatrix memory block without using logic. You can also use the attribute to prevent a warning message for dual-clock RAMs in the case that the inferred behavior in the device does not exactly match the read-during-write conditions described in the HDL code.

These examples use two addresses and normally require extra logic after the RAM to ensure that the read-during-write conditions in the device match the HDL code. If your design does not require a defined read-during-write condition, the extra logic is not necessary. With the `no_rw_check` attribute, Quartus Prime Integrated Synthesis does not generate the extra logic.

Table 41. Inferred RAM Using `no_rw_check` Attribute

HDL	Code

Verilog HDL	`module ram_infer (q, wa, ra, d, we, clk);`
	`    output [7:0] q;`
	`    input [7:0] d;`
	`    input [6:0] wa;`
	`    input [6:0] ra;`
	`    input we, clk;`
	`    reg [6:0] read_add;`
	`    (* ramstyle = "no_rw_check" *) reg [7:0] mem [127:0];`
	`    always @ (posedge clk) begin`
	`        if (we)`
	`            mem[wa] <= d;`
	`        read_add <= ra;`
	`    end`
	`    assign q = mem[read_add];`
	`endmodule`

VHDL	`LIBRARY ieee;`
	`USE ieee.std_logic_1164.ALL;`
	`ENTITY ram IS`
	`    PORT (`
	`        clock: IN STD_LOGIC;`

continued...

HDL	Code

	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 ram;
	ARCHITECTURE rtl OF ram IS
	TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(2 DOWNTO 0);
	SIGNAL ram_block: MEM;
	ATTRIBUTE ramstyle : string;
	ATTRIBUTE ramstyle of ram_block : signal is "no_rw_check";
	SIGNAL read_address_reg: INTEGER RANGE 0 to 31;
	BEGIN
	PROCESS (clock)
	BEGIN
	IF (clock'event AND clock = '1') 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;

You can use a ramstyle attribute with the MLAB value, so that the Quartus Prime software can infer a small RAM block and place it in an MLAB.

Note: You can use this attribute in cases in which some asynchronous RAM blocks might be coded with read-during-write behavior that does not match the Stratix IV and Stratix V architectures. Thus, the device behavior would not exactly match the behavior that the code describes. If the difference in behavior is acceptable in your design, use the `ramstyle` attribute with the `no_rw_check` value to specify that the software should not check the read-during-write behavior when inferring the RAM. When you set this attribute, Quartus Prime Integrated Synthesis allows the behavior of the output to differ when the asynchronous read occurs on an address that had a write on the most recent clock edge. That is, the functional HDL simulation results do not match the hardware behavior if you write to an address that is being read. To include these attributes, set the value of the ramstyle attribute to `MLAB, no_rw_check`.

These examples show the method of setting two values to the `ramstyle` attribute with a small asynchronous RAM block, with the ramstyle synthesis attribute set, so that the software can implement the memory in the MLAB memory block and so that the read-during-write behavior is not important. Without the attribute, this design requires 512 registers and 240 ALUTs. With the attribute, the design requires eight memory ALUTs and only 15 registers.

Table 42. Inferred RAM Using `no_rw_check` and MLAB Attributes

HDL	Code

Verilog HDL	module async_ram (
	input [5:0] addr,
	input [7:0] data_in,
	input clk,
	input write,
	output [7:0] data_out);
	(* ramstyle = "MLAB, no_rw_check" *) reg [7:0] mem[0:63];
	assign data_out = mem[addr];
	always @ (posedge clk)
	begin
	if (write)
	mem[addr] = data_in;
	end
	endmodule

continued...

HDL	Code

VHDL	`LIBRARY ieee;`
	`USE ieee.std_logic_1164.ALL;`
	`ENTITY 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 ram;`
	`ARCHITECTURE rtl OF ram IS`
	`  TYPE MEM IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(2 DOWNTO 0);`
	`  SIGNAL ram_block: MEM;`
	`  ATTRIBUTE ramstyle : string;`
	`  ATTRIBUTE ramstyle of ram_block : signal is "MLAB , no_rw_check";`
	`  SIGNAL read_address_reg: INTEGER RANGE 0 to 31;`
	`BEGIN`
	`  PROCESS (clock)`
	`  BEGIN`
	`  IF (clock'event AND clock = '1') 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;`

Related Information

Add Pass-Through Logic to Inferred RAMs logic option
For more information about the Add Pass-Through Logic to Inferred RAMs logic option and the supported devices

3.5.9. RAM Initialization File—for Inferred Memory

The `ram_init_file` attribute specifies the initial contents of an inferred memory with a **.mif**. The attribute takes a string value containing the name of the RAM initialization file.

The `ram_init_file` attribute is supported for ROM too.

Table 43. Applying a ram_init_file Attribute

HDL	Code
Verilog-1995	`reg [7:0] mem[0:255] /* synthesis ram_init_file` `= " my_init_file.mif" */;`
Verilog-2001	`(* ram_init_file = "my_init_file.mif" *) reg [7:0] mem[0:255];`

VHDL(6)	`type mem_t is array(0 to 255) of unsigned(7 downto 0);`
	`signal ram : mem_t;`
	`attribute ram_init_file : string;`
	`attribute ram_init_file of ram :`
	`signal is "my_init_file.mif";`

(6) You can also initialize the contents of an inferred memory by specifying a default value for the corresponding signal. In Verilog HDL, you can use an initial block to specify the memory contents. Quartus Prime Integrated Synthesis automatically converts the default value into a **.mif** for the inferred RAM.

3.5.10. Multiplier Style—for Inferred Multipliers

The `multstyle` attribute specifies the implementation style for multiplication operations (*) in your HDL source code. You can use this attribute to specify whether you prefer the Compiler to implement a multiplication operation in general logic or dedicated hardware, if available in the target device.

The `multstyle` attribute takes a string value of `"logic"` or `"dsp"`, indicating a preferred implementation in logic or in dedicated hardware, respectively. In Verilog HDL, apply the attribute to a module declaration, a variable declaration, or a specific binary expression that contains the * operator. In VHDL, apply the synthesis attribute to a signal, variable, entity, or architecture.

Note: Specifying a `multstyle` of `"dsp"` does not guarantee that the Quartus Prime software can implement a multiplication in dedicated DSP hardware. The final implementation depends on several conditions, including the availability of dedicated hardware in the target device, the size of the operands, and whether or not one or both operands are constant.

In addition to `multstyle`, the Quartus Prime software supports the `syn_multstyle` attribute name for compatibility with other synthesis tools.

When applied to a Verilog HDL module declaration, the attribute specifies the default implementation style for all instances of the * operator in the module. For example, in the following code examples, the `multstyle` attribute directs the Quartus Prime software to implement all multiplications inside module `my_module` in the dedicated multiplication hardware.

Table 44. Applying a `multstyle` Attribute to a Module Declaration

HDL	Code
Verilog-1995	`module my_module (...) /* synthesis multstyle = "dsp" */;`
Verilog-2001	`(* multstyle = "dsp" *) module my_module(...);`

When applied to a Verilog HDL variable declaration, the attribute specifies the implementation style for a multiplication operator, which has a result directly assigned to the variable. The attribute overrides the `multstyle` attribute with the enclosing module, if present.

In these examples, the `multstyle` attribute applied to variable `result` directs the Quartus Prime software to implement `a * b` in logic rather than the dedicated hardware.

Table 45. Applying a `multstyle` Attribute to a Variable Declaration

HDL	Code

Verilog-2001	`wire [8:0] a, b;`
	`(* multstyle = "logic" *) wire [17:0] result;`
	`assign result = a * b;  //Multiplication must be`
	`//directly assigned to result`

continued...

HDL	Code

Verilog-1995	`wire [8:0] a, b;`
	`wire [17:0] result /* synthesis multstyle = "logic" */;`
	`assign result = a * b;  //Multiplication must be`
	`//directly assigned to result`

When applied directly to a binary expression that contains the * operator, the attribute specifies the implementation style for that specific operator alone and overrides any `multstyle` attribute with the target variable or enclosing module.

In this example, the `multstyle` attribute indicates that you must implement `a * b` in the dedicated hardware.

Table 46. Applying a `multstyle` Attribute to a Binary Expression

HDL	Code

Verilog-2001	`wire [8:0] a, b;`
	`wire [17:0] result;`
	`assign result = a * (* multstyle = "dsp" *) b;`

Note: You cannot use Verilog-1995 attribute syntax to apply the `multstyle` attribute to a binary expression.

When applied to a VHDL entity or architecture, the attribute specifies the default implementation style for all instances of the * operator in the entity or architecture.

In this example, the `multstyle` attribute directs the Quartus Prime software to use dedicated hardware, if possible, for all multiplications inside architecture `rtl` of entity `my_entity`.

Table 47. Applying a `multstyle` Attribute to an Architecture

HDL	Code

VHDL	`architecture rtl of my_entity is`
	`attribute multstyle : string;`
	`attribute multstyle of rtl : architecture is "dsp";`
	`begin`

When applied to a VHDL signal or variable, the attribute specifies the implementation style for all instances of the * operator, which has a result directly assigned to the signal or variable. The attribute overrides the `multstyle` attribute with the enclosing entity or architecture, if present.

In this example, the `multstyle` attribute associated with signal `result` directs the Quartus Prime software to implement `a * b` in logic rather than the dedicated hardware.

Table 48. Applying a `multstyle` Attribute to a Signal or Variable

HDL	Code

VHDL	`signal a, b : unsigned(8 downto 0);`
	`signal result : unsigned(17 downto 0);`
	
	`attribute multstyle : string;`
	`attribute multstyle of result : signal is "logic";`
	`result <= a * b;`

3.5.11. Full Case Attribute

A Verilog HDL case statement is full when its case items cover all possible binary values of the case expression or when a default case statement is present. A `full_case` attribute attached to a case statement header that is not full forces synthesis to treat the unspecified states as a don't care value. VHDL case statements must be full, so the attribute does not apply to VHDL.

Using this attribute on a case statement that is not full allows you to avoid the latch inference problems.

Note: Latches have limited support in formal verification tools. Do not infer latches unintentionally, for example, through an incomplete case statement when using formal verification.

Formal verification tools support the `full_case` synthesis attribute (with limited support for attribute syntax, as described in Synthesis Attributes on page 139).

Using the `full_case` attribute might cause a simulation mismatch between the Verilog HDL functional and the post-Quartus Prime simulation because unknown case statement cases can still function as latches during functional simulation. For example, a simulation mismatch can occur with the code in Table 49 on page 171 when `sel` is `2'b11` because a functional HDL simulation output behaves as a latch and the Quartus Prime simulation output behaves as a don't care value.

Note: Altera recommends making the case statement "full" in your regular HDL code, instead of using the `full_case` attribute.

Table 49. A full_case Attribute

The case statement in this example is not full because you do not specify some `sel` binary values. Because you use the `full_case` attribute, synthesis treats the output as "don't care" when the `sel` input is `2'b11`.

HDL	Code

Verilog HDL	`module full_case (a, sel, y);`
	`  input [3:0] a;`
	`  input [1:0] sel;`
	`  output y;`
	`  reg y;`
	`  always @ (a or sel)`
	`  case (sel)  // synthesis full_case`
	`  2'b00: y=a[0];`
	`  2'b01: y=a[1];`
	`  2'b10: y=a[2];`
	`  endcase`
	`endmodule`

Verilog-2001 syntax also accepts the statements in Table 50 on page 171 in the `case` header instead of the comment form as shown in Table 49 on page 171.

Table 50. Syntax for the full_case Attribute

HDL	Syntax
Verilog-2001	`(* full_case *) case (sel)`

Related Information

Synthesis Attributes on page 139

3.5.12. Parallel Case

The `parallel_case` attribute indicates that you must consider a Verilog HDL case statement as parallel; that is, you can match only one case item at a time. Case items in Verilog HDL case statements might overlap. To resolve multiple matching case items, the Verilog HDL language defines a priority among case items in which the case statement always executes the first case item that matches the case expression value. By default, the Quartus Prime software implements the extra logic necessary to satisfy this priority relationship.

Attaching a `parallel_case` attribute to a case statement header allows the Quartus Prime software to consider its case items as inherently parallel; that is, at most one case item matches the case expression value. Parallel case items simplify the generated logic.

In VHDL, the individual choices in a case statement might not overlap, so they are always parallel and this attribute does not apply.

Altera recommends that you use this attribute only when the `case` statement is truly parallel. If you use the attribute in any other situation, the generated logic does not match the functional simulation behavior of the Verilog HDL.

Note: Altera recommends that you avoid using the `parallel_case` attribute, because you may mismatch the Verilog HDL functional and the post-Quartus Prime simulation.

If you specify SystemVerilog-2005 as the supported Verilog HDL version for your design, you can use the SystemVerilog keyword `unique` to achieve the same result as the `parallel_case` directive without causing simulation mismatches.

This example shows a `casez` statement with overlapping case items. In functional HDL simulation, the software prioritizes the three case items by the bits in `sel`. For example, `sel[2]` takes priority over `sel[1]`, which takes priority over `sel[0]`. However, the synthesized design can simulate differently because the `parallel_case` attribute eliminates this priority. If more than one bit of `sel` is high, more than one output (`a`, `b`, or `c`) is high as well, a situation that cannot occur in functional HDL simulation.

Table 51. A `parallel_case` Attribute

HDL	Code

Verilog HDL	`module parallel_case (sel, a, b, c);`
	`  input [2:0] sel;`
	`  output a, b, c;`
	`  reg a, b, c;`
	`  always @ (sel)`
	`  begin`
	`  {a, b, c} = 3'b0;`
	`  casez (sel)  // synthesis parallel_case`
	`  3'b1??: a = 1'b1;`
	`  3'b?1?: b = 1'b1;`
	`  3'b??1: c = 1'b1;`
	`  endcase`
	`  end`
	`endmodule`

Table 52. Verilog-2001 Syntax

Verilog-2001 syntax also accepts the statements as shown in the following table in the case (or casez) header instead of the comment form, as shown in Table 51 on page 172.

HDL	Syntax
Verilog-2001	`(* parallel_case *) casez (sel)`

3.5.13. Translate Off and On / Synthesis Off and On

The `translate_off` and `translate_on` synthesis directives indicate whether the Quartus Prime software or a third-party synthesis tool should compile a portion of HDL code that is not relevant for synthesis. The `translate_off` directive marks the beginning of code that the synthesis tool should ignore; the `translate_on` directive indicates that synthesis should resume. You can also use the `synthesis_on` and `synthesis_off` directives as a synonym for translate on and off.

You can use these directives to indicate a portion of code for simulation only. The synthesis tool reads synthesis-specific directives and processes them during synthesis; however, third-party simulation tools read the directives as comments and ignore them.

These examples show these directives.

Table 53. Translate Off and On

HDL	Code

Verilog HDL	`// synthesis translate_off`
	`parameter tpd = 2;  // Delay for simulation`
	`#tpd;`
	`// synthesis translate_on`

VHDL	`-- synthesis translate_off`
	`use std.textio.all;`
	`-- synthesis translate_on`

VHDL 2008	`/* synthesis translate_off */`
	`use std.textio.all;`
	`/* synthesis translate_on */`

If you want to ignore only a portion of code in Quartus Prime Integrated Synthesis, you can use the Altera-specific attribute keyword `altera`. For example, use the `// altera translate_off` and `// altera translate_on` directives to direct Quartus Prime Integrated Synthesis to ignore a portion of code that you intend only for other synthesis tools.

3.5.14. Ignore translate_off and synthesis_off Directives

The **Ignore translate_off and synthesis_off Directives** logic option directs Quartus Prime Integrated Synthesis to ignore the `translate_off` and `synthesis_off` directives. Turning on this logic option allows you to compile code that you want the third-party synthesis tools to ignore; for example, IP core declarations that the other tools treat as black boxes but the Quartus Prime software can compile. To set the **Ignore translate_off and synthesis_off Directives** logic option, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.

Related Information

Ignore translate_off and synthesis_off Directives logic option
For more information about the **Ignore translate_off and synthesis_off Directives** logic option and the supported devices

3.5.15. Read Comments as HDL

The `read_comments_as_HDL` synthesis directive indicates that the Quartus Prime software should compile a portion of HDL code that you commented out. This directive allows you to comment out portions of HDL source code that are not relevant for simulation, while instructing the Quartus Prime software to read and synthesize that same source code. Setting the `read_comments_as_HDL` directive to `on` indicates the beginning of commented code that the synthesis tool should read; setting the `read_comments_as_HDL` directive to `off` indicates the end of the code.

Note: You can use this directive with `translate_off` and `translate_on` to create one HDL source file that includes an IP core instantiation for synthesis and a behavioral description for simulation.

Formal verification tools do not support the `read_comments_as_HDL` directive because the tools do not recognize the directive.

In these examples, the Compiler synthesizes the commented code enclosed by `read_comments_as_HDL` because the directive is visible to the Quartus Prime Compiler. VHDL 2008 allows block comments, which comments are also supported for synthesis directives.

Note: Because synthesis directives are case sensitive in Verilog HDL, you must match the case of the directive, as shown in the following examples.

Table 54. Read Comments as HDL

HDL	Code

Verilog HDL	`// synthesis read_comments_as_HDL on`
	`// my_rom lpm_rom (.address (address),`
	`//  .data (data));`
	`// synthesis read_comments_as_HDL off`

VHDL	`-- synthesis read_comments_as_HDL on`
	`-- my_rom : entity lpm_rom`
	`--  port map (`
	`--  address => address,`
	`--  data => data, );`
	`-- synthesis read_comments_as_HDL off`

VHDL 2008	`/* synthesis read_comments_as_HDL on */`
	`/* my_rom : entity lpm_rom`
	`  port map (`
	`  address => address,`
	`  data => data, ); */`
	`  synthesis read_comments_as_HDL off */`

3.5.16. Use I/O Flipflops

The `useioff` attribute directs the Quartus Prime software to implement input, output, and output enable flipflops (or registers) in I/O cells that have fast, direct connections to an I/O pin, when possible. To improve I/O performance by minimizing setup, clock-to-output, and clock-to-output enable times, you can apply the `useioff` synthesis

attribute. The **Fast Input Register**, **Fast Output Register**, and **Fast Output Enable Register** logic options support this synthesis attribute. You can also set this synthesis attribute in the Assignment Editor.

The `useioff` synthesis attribute takes a boolean value. You can apply the value only to the port declarations of a top-level Verilog HDL module or VHDL entity (it is ignored if applied elsewhere). Setting the value to `1` (Verilog HDL) or TRUE (VHDL) instructs the Quartus Prime software to pack registers into I/O cells. Setting the value to `0` (Verilog HDL) or FALSE (VHDL) prevents register packing into I/O cells.

In Table 55 on page 175 and Table 56 on page 175, the `useioff` synthesis attribute directs the Quartus Prime software to implement the `a_reg`, `b_reg`, and `o_reg` registers in the I/O cells corresponding to the `a`, `b`, and `o` ports, respectively.

Table 55. Verilog HDL Code: The `useioff` Attribute

HDL	Code

Verilog HDL	`module top_level(clk, a, b, o);`
	`  input clk;`
	`  input [1:0] a, b /* synthesis useioff = 1 */;`
	`  output [2:0] o /* synthesis useioff = 1 */;`
	`  reg [1:0] a_reg, b_reg;`
	`  reg [2:0] o_reg;`
	`  always @ (posedge clk)`
	`  begin`
	`  a_reg <= a;`
	`  b_reg <= b;`
	`  o_reg <= a_reg + b_reg;`
	`  end`
	`  assign o = o_reg;`
	`endmodule`

Table 56 on page 175 and Table 57 on page 175 show that the Verilog-2001 syntax also accepts the type of statements instead of the comment form in Table 55 on page 175.

Table 56. Verilog-2001 Code: the useioff Attribute

HDL	Code
Verilog-2001	`(* useioff = 1 *)  input [1:0] a, b;` `(* useioff = 1 *)  output [2:0] o;`

Table 57. VHDL Code: the useioff Attribute

HDL	Code

VHDL	`library ieee;`
	`use ieee.std_logic_1164.all;`
	`use ieee.numeric_std.all;`
	`entity useioff_example is`
	`  port (`
	`  clk  : in  std_logic;`
	`  a, b : in  unsigned(1 downto 0);`
	`  o  : out unsigned(1 downto 0));`
	`  attribute useioff : boolean;`
	`  attribute useioff of a : signal is true;`
	`  attribute useioff of b : signal is true;`
	`  attribute useioff of o : signal is true;`
	`end useioff_example;`
	`architecture rtl of useioff_example is`
	`  signal o_reg, a_reg, b_reg : unsigned(1 downto 0);`
	`begin`
	`  process(clk)`
	`  begin`
	`  if (clk = '1' AND clk'event) then`
	`  a_reg <= a;`
	`  b_reg <= b;`
	`  o_reg <= a_reg + b_reg;`

HDL	Code

	``
	      end if;
	   end process;
	o <= o_reg;
	end rtl;
	``

3.5.17. Specifying Pin Locations with chip_pin

The `chip_pin` attribute allows you to assign pin locations in your HDL source. You can use the attribute only on the ports of the top-level entity or module in your design. You can assign pins only to single-bit or one-dimensional bus ports in your design.

For single-bit ports, the value of the `chip_pin` attribute is the name of the pin on the target device, as specified by the pin table of the device.

Note: In addition to the `chip_pin` attribute, the Quartus Prime software supports the `altera_chip_pin_lc` attribute name for compatibility with other synthesis tools. When using this attribute in other synthesis tools, some older device families require an "@" symbol in front of each pin assignment. In the Quartus Prime software, the "@" is optional.

Table 58. Applying Chip Pin to a Single Pin

These examples in this table show different ways of assigning `my_pin1` to Pin C1 and `my_pin2` to Pin 4 on a different target device.

HDL	Code
Verilog-1995	``` input my_pin1 /* synthesis chip_pin = "C1" */; input my_pin2 /* synthesis altera_chip_pin_lc = "@4" */; ```
Verilog-2001	``` (* chip_pin = "C1" *) input my_pin1; (* altera_chip_pin_lc = "@4" *) input my_pin2; ```
VHDL	``` entity my_entity is port(my_pin1: in std_logic; my_pin2: in std_logic;…); end my_entity; attribute chip_pin : string; attribute altera_chip_pin_lc : string; attribute chip_pin of my_pin1 : signal is "C1"; attribute altera_chip_pin_lc of my_pin2 : signal is "@4"; ```

For bus I/O ports, the value of the chip pin attribute is a comma-delimited list of pin assignments. The order in which you declare the range of the port determines the mapping of assignments to individual bits in the port. To leave a bit unassigned, leave its corresponding pin assignment blank.

Table 59. Applying Chip Pin to a Bus of Pins

The example in this table assigns `my_pin[2]` to `Pin_4`, `my_pin[1]` to `Pin_5`, and `my_pin[0]` to `Pin_6`.

HDL	Code
Verilog-1995	``` input [2:0] my_pin /* synthesis chip_pin = "4, 5, 6" */; ```

Table 60. Applying Chip Pin to Part of a Bus

The example in this table reverses the order of the signals in the bus, assigning `my_pin[0]` to `Pin_4` and `my_pin[2]` to `Pin_6` but leaves `my_pin[1]` unassigned.

HDL	Code
Verilog-1995	`input [0:2]  my_pin /* synthesis chip_pin = "4, ,6" */;`

Table 61. Applying Chip Pin to Part of a Bus of Pins

The example in this table assigns `my_pin[2]` to Pin 4 and `my_pin[0]` to Pin 6, but leaves `my_pin[1]` unassigned.

HDL	Code

VHDL	`entity my_entity is`
	`port(my_pin: in std_logic_vector(2 downto 0);…);`
	`end my_entity;`
	`attribute chip_pin of my_pin: signal is "4, , 6";`

Table 62. VHDL and Verilog-2001 Examples: Assigning Pin Location and I/O Standard

HDL	Code

VHDL	`attribute altera_chip_pin_lc: string;`
	`attribute  altera_attribute: string;`
	`attribute altera_chip_pin_lc of clk: signal is "B13";`
	`attribute altera_attribute of clk:signal is "-name IO_STANDARD ""3.3-V LVCMOS""";`

Verilog-2001	`(* altera_attribute = "-name IO_STANDARD \"3.3-V LVCMOS\"" *)(* chip_pin = "L5" *)input clk;`
	`(* altera_attribute = "-name IO_STANDARD LVDS" *)(* chip_pin = "L4" *)input sel;`
	`output [3:0] data_o, input [3:0] data_i);`

3.5.18. Using altera_attribute to Set Quartus Prime Logic Options

The `altera_attribute` attribute allows you to apply Quartus Prime logic options and assignments to an object in your HDL source code. You can set this attribute on an entity, architecture, instance, register, RAM block, or I/O pin. You cannot set it on an arbitrary combinational node such as a net. With `altera_attribute`, you can control synthesis options from your HDL source even when the options lack a specific HDL synthesis attribute. You can also use this attribute to pass entity-level settings and assignments to phases of the Compiler flow that follow Analysis & Synthesis, such as Fitting.

Assignments or settings made through the Quartus Prime software, the **.qsf**, or the Tcl interface take precedence over assignments or settings made with the `altera_attribute` synthesis attribute in your HDL code.

The attribute value is a single string containing a list of **.qsf** variable assignments separated by semicolons:

```
-name <variable_1> <value_1>;-name <variable_2> <value_2>[;…]
```

If the Quartus Prime option or assignment includes a target, source, and section tag, you must use the syntax in this example for each **.qsf** variable assignment:

```
-name <variable> <value>
-from <source> -to <target> -section_id <section>
```

This example shows the syntax for the full attribute value, including the optional target, source, and section tags for two different **.qsf** assignments:

```
" -name <variable_1> <value_1> [-from <source_1>] [-to <target_1>] [-section_id
\ <section_1>]; -name <variable_2> <value_2> [-from <source_2>] [-to <target_2>]
\
[-section_id <section_2>] "
```

Table 63. Example Usage

If the assigned value of a variable is a string of text, you must use escaped quotes around the value in Verilog HDL or double-quotes in VHDL:

HDL	Code
Assigned Value of a Variable in Verilog HDL (With Nonexistent Variable and Value Terms)	`"VARIABLE_NAME \"STRING_VALUE\""`
Assigned Value of a Variable in VHDL (With Nonexistent Variable and Value Terms)	`"VARIABLE_NAME ""STRING_VALUE"""`

To find the **.qsf** variable name or value corresponding to a specific Quartus Prime option or assignment, you can set the option setting or assignment in the Quartus Prime software, and then make the changes in the **.qsf**.

Applying altera_attribute to an Instance

These examples use `altera_attribute` to set the power-up level of an inferred register.

Table 64. Applying altera_attribute to an Instance

These examples use `altera_attribute` to set the power-up level of an inferred register.

HDL	Code
Verilog-1995	`reg my_reg /* synthesis altera_attribute = "-name POWER_UP_LEVEL HIGH" */;`
Verilog-2001	`(* altera_attribute = "-name POWER_UP_LEVEL HIGH" *) reg my_reg;`

VHDL	`signal my_reg : std_logic;`
	`attribute altera_attribute : string;`
	`attribute altera_attribute of my_reg: signal is "-name POWER_UP_LEVEL HIGH";`

Note: For inferred instances, you cannot apply the attribute to the instance directly. Therefore, you must apply the attribute to one of the output nets of the instance. The Quartus Prime software automatically moves the attribute to the inferred instance.

Applying altera_attribute to an Entity

These examples use the `altera_attribute` to disable the **Auto Shift Register Replacement** synthesis option for an entity. To apply the Altera Attribute to a VHDL entity, you must set the attribute on its architecture rather than on the entity itself.

Table 65. Applying altera_attribute to an Entity

HDL	Code
Verilog-1995	`module my_entity(…) /* synthesis altera_attribute = "-name AUTO_SHIFT_REGISTER_RECOGNITION OFF" */;`

continued...

HDL	Code
Verilog-2001	`(* altera_attribute = "-name AUTO_SHIFT_REGISTER_RECOGNITION OFF" *) module my_entity(…) ;`

VHDL	`entity my_entity is`
	`-- Declare generics and ports`
	`end my_entity;`
	`architecture rtl of my_entity is`
	`attribute altera_attribute : string;`
	`-- Attribute set on architecture, not entity`
	`attribute altera_attribute of rtl: architecture is "-name AUTO_SHIFT_REGISTER_RECOGNITION OFF";`
	`begin`
	`-- The architecture body`
	`end rtl;`

Applying altera_attribute with the -to Option

You can also use `altera_attribute` for more complex assignments that have more than one instance. In Table 66 on page 179, the `altera_attribute` cuts all timing paths from `reg1` to `reg2`, equivalent to this Tcl or **.qsf** command, as shown in the example below:

```
set_instance_assignment -name CUT ON -from reg1 -to reg2
```

Table 66. Applying altera_attribute with the -to Option

HDL	Code
Verilog-1995	`reg reg2;` `reg reg1 /* synthesis altera_attribute = "-name CUT ON -to reg2" */;`
Verilog-2001 and SystemVerilog	`reg reg2;` `(* altera_attribute = "-name CUT ON -to reg2" *) reg reg1;`

VHDL	`signal reg1, reg2 : std_logic;`
	`attribute altera_attribute: string;`
	`attribute altera_attribute of reg1 : signal is "-name CUT ON -to reg2";`

You can specify either the `-to` option or the `-from` option in a single `altera_attribute`; Integrated Synthesis automatically sets the remaining option to the target of the `altera_attribute`. You can also specify wildcards for either option. For example, if you specify "*" for the `-to` option instead of `reg2` in these examples, the Quartus Prime software cuts all timing paths from `reg1` to every other register in this design entity.

You can use the `altera_attribute` only for entity-level settings, and the assignments (including wildcards) apply only to the current entity.

Related Information

- Synthesis Attributes on page 139
- Quartus Prime Settings File Manual
  Lists all variable names

3.6. Analyzing Synthesis Results

After performing synthesis, you can check your synthesis results in the **Analysis & Synthesis** section of the Compilation Report and the Project Navigator.

3.6.1. Analysis & Synthesis Section of the Compilation Report

The Compilation Report, which provides a summary of results for the project, appears after a successful compilation. After Analysis & Synthesis, the Summary section of the Compilation Report provides a summary of utilization based on synthesis data, before Fitter optimizations have occurred. The **Analysis & Synthesis** section lists synthesis-specific information.

Analysis & Synthesis includes various report sections, including a list of the source files read for the project, the resource utilization by entity after synthesis, and information about state machines, latches, optimization results, and parameter settings.

Related Information

Analysis Synthesis Summary Reports
For more information about each report section

3.6.2. Project Navigator

The **Hierarchy** tab of the Project Navigator provides a view of the project hierarchy and a summary of resource and device information about the current project. After Analysis & Synthesis, before the Fitter begins, the Project Navigator provides a summary of utilization based on synthesis data, before Fitter optimizations have occurred.

If an entity in the Hierarchy tab contains parameter settings, a tooltip displays the settings when you hold the pointer over the entity.

3.6.2.1. Upgrade IP Components Dialog Box

In the Quartus Prime software version 12.1 SP1 and later, the **Upgrade IP Components** dialog box allows you to upgrade all outdated IP in your project after you move to a newer version of the Quartus Prime software.

Related Information

Upgrade IP Components dialog box
For more information about the Upgrade IP Components dialog box

3.7. Analyzing and Controlling Synthesis Messages

You can analyze the generated messages during synthesis and control which messages appear during compilation.

3.7.1. Quartus Prime Messages

The messages that appear during Analysis & Synthesis describe many of the optimizations during the synthesis stage, and provide information about how the software interprets your design. Altera recommends checking the messages to analyze **Critical Warnings** and **Warnings**, because these messages can relate to important design problems. Read the **Info** messages to get more information about how the software processes your design.

The software groups the messages by following types: **Info**, **Warning**, **Critical Warning**, and **Error**.

You can specify the type of Analysis & Synthesis messages that you want to view by selecting the **Analysis & Synthesis Message Level** option. To specify the display level, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**

Related Information

About the Messages Window
For more information about the Messages window and message suppression

3.7.2. VHDL and Verilog HDL Messages

The Quartus Prime software issues a variety of messages when it is analyzing and elaborating the Verilog HDL and VHDL files in your design. These HDL messages are a subset of all Quartus Prime messages that help you identify potential problems early in the design process.

HDL messages fall into the following categories:

- **Info message**—lists a property of your design.
- **Warning message**—indicates a potential problem in your design. Potential problems come from a variety of sources, including typos, inappropriate design practices, or the functional limitations of your target device. Though HDL warning messages do not always identify actual problems, Altera recommends investigating code that generates an HDL warning. Otherwise, the synthesized behavior of your design might not match your original intent or its simulated behavior.
- **Error message**—indicates an actual problem with your design. Your HDL code can be invalid due to a syntax or semantic error, or it might not be synthesizable as written.

In this example, the sensitivity list contains multiple copies of the variable `i`. While the Verilog HDL language does not prohibit duplicate entries in a sensitivity list, it is clear that this design has a typing error: Variable `j` should be listed on the sensitivity list to avoid a possible simulation or synthesis mismatch.

```
//dup.v
module dup(input i, input j, output reg o);
always @ (i or i)
    o = i & j;
endmodule
```

When processing the HDL code, the Quartus Prime software generates the following warning message.

```
Warning: (10276) Verilog HDL sensitivity list warning at dup.v(2): sensitivity
list contains multiple entries for "i".
```

In Verilog HDL, variable names are case sensitive, so the variables `my_reg` and `MY_REG` below are two different variables. However, declaring variables that have names in different cases is confusing, especially if you use VHDL, in which variables are not case sensitive.

```
// namecase.v
module namecase (input i, output o);
    reg my_reg;
```

```
    reg MY_REG;
    assign o = i;
endmodule
```

When processing the HDL code, the Quartus Prime software generates the following informational message:

```
Info: (10281) Verilog HDL information at namecase.v(3): variable name "MY_REG"
and variable name "my_reg" should not differ only in case.
```

In addition, the Quartus Prime software generates additional HDL info messages to inform you that this small design does not use neither `my_reg` nor `MY_REG`:

```
Info: (10035) Verilog HDL or VHDL information at namecase.v(3): object "my_reg"
declared but not used
Info: (10035) Verilog HDL or VHDL information at namecase.v(4): object "MY_REG"
declared but not used
```

The Quartus Prime software allows you to control how many HDL messages you can view during the Analysis & Elaboration of your design files. You can set the HDL Message Level to enable or disable groups of HDL messages, or you can enable or disable specific messages.

Related Information

Synthesis Directives on page 141
For more information about synthesis directives and their syntax

3.7.2.1. Setting the HDL Message Level

The HDL Message Level specifies the types of messages that the Quartus Prime software displays when it is analyzing and elaborating your design files.

Table 67. HDL Info Message Level

Level	Purpose	Description
Level1	High-severity messages only	If you want to view only the HDL messages that identify likely problems with your design, select Level1. When you select Level1, the Quartus Prime software issues a message only if there is an actual problem with your design.
Level2	High-severity and medium-severity messages	If you want to view additional HDL messages that identify possible problems with your design, select Level2. Level2 is the default setting.
Level3	All messages, including low-severity messages	If you want to view all HDL info and warning messages, select Level3. This level includes extra "LINT" messages that suggest changes to improve the style of your HDL code.

You must address all issues reported at the **Level1** setting. The default HDL message level is **Level2**.

To set the HDL Message Level in the Quartus Prime software, follow these steps:

1. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**
2. Set the necessary message level from the pull-down menu in the **HDL Message Level** list, and then click **OK.**

   You can override this default setting in a source file with the `message_level synthesis` directive, which takes the values `level1`, `level2`, and `level3`, as shown in the following table.

Table 68. HDL Examples of message_level Directive

HDL	Code

Verilog HDL	`// altera message_level level1`
	`or`
	`/* altera message_level level3 */`

VHDL	`-- altera message_level level2`

A `message_level` synthesis directive remains effective until the end of a file or until the next `message_level` directive. In VHDL, you can use the `message_level` synthesis directive to set the HDL Message Level for entities and architectures, but not for other design units. An HDL Message Level for an entity applies to its architectures, unless overridden by another `message_level` directive. In Verilog HDL, you can use the `message_level` directive to set the HDL Message Level for a module.

3.7.2.2. Enabling or Disabling Specific HDL Messages by Module/Entity

Message ID is in parentheses at the beginning of the message. Use the Message ID to enable or disable a specific HDL info or warning message. Enabling or disabling a specific message overrides its HDL Message Level. This method is different from the message suppression in the Messages window because you can disable messages for a specific module or a specific entity. This method applies only to the HDL messages, and if you disable a message with this method, the Quartus Prime software lists the message as a suppressed message.

To disable specific HDL messages in the Quartus Prime software, follow these steps:

1. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.
2. In the **Advanced Message Settings** dialog box, add the Message IDs you want to enable or disable.

   To enable or disable specific HDL messages in your HDL, use the `message_on` and `message_off` synthesis directives. These directives require a space-separated list of Message IDs. You can enable or disable messages with these synthesis directives immediately before Verilog HDL modules, VHDL entities, or VHDL architectures. You cannot enable or disable a message during an HDL construct.

   A message enabled or disabled via a `message_on` or `message_off` synthesis directive overrides its HDL Message Level or any `message_level` synthesis directive. The message remains disabled until the end of the source file or until you use another `message_on` or `message_off` directive to change the status of the message.

Table 69. HDL message_off Directive for Message with ID 10000

HDL	Code

Verilog HDL	`// altera message_off 10000`
	`or`
	`/* altera message_off 10000 */`

VHDL	`-- altera message_off 10000`

3.8. Node-Naming Conventions in Quartus Prime Integrated Synthesis

Whenever possible, Quartus Prime Integrated Synthesis uses wire or signal names from your source code to name nodes such as LEs or ALMs. Some nodes, such as registers, have predictable names that do not change when a design is resynthesized, although certain optimizations can affect register names. The names of other nodes, particularly LEs or ALMs that contain only combinational logic, can change due to logic optimizations that the software performs.

3.8.1. Hierarchical Node-Naming Conventions

To make each name in your design unique, the Quartus Prime software adds the hierarchy path to the beginning of each name. The "|" separator indicates a level of hierarchy. For each instance in the hierarchy, the software adds the entity name and the instance name of that entity, with the ":" separator between each entity name and its instance name. For example, if a design defines entity A with the name `my_A_inst`, the hierarchy path of that entity would be `A:my_A_inst`. You can obtain the full name of any node by starting with the hierarchical instance path, followed by a "|", and ending with the node name inside that entity.

This example shows you the convention:

```
<entity 0>:<instance_name 0>|<entity 1>:<instance_name 1>|...|<instance_name n>|
<node_name>
```

For example, if entity A contains a register (DFF atom) called `my_dff`, its full hierarchy name would be `A:my_A_inst|my_dff`.

To instruct the Compiler to generate node names that do not contain entity names, on the **Compilation Process Settings** page of the **Settings** dialog box, click **More Settings**, and then turn off D**isplay entity name for node name**.

With this option turned off, the node names use the convention in shown in this example:

```
<instance_name 0>|<instance_name 1>|...|<instance_name n> |<node_name>
```

3.8.2. Node-Naming Conventions for Registers (DFF or D Flipflop Atoms)

In Verilog HDL and VHDL, inferred registers use the names of the `reg` or `signal` connected to the output.

Table 70. HDL Example of a Register that Creates `my_dff_out` DFF Primitive

HDL Register	Code

Verilog HDL	`wire dff_in, my_dff_out, clk;`
	`always @ (posedge clk)`
	`my_dff_out <= dff_in;`

VHDL	`signal dff_in, my_dff_out, clk;`
	`process (clk)`
	`begin`
	`if (rising_edge(clk)) then`
	`my_dff_out <= dff_in;`
	`end if;`
	`end process;`

AHDL designs explicitly declare DFF registers rather than infer, so the software uses the user-declared name for the register.

For schematic designs using a **.bdf**, your design names all elements when you instantiate the elements in your design, so the software uses the name you defined for the register or DFF.

In the special case that a wire or signal (such as `my_dff_out` in the preceding examples) is also an output pin of your top-level design, the Quartus Prime software cannot use that name for the register (for example, cannot use `my_dff_out`) because the software requires that all logic and I/O cells have unique names. Here, Quartus Prime Integrated Synthesis appends `~reg0` to the register name.

Table 71. Verilog HDL Register Feeding Output Pin

For example, the Verilog HDL code example in this table generates a register called `q~reg0`.

HDL	Code

Verilog HDL	`module my_dff (input clk, input d, output q);`
	`always @ (posedge clk)`
	`q <= d;`
	`endmodule`

This situation occurs only for registers driving top-level pins. If a register drives a port of a lower level of the hierarchy, the software removes the port during hierarchy flattening and the register retains its original name, in this case, `q`.

3.8.3. Register Changes During Synthesis

On some occasions, you might not find registers that you expect to view in the synthesis netlist. Logic optimization might remove registers and synthesis optimizations might change the names of the registers. Common optimizations include inference of a state machine, counter, adder-subtractor, or shift register from registers and surrounding logic. Other common register changes occur when the software packs these registers into dedicated hardware on the FPGA, such as a DSP block or a RAM block.

The following factors can affect register names:

- Synthesis and Fitting Optimizations on page 185
- State Machines on page 186
- Inferred Adder-Subtractors, Shift Registers, Memory, and DSP Functions on page 187
- Packed Input and Output Registers of RAM and DSP Blocks on page 187

3.8.3.1. Synthesis and Fitting Optimizations

Logic optimization during synthesis might remove registers if you do not connect the registers to inputs or outputs in your design, or if you can simplify the logic due to constant signal values. Synthesis optimizations might change register names, such as when the software merges duplicate registers to reduce resource utilization.

NOT-gate push back optimizations can affect registers that use preset signals. This type of optimization can impact your timing assignments when the software uses registers as clock dividers. If this situation occurs in your design, change the clock settings to work on the new register name.

Synthesis netlist optimizations often change node names because the software can combine or duplicate registers to optimize your design.

The Quartus Prime Compilation Report provides a list of registers that synthesis optimizations remove, and a brief reason for the removal. To generate the Quartus Prime Compilation Report, follow these steps:

1. In the **Analysis & Synthesis** folder, open **Optimization Results.**
2. Open **Register Statistics,** and then click the **Registers Removed During Synthesis** report.
3. Click **Removed Registers Triggering Further Register Optimizations**.

The second report contains a list of registers that causes synthesis optimizations to remove other registers from your design. The report provides a brief reason for the removal, and a list of registers that synthesis optimizations remove due to the removal of the initial register.

Quartus Prime Integrated Synthesis creates synonyms for registers duplicated with the **Maximum Fan-Out** option (or `maxfan` attribute). Therefore, timing assignments applied to nodes that are duplicated with this option are applied to the new nodes as well.

The Quartus Prime Fitter can also change node names after synthesis (for example, when the Fitter uses register packing to pack a register into an I/O element, or when physical synthesis modifies logic). The Fitter creates synonyms for duplicated registers so timing analysis can use the existing node name when applying assignments.

You can instruct the Quartus Prime software to preserve certain nodes throughout compilation so you can use them for verification or making assignments.

3.8.3.2. State Machines

If your HDL code infers a state machine, the software maps the registers that represent the states into a new set of registers that implement the state machine. Most commonly, the software converts the state machine into a one-hot form in which one register represents each state. In this case, for Verilog HDL or VHDL designs, the registers take the name of the state register and the states.

For example, consider a Verilog HDL state machine in which the states are `parameter state0 = 1`, `state1 = 2`, `state2 = 3`, and in which the software declares the state machine register as `reg [1:0] my_fsm`. In this example, the three one-hot state registers are `my_fsm.state0`, `my_fsm.state1`, and `my_fsm.state2`.

An AHDL design explicitly specifies state machines with a state machine name. Your design names state machine registers with synthesized names based on the state machine name, but not the state names. For example, if a `my_fsm` state machine has four state bits, The software might synthesize these state bits with names such as `my_fsm~12`, `my_fsm~13`, `my_fsm~14`, and `my_fsm~15`.

3.8.3.3. Inferred Adder-Subtractors, Shift Registers, Memory, and DSP Functions

The Quartus Prime software infers IP cores from Verilog HDL and VHDL code for logic that forms adder-subtractors, shift registers, RAM, ROM, and arithmetic functions that are placed in DSP blocks.

Because adder-subtractors are part of an IP core instead of generic logic, the combinational logic exists in the design with different names. For shift registers, memory, and DSP functions, the software implements the registers and logic inside the dedicated RAM or DSP blocks in the device. Thus, the registers are not visible as separate LEs or ALMs.

3.8.3.4. Packed Input and Output Registers of RAM and DSP Blocks

The software packs registers into the input registers and output registers of RAM and DSP blocks, so that they are not visible as separate registers in LEs or ALMs.

3.8.4. Preserving Register Names

Altera recommends that you preserve certain register names for verification or debugging, or to ensure that you applied timing assignments correctly. Quartus Prime Integrated Synthesis preserves certain nodes automatically if the software uses the nodes in a timing constraint.

Related Information

- Preserve Registers on page 153
  Use the `preserve` attribute to instruct the Compiler not to minimize or remove a specified register during synthesis optimizations or register netlist optimizations
- Noprune Synthesis Attribute/Preserve Fan-out Free Register Node on page 155
  Use the `noprune` attribute to preserve a fan-out-free register through the entire compilation flow
- Disable Register Merging/Don't Merge Register on page 154
  Use the synthesis attribute `syn_dont_merge` to ensure that the Compiler does not merge registers with other registers

3.8.5. Node-Naming Conventions for Combinational Logic Cells

Whenever possible for Verilog HDL, VHDL, and AHDL code, the Quartus Prime software uses wire names that are the targets of assignments, but can change the node names due to synthesis optimizations.

For example, consider the Verilog HDL code in this example. Quartus Prime Integrated Synthesis uses the names `c`, `d`, `e`, and `f` for the generated combinational logic cells.

```
wire c;
reg d, e, f;
assign c = a | b;
always @ (a or b)
d = a & b;
always @ (a or b) begin : my_label
e = a ^ b;
end
always @ (a or b)
f = ~(a | b);
```

For schematic designs using a **.bdf**, your design names all elements when you instantiate the elements in your design and the software uses the name you defined when possible.

If logic cells are packed with registers in device architectures such as the Stratix and Cyclone device families, those names might not appear in the netlist after fitting. In other devices, such as newer families in the Stratix and Cyclone series device families, the register and combinational nodes are kept separate throughout the compilation, so these names are more often maintained through fitting.

When logic optimizations occur during synthesis, it is not always possible to retain the initial names as described. Sometimes, synthesized names are used, which are the wire names with a tilde (~) and a number appended. For example, if a complex expression is assigned to wire `w` and that expression generates several logic cells, those cells can have names such as `w`, `w~1`, and `w~2`. Sometimes the original wire name `w` is removed, and an arbitrary name such as `rtl~123` is created. Quartus Prime Integrated Synthesis attempts to retain user names whenever possible. Any node name ending with ~<*number*> is a name created during synthesis, which can change if the design is changed and re-synthesized. Knowing these naming conventions helps you understand your post-synthesis results, helping you to debug your design or create assignments.

During synthesis, the software maintains combinational clock logic by not changing nodes that might be clocks. The software also maintains or protects multiplexers in clock trees, so that the Timing Analyzer has information about which paths are unate, to allow complete and correct analysis of combinational clocks. Multiplexers often occur in clock trees when the software selects between different clocks. To help with the analysis of clock trees, the software ensures that each multiplexer encountered in a clock tree is broken into 2:1 multiplexers, and each of those 2:1 multiplexers is mapped into one lookup table (independent of the device family). This optimization might result in a slight increase in area, and for some designs a decrease in timing performance. To disable the option, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis) ➤ Clock MUX Protection**.

Related Information

Clock MUX Protection logic option
For more information about Clock MUX Protection logic option and a list of supported devices

3.8.6. Preserving Combinational Logic Names

You can preserve certain combinational logic node names for verification or debugging, or to ensure that timing assignments are applied correctly.

Use the `keep` attribute to keep a wire name or combinational node name through logic synthesis minimizations and netlist optimizations.

For any internal node in your design clock network, use `keep` to protect the name so that you can apply correct clock settings. Also, set the attribute for combinational logic involved in `cut` and `-through` assignments.

Note: Setting the `keep` attribute for combinational logic can increase the area utilization and increase the delay of the final mapped logic because the attribute requires the insertion of extra combinational logic. Use the attribute only when necessary.

Related Information

Keep Combinational Node/Implement as Output of Logic Cell on page 156

3.9. Scripting Support

You can run procedures and make settings 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 command at the command prompt shown in this example:

```
quartus_sh --qhelp
```

You can specify many of the options either on an instance, at the global level, or both.

To make a global assignment, use the Tcl command shown in this example:

```
set_global_assignment -name <QSF Variable Name> <Value>
```

To make an instance assignment, use the Tcl command shown in this example:

```
set_instance_assignment -name <QSF Variable Name> <Value>\ -to <Instance Name>
```

To set the **Synthesis Effort** option at the command line, use the `--effort` option with the `quartus_map` executable shown in this example:

```
quartus_map <Design name> --effort= "auto | fast"
```

Related Information

- Tcl Scripting
  For more information about Tcl scripting
- Quartus Prime Settings File Manual
  For more information about all settings and constraints in the Quartus Prime software
- Command-Line Scripting
  For more information about command-line scripting

3.9.1. Adding an HDL File to a Project and Setting the HDL Version

To add an HDL or schematic entry design file to your project, use the Tcl assignments shown in this example:

```
set_global_assignment –name VERILOG_FILE <file name>.<v|sv>
set_global_assignment –name SYSTEMVERILOG_FILE <file name>.sv
set_global_assignment –name VHDL_FILE <file name>.<vhd|vhdl>
set_global_assignment -name AHDL_FILE <file name>.tdf
set_global_assignment -name BDF_FILE <file name>.bdf
```

Note: You can use any file extension for design files, as long as you specify the correct language when adding the design file. For example, you can use **.h** for Verilog HDL header files.

To specify the Verilog HDL or VHDL version, use the option shown in this example, at the end of the VERILOG_FILE or VHDL_FILE command:

```
-    HDL_VERSION <language version>
```

The variable <language version> takes one of the following values:

- VERILOG_1995
- VERILOG_2001
- SYSTEMVERILOG_2005
- VHDL_1987
- VHDL_1993
- VHDL_2008

For example, to add a Verilog HDL file called **my_file.v** written in Verilog-1995, use the command shown in this example:

```
set_global_assignment –name VERILOG_FILE my_file.v –HDL_VERSION \ VERILOG_1995
```

In this example, the syn_encoding attribute associates a binary encoding with the states in the enumerated type count_state. In this example, the states are encoded with the following values: zero = "11", one = "01", two = "10", three = "00".

```
ARCHITECTURE rtl OF my_fsm IS
    TYPE count_state is (zero, one, two, three);
    ATTRIBUTE syn_encoding : STRING;
    ATTRIBUTE syn_encoding OF count_state : TYPE IS "11 01 10 00";
    SIGNAL present_state, next_state : count_state;
BEGIN
```

You can also use the syn_encoding attribute in Verilog HDL to direct the synthesis tool to use the encoding from your HDL code, instead of using the **State Machine Processing** option.

The syn_encoding value "user" instructs the Quartus Prime software to encode each state with its corresponding value from the Verilog HDL source code. By changing the values of your state constants, you can change the encoding of your state machine.

In Verilog-2001 and SystemVerilog Code: Specifying User-Encoded States with the syn_encoding Attribute on page 190, the states are encoded as follows:

```
init = "00"
last = "11"
next = "01"
later = "10"
```

Example 22. Verilog-2001 and SystemVerilog Code: Specifying User-Encoded States with the syn_encoding Attribute

```
(* syn_encoding = "user" *) reg [1:0] state;
parameter init = 0, last = 3, next = 1, later = 2;
always @ (state) begin
case (state)
init:
out = 2'b01;
next:
```

```
out = 2'b10;
later:
out = 2'b11;
last:
out = 2'b00;
endcase
end
```

Without the syn_encoding attribute, the Quartus Prime software encodes the state machine based on the current value of the **State Machine Processing** logic option.

If you also specify a safe state machine (as described in Safe State Machine on page 151), separate the encoding style value in the quotation marks from the safe value with a comma, as follows: "safe, one-hot" or "safe, gray".

Related Information

- Safe State Machine on page 151
- Manually Specifying State Assignments Using the syn_encoding Attribute on page 148

3.9.2. 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.

3.9.3. Creating Design Partitions for Incremental Compilation

To create a partition, use the command shown in this example:

```
set_instance_assignment -name PARTITION_HIERARCHY \
<file name> -to <destination> -section_id <partition name>
```

The <*file name*> variable is the name used for internally generated netlist files during incremental compilation. If you create the partition in the Quartus Prime software, netlist files are named automatically by the Quartus Prime software based on the instance name. If you use Tcl to create your partitions, you must assign a custom file name that is unique across all partitions. For the top-level partition, the specified file name is ignored, and you can use any dummy value. To ensure the names are safe and platform independent, file names should be unique, regardless of case. For example, if a partition uses the file name my_file, no other partition can use the file name MY_FILE. To make file naming simple, Altera recommends that you base each file name on the corresponding instance name for the partition.

The <*destination*> is the short hierarchy path of the entity. A short hierarchy path is the full hierarchy path without the top-level name, for example: "ram:ram_unit|altsyncram:altsyncram_component" (with quotation marks). For the top-level partition, you can use the pipe (|) symbol to represent the top-level entity.

The <*partition name*> is the partition name you designate, which should be unique and less than 1024 characters long. The name may only consist of alphanumeric characters, as well as pipe (|), colon (:), and underscore (_) characters. Altera recommends enclosing the name in double quotation marks (" ").

Related Information

Node-Naming Conventions in Quartus Prime Integrated Synthesis on page 184
For more information about hierarchical naming conventions

3.10. Document Revision History

Table 72. Document Revision History

Date	Version	Changes
2021.11.01	21.1	• Added "Support for VHDL 2008" and provide list of supported VHDL 2008 constructs. The previous removal of this topic was done in error.
2019.01.25	18.1.0	• Removed reference to Add Pass-Through Logic to Inferred RAMs GUI option. This option can only be set in the Intel Quartus Prime Settings File (.qsf).
2018.09.24	18.1.0	• Added *Factors Affecting Compilation Results* topic. • Removed references to VHDL-2008 synthesis support. This support was listed in error and VHDL-2008 is only supported in Quartus Prime Pro Edition
2016.05.03	16.0.0	Corrected description of Fitter Initial Placement Seed option.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
2015.05.04	15.0.0	• Removed support for early timing estimate feature. • Removed the note on the assignment of the RAM style attributes as it is no longer relevant.
2014.12.15	14.1.0	Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Optimization Settings to Compiler Settings.
2014.06.30	14.0.0	Template update.
November 2013	13.1.0	• Added a note regarding ROM inference using the `ram_init_file` in "RAM Initialization File—for Inferred Memory" on page 16–61.
May 2013	13.0.0	• Added "Verilog HDL Configuration" on page 16–6. • Added "RAM Style Attribute—For Shift Registers Inference" on page 16–57. • Added "Upgrade IP Components Dialog Box" on page 16–75.
June 2012	12.0.0	• Updated "Design Flow" on page 16–2.
November 2011	11.1.0	• Updated "Language Support" on page 16–5, "Incremental Compilation" on page 16–22, "Quartus Prime Synthesis Options" on page 16–24.
May 2011	11.0.0	• Updated "Specifying Pin Locations with chip_pin" on page 14–65, and "Shift Registers" on page 14–48. • Added a link to Quartus Prime Help in "SystemVerilog Support" on page 14–5. • Added Example 14–106 and Example 14–107 on page 14–67.
December 2010	10.1.0	• Updated "Verilog HDL Support" on page 13–4 to include Verilog-2001 support. • Updated "VHDL-2008 Support" on page 13–9 to include the condition operator (explicit and implicit) support. • Rewrote "Limiting Resource Usage in Partitions" on page 13–32. • Added "Creating LogicLock Regions" on page 13–32 and "Using Assignments to Limit the Number of RAM and DSP Blocks" on page 13–33.

continued...

Date	Version	Changes
		• Updated "Turning Off the Add Pass-Through Logic to Inferred RAMs no_rw_check Attribute" on page 13–55. • Updated "Auto Gated Clock Conversion" on page 13–28. • Added links to Quartus Prime Help.
July 2010	10.0.0	• Removed Referenced Documents section. • Added "Synthesis Seed" on page 9–36 section. • Updated the following sections: "SystemVerilog Support" on page 9–5 "VHDL-2008 Support" on page 9–10 "Using Parameters/Generics" on page 9–16 "Parallel Synthesis" on page 9–21 "Limiting Resource Usage in Partitions" on page 9–32 "Synthesis Effort" on page 9–35 "Synthesis Attributes" on page 9–25 "Synthesis Directives" on page 9–27 "Auto Gated Clock Conversion" on page 9–29 "State Machine Processing" on page 9–36 "Multiply-Accumulators and Multiply-Adders" on page 9–50 "Resource Aware RAM, ROM, and Shift-Register Inference" on page 9–52 "RAM Style and ROM Style—for Inferred Memory" on page 9–53 "Turning Off the Add Pass-Through Logic to Inferred RAMs no_rw_check Attribute" on page 9–55 "Using altera_attribute to Set Quartus Prime Logic Options" on page 9–68 "Adding an HDL File to a Project and Setting the HDL Version" on page 9–83 "Creating Design Partitions for Incremental Compilation" on page 9–85 "Inferring Multiplier, DSP, and Memory Functions from HDL Code" on page 9–50 • Updated Table 9–9 on page 9–86.
December 2009	9.1.1	• Added information clarifying inheritance of Synthesis settings by lower-level entities, including Altera and third-party IP • Updated "Keep Combinational Node/Implement as Output of Logic Cell" on page 9–46
November 2009	9.1.0	• Updated the following sections: "Initial Constructs and Memory System Tasks" on page 9–7 "VHDL Support" on page 9–9 "Parallel Synthesis" on page 9–21 "Synthesis Directives" on page 9–27 "Timing-Driven Synthesis" on page 9–31 "Safe State Machines" on page 9–40 "RAM Style and ROM Style—for Inferred Memory" on page 9–53 "Translate Off and On / Synthesis Off and On" on page 9–62 "Read Comments as HDL" on page 9–63 "Adding an HDL File to a Project and Setting the HDL Version" on page 9–81 • Removed "Remove Redundant Logic Cells" section • Added "Resource Aware RAM, ROM, and Shift-Register Inference" section • Updated Table 9–9 on page 9–83
March 2009	9.0.0	• Updated Table 9–9. • Updated the following sections: "Partitions for Preserving Hierarchical Boundaries" on page 9–20 "Analysis & Synthesis Settings Page of the Settings Dialog Box" on page 9–24 "Timing-Driven Synthesis" on page 9–30 "Turning Off Add Pass-Through Logic to Inferred RAMs/ no_rw_check Attribute Setting" on page 9–54 • Added "Parallel Synthesis" on page 9–21 • Chapter 9 was previously Chapter 8 in software version 8.1

Related Information

Documentation Archive

For previous versions of the *Quartus Prime Handbook*, search the documentation archives.

4. Reducing Compilation Time

You can employ various techniques to reduce the time required for synthesis and fitting in the Quartus Prime Compiler.

4.1. Strategies to Reduce the Overall Compilation Time

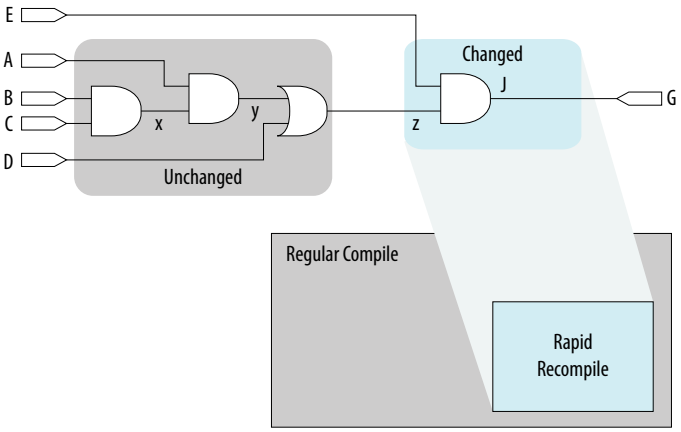
You can use the following strategies to reduce the overall time required to compile your design:

- Enabling Multi-Processor Compilation on page 196
- Using Incremental Compilation on page 200
- Using Block-Based Compilation on page 201
- Incremental compilation reduces compilation time by only recompiling design partitions that have not met design requirements.

4.1.1. Running Rapid Recompile

During Rapid Recompile the Compiler reuses previous synthesis and fitting results whenever possible, and does not reprocess unchanged design blocks. Use Rapid Recompile to reduce timing variations and the total recompilation time after making small design changes.

Figure 36. Rapid Recompile



© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

To run Rapid Recompile, follow these steps:

1. Prior to initial compilation, click **Assignments ➤ Settings ➤ Compiler Settings** and turn on **Enable Intermediate Fitter Snapshots**. This option must be enabled to subsequently use the Rapid Recompile feature.
2. To start Rapid Recompile following an initial compilation (or after running the Route stage of the Fitter), click **Processing ➤ Start ➤ Start Rapid Recompile**. Rapid Recompile implements the following types of design changes without full recompilation:
   - Changes to nodes tapped by the Signal Tap Logic Analyzer
   - Changes to combinational logic functions
   - Changes to state machine logic (for example, new states, state transition changes)
   - Changes to signal or bus latency or addition of pipeline registers
   - Changes to coefficients of an adder or multiplier
   - Changes register packing behavior of DSP, RAM, or I/O
   - Removal of unnecessary logic
   - Changes to synthesis directives

   The Incremental Compilation Preservation Summary report provides details about placement and routing implementation.
3. Click the Rapid Recompile Preservation Summary report to view detailed information about the percentage of preserved compilation results.

Figure 37. Rapid Recompile Preservation Summary

Rapid Recompile Preservation Summary

	Type	Achieved
1	Placement (by node)	33.25 % (2160 / 6497)
2	Routing (by connection)	49.93 % (14165 / 28372)

4.1.2. 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.

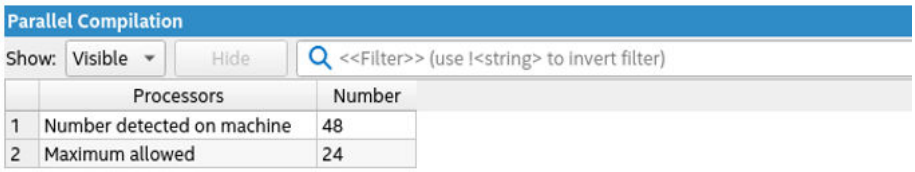
The use of multiple processors does not affect the quality of the fit. The fit is the same and deterministic for a given Fitter seed and given **Maximum processors allowed** setting on a specific design. This remains true regardless of the target system and the number of available processors. Different **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 38. 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
```

The actual reduction in compilation time when using incremental compilation partitions depends on your design and on the specific compilation settings. For example, compilations with multi-corner optimization enabled benefit more from using multiple processors than compilations without multi-corner optimization. The Fitter (`quartus_fit`) and the Quartus Prime Timing Analyzer (`quartus_sta`) stages in the compilation can, in certain cases, benefit from the use of multiple processors. The Flow Elapsed Time report shows the average number of processors for these stages. The Parallel Compilation report shows a more detailed breakdown of processor usage. This report displays only if you enable parallel compilation.

For designs with partitions, once you partition your design and enable partial compilation, the Quartus Prime software can use different processors to compile those partitions simultaneously during Analysis & Synthesis. This can cause higher peak memory usage during Analysis & Synthesis.

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:

4.1.2.1. 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.

4.1.2.2. 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 39. 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.

4.1.2.3. 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.

4.1.3. Using Incremental Compilation

The incremental compilation feature can accelerate design iteration time by up to 70% for small design changes, and helps you reach design timing closure more efficiently.

You can speed up design iterations by recompiling only a particular design partition and merging results with previous compilation results from other partitions. You can also use physical synthesis optimization techniques for specific design partitions while leaving other parts of your design untouched to preserve performance.

If you are using a third-party synthesis tool, you can create separate atom netlist files for the parts of your design that you already have synthesized and optimized so that you update only the parts of your design that change.

In the standard incremental compilation design flow, you can divide the top-level design into partitions, which the software can compile and optimize in the top-level Quartus Prime project. You can preserve fitting results and performance for completed partitions while other parts of your design are changing. Incremental compilation reduces the compilation time for each design iteration because the software does not recompile the unchanged partitions in your design.

The incremental compilation feature facilitates team-based design flows by enabling designers to create and optimize design blocks independently, when necessary, and supports third-party IP integration.

4.1.4. Using Block-Based Compilation

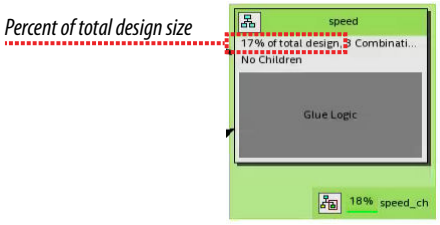
During the design process, when making minor modifications to a design, recompiling the entire design can result in longer compilation times than anticipated. This is because every time you recompile a design following a change, the compiler may apply global optimizations to enhance resource utilization 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.

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 40. 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.

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.

4.2. 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.

4.2.1. 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.

4.2.2. 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.

4.3. 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 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.

You can reduce the placement time by changing the settings for the placement algorithm, or by using incremental compilation to preserve the placement for the unchanged parts of your design.

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.

4.3.1. Fitter Effort Setting

For designs with very tight timing requirements, both **Auto Fit** and **Standard Fit** use the maximum effort during optimization.Intel recommends using **Auto Fit** for reducing compilation time.

The highest Fitter effort setting, **Standard Fit**, requires the most runtime, but does not always yield a better result than using the default **Auto Fit**. If you are certain that your design has only easy-to-meet timing constraints, you can select **Fast Fit** for an even greater runtime savings.

4.3.2. 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.

4.3.3. Physical Synthesis Effort Settings

Physical synthesis options enable you to optimize the post-synthesis netlist and improve timing performance. These options, which affect placement, can significantly increase compilation time.

If your design meets your performance requirements without physical synthesis options, turn them off to reduce compilation time. For example, if some or all the physical synthesis algorithm information messages display an improvement of 0 ps, turning off physical synthesis can reduce compilation time.

You also can use the **Physical synthesis effort** setting on the **Advanced Fitter Settings** dialog box to reduce the amount of extra compilation time used by these optimizations.

The **Fast** setting directs the Quartus Prime software to use a lower level of physical synthesis optimization. Compared to the **Normal** physical synthesis effort level, using the **Fast** setting can cause a smaller increase in compilation time. However, the lower level of optimization can result in a smaller increase in design performance.

4.3.4. Preserving Placement with Incremental Compilation

Preserving information about previous placements can make future placements faster. The incremental compilation feature provides an easy-to-use method for preserving placement results.

4.4. 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**.
- Use incremental compilation to preserve routing information for parts of your design.

4.4.1. 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.

Related Information

Using Incremental Compilation on page 200

4.4.1.1. 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.

4.4.1.2. 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.

4.4.1.3. Preserving Routing with Incremental Compilation

Preserving the previous routing results for part of your design can reduce future routing time. Incremental compilation provides an easy-to-use methodology that preserves placement and routing results.

4.5. 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.

4.6. 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.

4.7. Reducing Compilation Time Revision History

Date	Version	Changes
2016.05.02	16.0.0	• Corrected typo in Using Parallel Compilation with Multiple Processors. • Stated limitations 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.

continued...

Date	Version	Changes
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.

A. Quartus Prime Standard Edition User Guides

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

Related Information

- Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Quartus Prime Standard Edition software, including managing Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Quartus Prime Standard 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, and optimization of device resource usage.
- Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Quartus Prime Standard 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 Standard 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.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Quartus Prime Standard 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.
- Quartus Prime Standard 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.
- Quartus Prime Standard Edition User Guide: Debug Tools
  Describes a portfolio of Quartus Prime Standard 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.
- Quartus Prime Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Quartus Prime Standard 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.
- Quartus Prime Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Quartus Prime Standard 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 Standard Edition User Guide

Design Optimization

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Design Optimization Overview 6
1.1. Device Considerations 6
1.1.1. Device Migration Considerations 6
1.2. Required Settings for Initial Compilation 7
1.2.1. Guidelines for I/O Assignments 7
1.2.2. Guidelines for Time Constraints 7
1.2.3. Partitions and Floorplan Assignments for Incremental Compilation 8
1.3. Trade-Offs and Limitations 8
1.3.1. Preserving Results and Enabling Teamwork 9
1.3.2. Reducing Area 9
1.3.3. Reducing Critical Path Delay 9
1.3.4. Reducing Power Consumption 10
1.3.5. Reducing Runtime 10
1.4. Intel Quartus Prime Software Tools for Design Optimization 11
1.4.1. Design Visualization Tools 11
1.4.2. Advisors 11
1.4.3. Design Exploration 12
1.5. Design Space Explorer II 12
1.5.1. How DSE II Works 13
1.5.2. Performing a Design Exploration with the DSE II Utility 14
1.6. Design Optimization Overview Revision History 15

2. Optimizing the Design Netlist 16
2.1. When to Use the Netlist Viewers: Analyzing Design Problems 16
2.2. Intel Quartus Prime Design Flow with the Netlist Viewers 17
2.3. RTL Viewer Overview 18
2.3.1. Maximizing Readability in RTL Viewer 19
2.3.2. Running the RTL Viewer 20
2.4. State Machine Viewer Overview 20
2.5. Technology Map Viewer Overview 20
2.6. Netlist Viewer User Interface 21
2.6.1. Netlist Navigator Pane 24
2.6.2. Properties Pane 24
2.6.3. Netlist Viewers Find Pane 26
2.7. Schematic View 26
2.7.1. Display Schematics in Multiple Tabbed View 26
2.7.2. Schematic Symbols 27
2.7.3. Select Items in the Schematic View 31
2.7.4. Shortcut Menu Commands in the Schematic View 32
2.7.5. Filtering in the Schematic View 32
2.7.6. View Contents of Nodes in the Schematic View 33
2.7.7. Moving Nodes in the Schematic View 34
2.7.8. View LUT Representations in the Technology Map Viewer 35
2.7.9. Zoom Controls 35
2.7.10. Navigating with the Bird's Eye View 36
2.7.11. Partition the Schematic into Pages 36
2.7.12. Follow Nets Across Schematic Pages 37

2.8. State Machine Viewer 37
2.8.1. State Diagram View 38
2.8.2. State Transition Table 38
2.8.3. State Encoding Table 38
2.8.4. Switch Between State Machines 39
2.9. Cross-Probing to a Source Design File and Other Intel Quartus Prime Windows 39
2.10. Cross-Probing to the Netlist Viewers from Other Intel Quartus Prime Windows 40
2.11. Viewing a Timing Path 40
2.12. Optimizing the Design Netlist Revision History 41

3. Timing Closure and Optimization 43
3.1. Optimize Multi Corner Timing 43
3.2. Critical Paths 43
3.2.1. Viewing Critical Paths 44
3.3. Design Evaluation for Timing Closure 44
3.3.1. Review Compilation Results 44
3.3.2. Review Details of Timing Paths 55
3.3.3. Adjusting and Recompiling 58
3.4. Design Analysis 59
3.4.1. Ignored Timing Constraints 59
3.4.2. I/O Timing 59
3.4.3. Register-to-Register Timing Analysis 60
3.5. Timing Optimization 64
3.5.1. Displaying Timing Closure Recommendations for Failing Paths 64
3.5.2. Timing Optimization Advisor 65
3.5.3. Optional Fitter Settings 66
3.5.4. I/O Timing Optimization Techniques 68
3.5.5. Register-to-Register Timing Optimization Techniques 72
3.5.6. Logic Lock (Standard) Assignments 78
3.5.7. Location Assignments 80
3.5.8. Metastability Analysis and Optimization Techniques 80
3.6. Periphery to Core Register Placement and Routing Optimization 80
3.6.1. Setting Periphery to Core Optimizations in the Advanced Fitter Setting Dialog Box 81
3.6.2. Setting Periphery to Core Optimizations in the Assignment Editor 82
3.6.3. Viewing Periphery to Core Optimizations in the Fitter Report 82
3.7. Scripting Support 83
3.7.1. Initial Compilation Settings 84
3.7.2. I/O Timing Optimization Techniques 84
3.7.3. Register-to-Register Timing Optimization Techniques 85
3.8. Timing Closure and Optimization Revision History 86

4. Area Optimization 89
4.1. Resource Utilization Information 89
4.1.1. Flow Summary Report 89
4.1.2. Fitter Reports 89
4.1.3. Analysis and Synthesis Reports 90
4.1.4. Compilation Messages 90
4.2. Optimizing Resource Utilization 90
4.2.1. Using the Resource Optimization Advisor 91
4.2.2. Resource Utilization Issues Overview 91
4.2.3. I/O Pin Utilization or Placement 91

4.2.4. Logic Utilization or Placement..... 92
4.2.5. Routing..... 96
4.3. Scripting Support.....98
4.3.1. Initial Compilation Settings..... 99
4.3.2. Resource Utilization Optimization Techniques..... 100
4.4. Area Optimization Revision History..... 101

5. Analyzing and Optimizing the Design Floorplan..... 102
5.1. Design Floorplan Analysis in the Chip Planner.....102
5.1.1. Starting the Chip Planner..... 103
5.1.2. Chip Planner GUI Components..... 103
5.1.3. Viewing Architecture-Specific Design Information..... 105
5.1.4. Viewing Available Clock Networks in the Device.....105
5.1.5. Viewing Routing Congestion.....106
5.1.6. Viewing I/O Banks.....107
5.1.7. Viewing High-Speed Serial Interfaces (HSSI)..... 107
5.1.8. Viewing the Source and Destination of Placed Nodes.....108
5.1.9. Viewing Fan-In and Fan-Out Connections of Placed Resources..... 109
5.1.10. Generating Immediate Fan-In and Fan-Out Connections.....110
5.1.11. Exploring Paths in the Chip Planner.....110
5.1.12. Viewing Assignments in the Chip Planner.....112
5.1.13. Viewing High-Speed and Low-Power Tiles in the Chip Planner..... 113
5.1.14. Viewing Design Partition Placement.....114
5.2. Logic Lock (Standard) Regions.....114
5.2.1. Attributes of a Logic Lock (Standard) Region.....115
5.2.2. Creating Logic Lock (Standard) Regions..... 115
5.2.3. Customizing the Shape of Logic Lock Regions.....118
5.2.4. Placing Logic Lock (Standard) Regions..... 119
5.2.5. Placing Device Resources into Logic Lock (Standard) Regions..... 120
5.2.6. Hierarchical (Parent and Child) Logic Lock (Standard) Regions.....123
5.2.7. Additional Intel Quartus Prime Logic Lock (Standard) Design Features.....124
5.2.8. Logic Lock (Standard) Regions Window..... 124
5.3. Using Logic Lock (Standard) Regions in the Chip Planner..... 125
5.3.1. Viewing Connections Between Logic Lock (Standard) Regions in the Chip Planner.....125
5.3.2. Using Logic Lock (Standard) Regions with the Design Partition Planner..... 126
5.4. Scripting Support.....126
5.4.1. Initializing and Uninitializing a Logic Lock (Standard) Region.....126
5.4.2. Creating or Modifying Logic Lock (Standard) Regions..... 126
5.4.3. Obtaining Logic Lock (Standard) Region Properties..... 127
5.4.4. Assigning Logic Lock (Standard) Region Content.....127
5.4.5. Save a Node-Level Netlist for the Entire Design into a Persistent Source File.. 127
5.4.6. Setting Logic Lock (Standard) Assignment Priority..... 128
5.4.7. Assigning Virtual Pins with a Tcl command..... 128
5.5. Analyzing and Optimizing the Design Floorplan Revision History..... 128

6. Netlist Optimizations and Physical Synthesis..... 131
6.1. Physical Synthesis Optimizations.....131
6.1.1. Enabling Physical Synthesis Optimization.....132
6.1.2. Physical Synthesis Options..... 132
6.1.3. Perform Register Retiming for Performance.....133
6.1.4. Preventing Register Movement During Retiming.....134

6.2. Applying Netlist Optimizations....135
6.2.1. WYSIWYG Primitive Resynthesis....136
6.2.2. Saving a Node-Level Netlist....137
6.3. Viewing Synthesis and Netlist Optimization Reports....138
6.4. Scripting Support....138
6.4.1. Synthesis Netlist Optimizations....139
6.4.2. Physical Synthesis Optimizations....139
6.4.3. Back-Annotating Assignments....140
6.5. Netlist Optimizations and Physical Synthesis Revision History....140

7. Engineering Change Orders with the Chip Planner....142
7.1. Engineering Change Orders....142
7.1.1. Performance Preservation....143
7.1.2. Compilation Time....143
7.1.3. Verification....143
7.1.4. Change Modification Record....144
7.2. ECO Design Flow....144
7.3. The Chip Planner Overview....146
7.3.1. Opening the Chip Planner....146
7.3.2. The Chip Planner Tasks and Layers....147
7.4. Performing ECOs with the Chip Planner (Floorplan View)....147
7.4.1. Creating, Deleting, and Moving Atoms....147
7.4.2. Check and Save Netlist Changes....147
7.5. Performing ECOs in the Resource Property Editor....147
7.5.1. Logic Elements....148
7.5.2. Adaptive Logic Modules....150
7.5.3. FPGA I/O Elements....151
7.5.4. FPGA RAM Blocks....155
7.5.5. FPGA DSP Blocks....156
7.6. Change Manager....157
7.6.1. Complex Changes in the Change Manager....158
7.6.2. Managing Signal Probe Signals....158
7.6.3. Exporting Changes....158
7.7. Scripting Support....158
7.8. Common ECO Applications....158
7.8.1. Adjust the Drive Strength of an I/O with the Chip Planner....159
7.8.2. Modify the PLL Properties With the Chip Planner....160
7.8.3. PLL Properties....161
7.8.4. Modify the Connectivity between Resource Atoms....163
7.9. Post ECO Steps....164
7.10. Engineering Change Orders with the Chip Planner Revision History....164

A. Intel Quartus Prime Standard Edition User Guides....166

1. Design Optimization Overview

In a typical design flow, the early stages of development concentrate on meeting timing, area and power goals. Once the design meets those goals, the efforts focus on improving performance. This chapter introduces techniques and tools in the Intel® Quartus® Prime software that you can use to achieve the highest design performance.

Optimization of a FPGA design requires a multi-dimensional approach that meets the design goals while reducing area, critical path delay, power consumption, and runtime. The Intel Quartus Prime software includes advisors to address each of these issues. By implementing the advisor's suggestions, you can reduce the time spent on design iterations.

Related Information

Intel Quartus Prime Design Software - Support Center

1.1. Device Considerations

All Intel FPGAs have a unique timing model that contains delay information for all physical elements in the device, such as combinational adaptive logic modules, memory blocks, interconnects, and registers. The delays encompass all valid combinations of operating conditions for the target FPGA. Additionally, the device size and package determine pin-out and the resource availability.

Related Information

Guaranteeing Silicon Performance with FPGA Timing Models
Intel FPGA White Paper (PDF)

1.1.1. Device Migration Considerations

If you anticipate a change to the target device later in the design cycle, plan for the migration from the beginning of cycle. This strategy helps to minimize changes to the design at a later stage.

When choosing a design's target device in the Intel Quartus Prime software, you can see a list of compatible devices by clicking the **Migration Devices** button in the **Device** dialog box.

Related Information

Migration Devices Dialog Box

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

1.2. Required Settings for Initial Compilation

Compilation results can vary significantly depending on the assignments and settings that you choose. In the Intel Quartus Prime software, the default values for settings and options provide the best trade-off between compilation time, resource utilization, and timing performance. Before compiling a design in the Intel Quartus Prime software, consider the following guidelines.

1.2.1. Guidelines for I/O Assignments

In a FPGA design, I/O standards and drive strengths affect I/O timing.

- When specifying I/O assignments, make sure that the Intel Quartus Prime software is using 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

I/O Planning Overview

1.2.2. Guidelines for Time Constraints

For best results, use real time requirements. Applying more demanding timing requirements than the design needs can cause the Compiler to trade off by increasing resource usage, power utilization, or compilation time.

Comprehensive timing requirement settings achieve the best results for the following reasons:

- Correct timing assignments enable the software to work hardest to optimize the performance of the timing-critical parts of the design and make trade-offs for performance. This optimization can also save area or power utilization in non-critical parts of the design.
- If enabled, the Intel Quartus Prime software performs physical synthesis optimizations based on timing requirements.
- Depending on the **Fitter Effort** setting, the Fitter can reduce runtime if the design meets the timing requirements.

The Intel Quartus Prime Timing Analyzer determines if the design implementation meets the timing requirement. The Compilation Report shows whether the design meets the timing requirements, while the timing analysis reporting commands provide detailed information about the timing paths.

Related Information

- Timing Closure and Optimization on page 43
- Using the Intel Quartus Prime Timing Analyzer
- Intel Quartus Prime Timing Analyzer Cookbook
- Advanced Settings (Fitter)

1.2.3. Partitions and Floorplan Assignments for Incremental Compilation

The Intel Quartus Prime incremental compilation feature enables hierarchical and team-based design flows in which you can compile parts of your design while other parts of your design remain unchanged. You can also Import parts of your design from separate Intel Quartus Prime projects.

Using incremental compilation for your design with good design partitioning methodology helps to achieve timing closure. Creating design partitions on some of the major blocks in your design and assigning them to Logic Lock (Standard)™ regions, reduces Fitter time and improves the quality and repeatability of the results. Logic Lock (Standard) regions are flexible, reusable floorplan location constraints that help you place logic on the target device. When you assign entity instances or nodes to a Logic Lock (Standard) region, you direct the Fitter to place those entity instances or nodes inside the region during fitting.

Using incremental compilation helps you achieve timing closure block by block and preserve the timing performance between iterations, which aid in achieving timing closure for the entire design. Incremental compilation may also help reduce compilation times.

Note: If you plan to use incremental compilation, you must create a floorplan for your design. If you are not using incremental compilation, creating a floorplan is optional.

Related Information

- Reducing Compilation Time
- Best Practices for Incremental Compilation Partitions and Floorplan Assignments

1.3. Trade-Offs and Limitations

Many optimization goals can conflict with one another, so you might need to resolve conflicting goals.

Table 1. Examples of Trade offs in Design Optimization

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, constrains that are too severe limit design feasibility as far as 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. Preserving Results and Enabling Teamwork

For some Intel Quartus Prime Fitter algorithms, small changes to the design can have a large impact on the final result. For example, a critical path delay can change by 10% or more because of seemingly insignificant changes. If you are close to meeting your timing objectives, you can use the Fitter algorithm to your advantage by changing the fitter seed, which changes the pseudo-random result of the Fitter.

Conversely, if you cannot meet timing on a portion of your design, you can partition that portion and prevent it from recompiling if an unrelated part of the design is changed. This feature, known as incremental compilation, can reduce the Fitter runtimes by up to 70% if the design is partitioned, such that only small portions require recompilation at any one time.

When you use incremental compilation, you can apply design optimization options to individual design partitions and preserve performance in other partitions by leaving them untouched. Many optimization techniques often result in longer compilation times, but by applying them only on specific partitions, you can reduce this impact and complete iterations more quickly.

In addition, by physically floorplanning your partitions with Logic Lock (Standard) regions, you can enable team-based flows and allow multiple people to work on different portions of the design.

Related Information

Intel Quartus Prime Incremental Compilation for Hierarchical and Team-Based Designs

1.3.2. Reducing Area

By default, the Intel 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. 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.

Related Information

- Area Optimization on page 89
- Netlist Optimizations and Physical Synthesis on page 131

1.3.3. Reducing Critical Path Delay

To meet complex timing requirements involving multiple clocks, routing resources, and area constraints, the Intel Quartus Prime software offers a close interaction between synthesis, floorplan editing, place-and-route, and timing analysis processes.

By default, the Intel Quartus Prime Fitter works to meet the timing requirements, and stops when the requirements are met. Therefore, realistic constraints are crucial for timing closure.

Under-constrained designs can lead to sub-optimal results. For over-constrained designs, the Fitter might over-optimize non-critical paths at the expense of true critical paths. In addition, area and compilation time may also increase.

For designs with high resource usage, the Intel Quartus Prime Fitter might have trouble finding a legal placement. In such circumstances, the Fitter automatically modifies settings to try to trade off performance for area.

The Intel Quartus Prime Fitter offers advanced options that can help improve the design performance when you properly set constraints. Use the Timing Optimization Advisor to determine which options are best suited for the design.

If you use incremental compilation, you can help resolve inter-partition timing requirements by locking down results, one partition at a time, or by guiding the placement of the partitions with Logic Lock (Standard) regions. You might improve the timing on such paths by placing the partitions optimally to reduce the length of critical paths. Once the inter-partition timing requirements are met, use incremental compilation to preserve the results and work on partitions that have not met timing requirements.

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 Intel 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. 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

Critical Paths on page 43

1.3.4. Reducing Power Consumption

The Intel 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.

Related Information

Power Optimization

1.3.5. Reducing Runtime

Many Fitter settings influence compilation time. Most of the default settings in the Intel Quartus Prime software are set for reduced compilation time. You can modify these settings based on your project requirements.

The Intel Quartus Prime software supports parallel compilation in computers with multiple processors. This can reduce compilation times by up to 15%.

You can also reduce compilation time with your iterations by using incremental compilation. Use incremental compilation when you want to change parts of your design, while keeping most of the remaining logic unchanged.

Related Information

Reducing Compilation Time

1.4. Intel Quartus Prime Software Tools for Design Optimization

The Intel Quartus Prime software offers tools that you can use to optimize a design.

1.4.1. Design Visualization Tools

The Intel Quartus Prime software provides tools that display graphical representations of a design.

Table 2. Visualization Tools

Tool	Description
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.
Design Partition Planner	Displays designs at partition and entity levels, and can display connectivity between entities
Design Partition Planner and Chip Planner (With incremental compilation)	Allow you to partition and layout the design at a higher level.
Chip Planner	Allow you to make floorplan assignments, implement engineering change orders (ECOs), perform power analysis, and visualize critical paths and routing congestion.

Related Information

- Design Floorplan Analysis in the Chip Planner on page 102
- Optimizing the Design Netlist on page 16
- RTL Viewer Overview on page 18

1.4.2. Advisors

The Intel Quartus Prime software includes several advisors to help you optimize your design and reduce compilation time.

The advisors provide recommendations based on the project settings and design constraints. Those recommendations can help you to fit the project, meet timing or power requirements, or improve the design performance.

The advisors organize the recommendations from general to specific. Where applicable, the categories are divided into of stages presented by complexity.

The advisors are:

- Resource Optimization Advisor
- Timing Optimization Advisor
- Power Optimization Advisor
- Compilation Time Advisor
- Pin Optimization Advisor
- Incremental Compilation Advisor

Related Information

- Compilation Time Advisor
- Advisors in the Intel Quartus Prime Software
- Timing Optimization Advisor on page 65
- Power Optimization Advisor

1.4.3. Design Exploration

The Design Space Explorer II tool (DSE II) provides an easy and efficient way for you to run experiments on your design settings. You can run a single compilation locally on your PC or remotely using compute farm resources.

Related Information

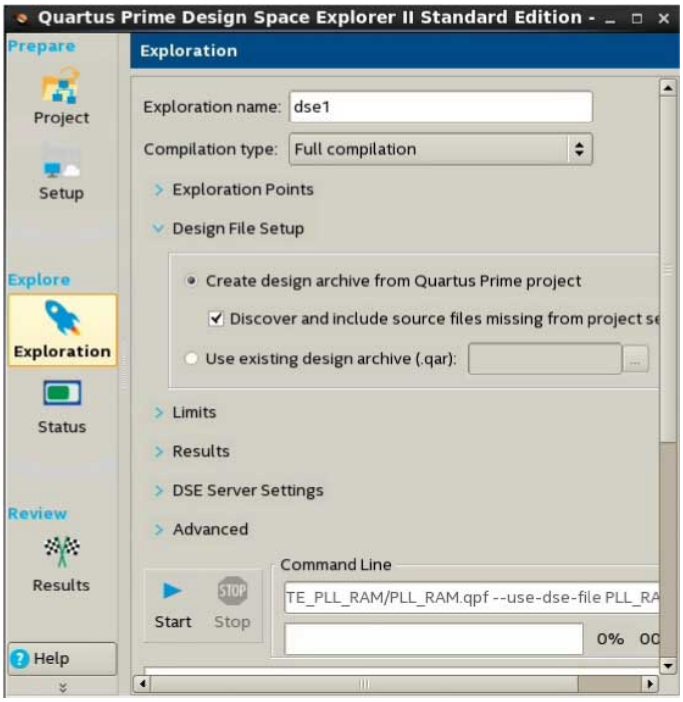
Design Space Explorer II on page 12

1.5. 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 settings for the design. You can take advantage of the DSE II parallelization abilities to compile on multiple computers.

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 1. Design Space Explorer II User Interface



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.

Related Information

- Design Space Explorer II Tool
- Using Design Space Explorer
  21 Minute Online Course

1.5.1. How DSE II Works

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.

Related Information

Design Space Explorer II for Power-Driven Optimization

1.5.1.1. Use of 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.

Related Information

- Setup Page (Design Space Explorer II)
- Status Page (Design Space Explorer II)

1.5.1.2. 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)

1.5.1.3. Result Management

DSE II compares the compilation results to determine the best Intel 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 Intel 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 Intel Quartus Prime Archive Files (`.qar`).

Related Information

Report Page (Design Space Explorer II)

1.5.2. Performing a Design Exploration with the DSE II Utility

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 Intel 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 Intel Quartus Prime software and launch DSE II, a dialog box appears asking if you want to close the Intel 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**.

Related Information

- Design Space Explorer II Tool
- Using Design Space Explorer
  21 Minute Online Course

1.6. Design Optimization Overview Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2018.05.07	18.0.0	• General topic reorganization.
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. • Updated DSE II content.
June 2014	14.0.0	Updated format.
November 2013	13.1.0	Minor changes for HardCopy.
May 2013	13.0.0	Added the information about initial compilation requirements. This section was moved from the Area Optimization chapter of the Intel Quartus Prime Handbook. Minor updates to delineate division of Timing and Area optimization chapters.
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 link
July 2010	10.0.0	Initial release. Chapter based on topics and text in Section III of volume 2.

Related Information

Documentation Archive
For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

2. Optimizing the Design Netlist

This chapter describes how you can use the Intel Quartus Prime Netlist Viewers to analyze and debug your designs.

As FPGA designs grow in size and complexity, the ability to analyze, debug, optimize, and constrain your design is critical. With today's advanced designs, several design engineers are involved in coding and synthesizing different design blocks, making it difficult to analyze and debug the design. The Intel Quartus Prime RTL Viewer, State Machine Viewer, and Technology Map Viewer provide powerful ways to view your initial and fully mapped synthesis results during the debugging, optimization, and constraint entry processes.

Related Information

- Intel Quartus Prime Design Flow with the Netlist Viewers on page 17
- State Machine Viewer Overview on page 20
- RTL Viewer Overview on page 18
- Technology Map Viewer Overview on page 20
- Filtering in the Schematic View on page 32
- Viewing a Timing Path on page 40

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, State Machine 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 and State Machine 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. You can also view state machine transitions and transition equations with the State Machine Viewer. 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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.

Related Information

- Intel Quartus Prime Design Flow with the Netlist Viewers on page 17
- State Machine Viewer Overview on page 20
- RTL Viewer Overview on page 18
- Technology Map Viewer Overview on page 20

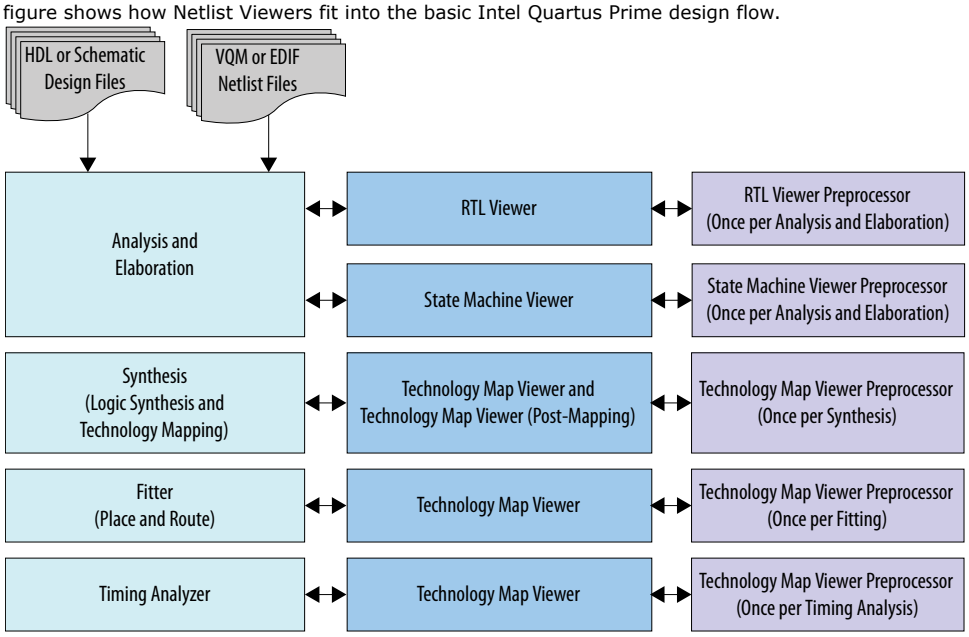
2.2. Intel Quartus Prime Design Flow with the Netlist Viewers

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 2. Intel Quartus Prime Design Flow Including the RTL Viewer and Technology Map Viewer

This figure shows how Netlist Viewers fit into the basic Intel Quartus Prime design flow.



Before the Netlist Viewer can run the preprocessor stage, you must compile your design:

- To open the RTL Viewer or State Machine 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 Intel 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 Intel Quartus Prime integrated synthesis results or third-party netlist files in the Intel Quartus Prime software.

You can view results after Analysis and Elaboration for designs that use any supported Intel 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 Intel 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 Intel Quartus Prime integrated synthesis, this view shows how the Intel 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 a Intel 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 Intel Quartus Prime compilation flow.

To open the RTL Viewer, click **Tools ➤ Netlist Viewers ➤ RTL Viewer**.

Related Information

Netlist Viewer User Interface on page 21

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.
- Converts state machine logic into a state diagram, state transition table, and state encoding table, which appear in the State Machine Viewer.

Related Information

State Machine Viewer Overview on page 20

2.3.2. Running the RTL Viewer

To run the RTL Viewer for an Intel 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 Intel Quartus Prime compilation flow.
2. Open the RTL Viewer by clicking **Tools ➤ Netlist Viewers ➤ RTL Viewer**.

2.4. State Machine Viewer Overview

The State Machine Viewer presents a high-level view of finite state machines in your design. The State Machine Viewer provides a graphical representation of the states and their related transitions, as well as a state transition table that displays the condition equation for each of the state transitions, and encoding information for each state.

To run the State Machine Viewer, on the Tools menu, point to **Netlist Viewers** and click **State Machine Viewer**. To open the State Machine Viewer for a particular state machine, double-click the state machine instance in the RTL Viewer.

2.5. Technology Map Viewer Overview

The Intel 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 Intel 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 Intel Quartus Prime technology-mapped results after synthesis, fitting, or timing analysis. To run the Technology Map Viewer for a Intel 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 40
- View Contents of Nodes in the Schematic View on page 33
- Netlist Viewer User Interface on page 21

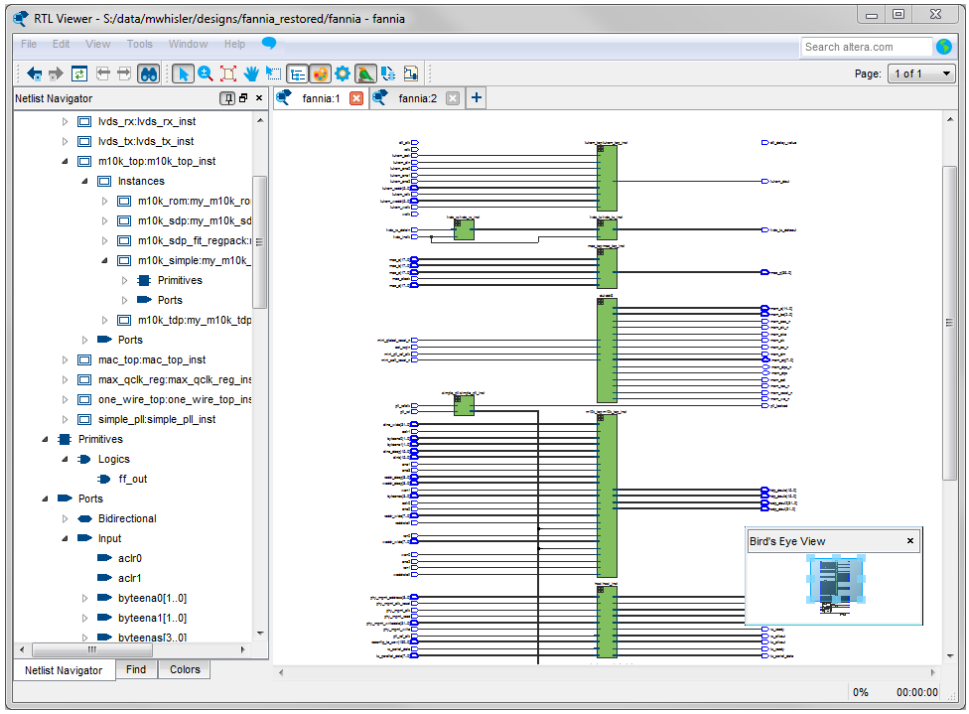
2.6. Netlist Viewer User Interface

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 3. RTL Viewer

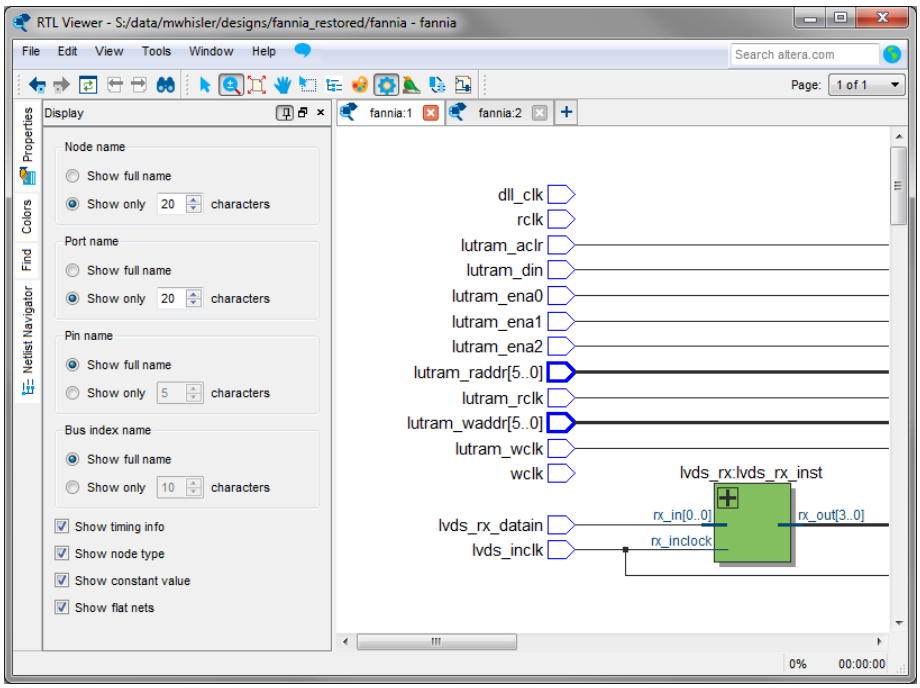


Netlist Viewers also contain a toolbar that provides tools to use in the schematic view.

- Use the **Back** and **Forward** buttons to switch between schematic views. You can go forward only if you have not made any changes to the view since going back. These commands do not undo an action, such as selecting a node. The Netlist Viewer caches up to ten actions including filtering, hierarchy navigation, netlist navigation, and zoom actions.
- 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 Page** button resets the schematic view to encompass the entire design.
- 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.

- 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 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.

Figure 4. Display Settings



- 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.

You can have only one RTL Viewer, one Technology Map Viewer (Post-Fitting), one Technology Map Viewer (Post-Mapping), and one State Machine Viewer 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

- RTL Viewer Overview on page 18
- Technology Map Viewer Overview on page 20
- Netlist Navigator Pane on page 24
- Netlist Viewers Find Pane on page 26
- Properties Pane on page 24

2.6.1. 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 3. Netlist Navigator Pane Elements

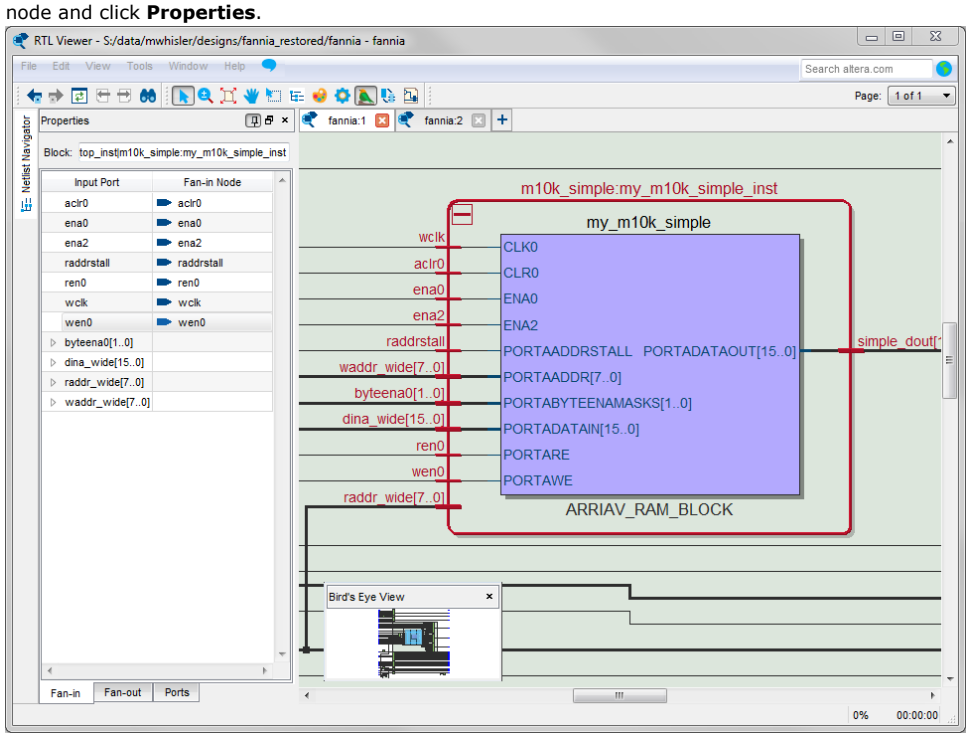
Elements	Description
Instances	Modules or instances in the design that can be expanded to lower hierarchy levels.
State Machines	State machine instances in the design that can be viewed in the State Machine Viewer.
Primitives	Low-level nodes that cannot be expanded to any lower hierarchy level. These primitives include: • Registers and gates that you can view in the RTL Viewer when using Intel Quartus Prime integrated 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.

2.6.2. Properties Pane

You can view the properties of an instance or primitive with the **Properties** pane.

Figure 5. 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 4. 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.

2.6.3. Netlist Viewers Find Pane

You can narrow the range of the search process by setting the following options in the **Find** pane:

- Click **Browse** in the **Find** pane to specify the hierarchy level of the search. In the **Select Hierarchy Level** dialog box, select the particular instance you want to search.
- Turn on the **Include subentities** option to include child hierarchies of the parent instance during the search.
- Click **Options** to open the **Find Options** dialog box. Turn on **Instances**, **Nodes**, **Ports**, or any combination of the three to further refine the parameters of the search.

When you click the **List** 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.

2.7. 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.

2.7.1. 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

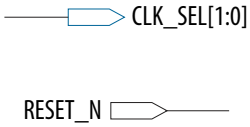
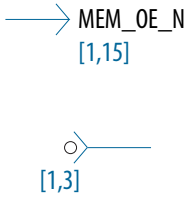
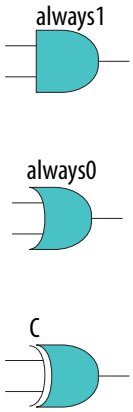
2.7.2. Schematic Symbols

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.

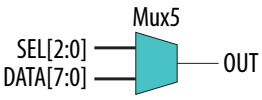
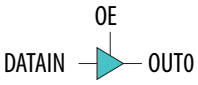
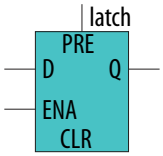
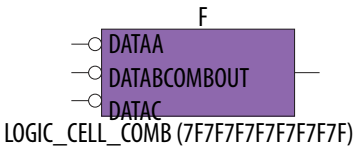
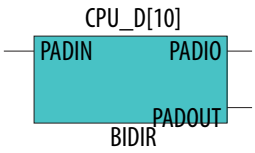
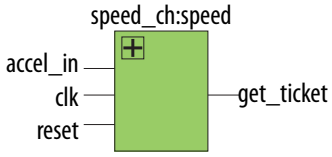
Note: The logic gates and operator primitives appear only in the RTL Viewer. Logic in the Technology Map Viewer is represented by atom primitives, such as registers and LCELLs.

Table 5. 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
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, CARRY, 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...

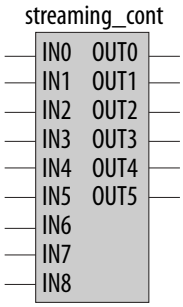
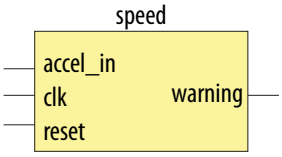
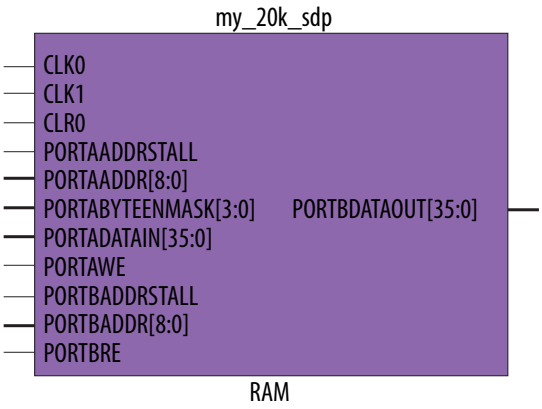
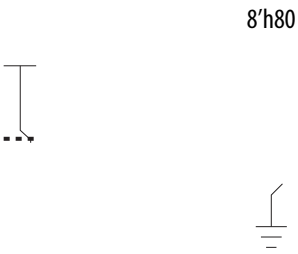
Symbol	Description
	
State Machine Instance 	A finite state machine instance in the design.
RAM 	A synchronous memory instance with registered inputs and optionally registered outputs. The symbol shows the device family and the type of memory block. This figure shows a true dual-port memory block in 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.

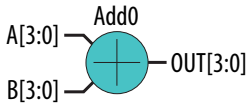
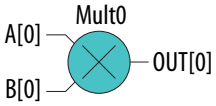
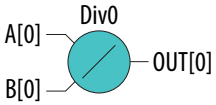
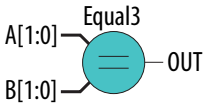
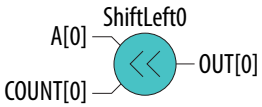
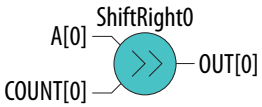
Table 6. Symbol Available Only in the State Machine Viewer

The following table lists and describes the symbol open only in the State Machine Viewer.

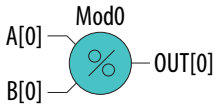
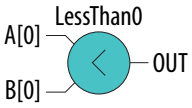
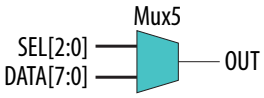
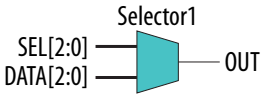
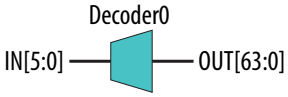
Symbol	Description
State Node	The node representing a state in a finite state machine. State transitions are indicated with arcs between state nodes. The double circle border indicates the state connects to logic outside the state machine, and a single circle border indicates the state node does not feed outside logic.

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`
	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)`

continued...

Symbol	Description
	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 $2^{\text{sel range size}}$
	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 36
- Follow Nets Across Schematic Pages on page 37

2.7.3. 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 24

2.7.4. 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.

2.7.5. 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, ports of a node, or states in a state machine 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.

When filtering in a state diagram in the State Machine Viewer, sources and destinations refer to the previous and next transition states or paths between transition states in the state diagram. The transition table and encoding table also reflect the filtering.

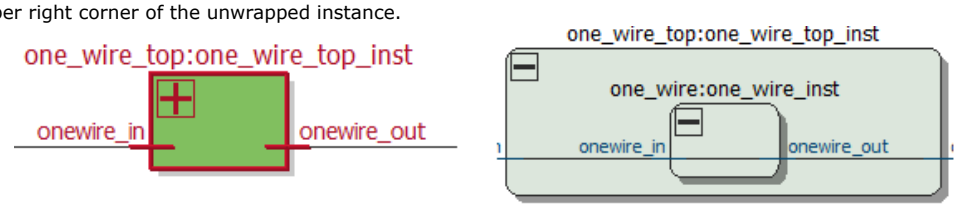
2.7.6. View Contents of Nodes in the Schematic View

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 6. 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 7. 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 .

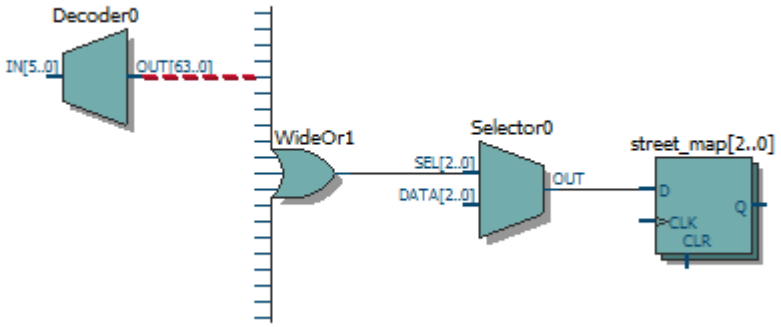


Figure 8. Display Nets Across Hierarchies

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.

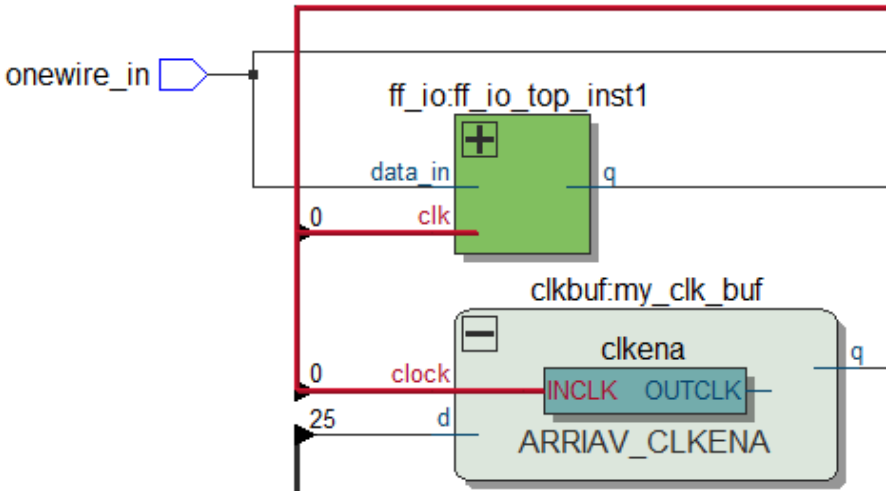
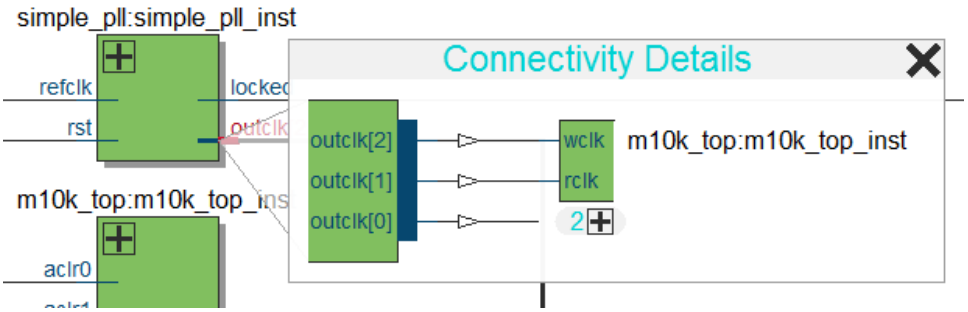


Figure 9. Show Connectivity Details

You can select a bus port or bus pin and click **Connectivity Details** in the context menu for that object.



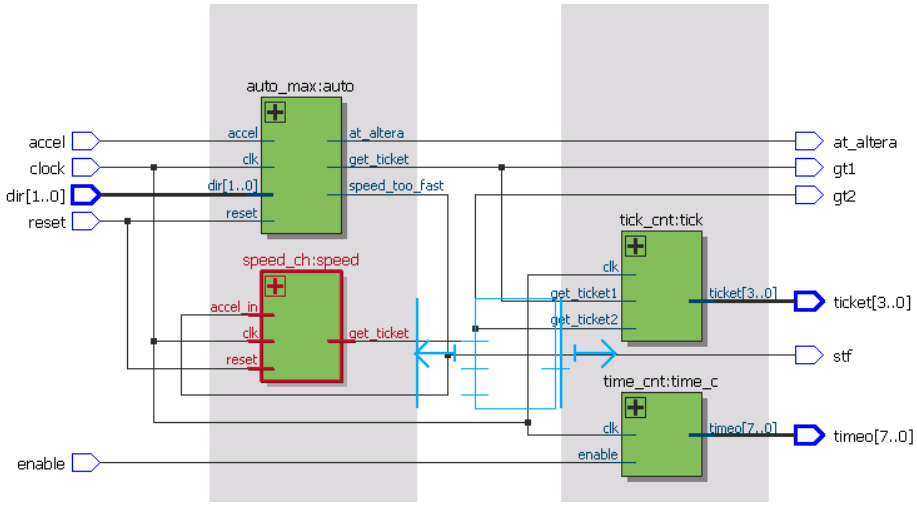
You can double-click objects in the **Connectivity Details** window to navigate to them quickly. If the plus symbol appears, you can further unwrap objects in the view. This can be very useful when tracing a signal in a complex netlist.

2.7.7. 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 10. Legal Placement when Moving Nodes



To restore the schematic view to its default arrangement, right-click and click **Refresh**.

2.7.8. 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.

Related Information

Properties Pane on page 24

2.7.9. 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 32

2.7.10. 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.

Viewing the entire schematic can be useful when debugging and tracing through a large netlist. The Intel 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.

2.7.11. 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 21

2.7.12. 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 27

2.8. State Machine Viewer

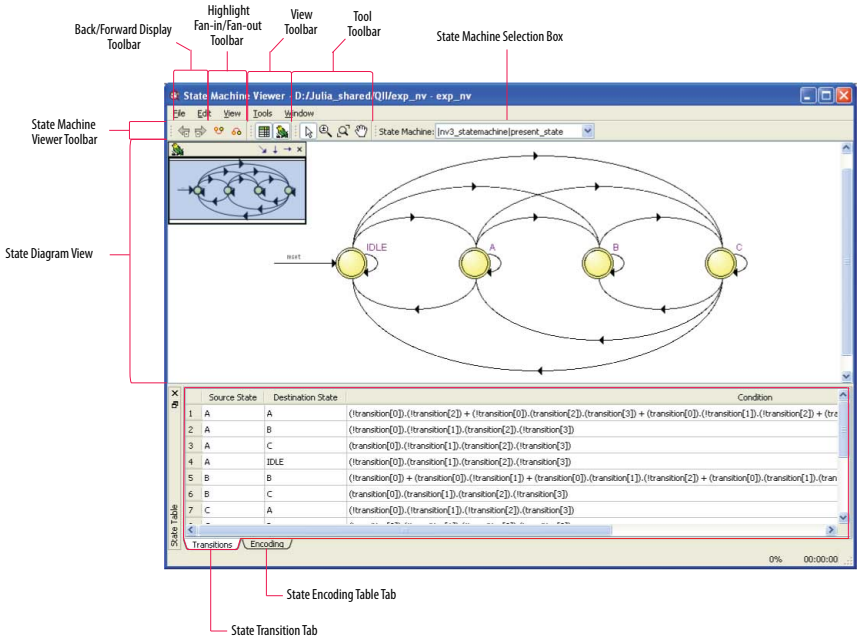
The State Machine Viewer displays a graphical representation of the state machines in your design.

You can open the State Machine Viewer in any of the following ways:

- On the Tools menu, point to **Netlist Viewers** and click **State Machine Viewer**.
- Double-click a state machine instance in the RTL Viewer

Figure 11. The State Machine Viewer

The following figure shows an example of the State Machine Viewer for a simple state machine and lists the components of the viewer.



2.8.1. State Diagram View

The state diagram view appears at the top of the State Machine Viewer. It contains a diagram of the states and state transitions.

The nodes that represent each state are arranged horizontally in the state diagram view with the initial state (the state node that receives the reset signal) in the left-most position. Nodes that connect to logic outside of the state machine instance are represented by a double circle. The state transition is represented by an arc with an arrow pointing in the direction of the transition.

When you select a node in the state diagram view, and turn on the **Highlight Fan-in** or **Highlight Fan-out** command from the View menu or the State Machine Viewer toolbar, the respective fan-in or fan-out transitions from the node are highlighted in red.

Note: An encrypted block with a state machine displays encoding information in the state encoding table, but does not display a state transition diagram or table.

2.8.2. State Transition Table

The state transition table on the **Transitions** tab at the bottom of the State Machine Viewer displays the condition equation for each state transition.

Each row in the table represents a transition (each arc in the state diagram view). The table has the following columns:

- **Source State**—the name of the source state for the transition
- **Destination State**—the name of the destination state for the transition
- **Condition**—the condition equation that causes the transition from source state to destination state

To see all of the transitions to and from each state name, click the appropriate column heading to sort on that column.

The text in each column is left-aligned by default; to change the alignment and to make it easier to see the relevant part of the text, right-click the column and click **Align Right**. To revert to left alignment, click **Align Left**.

Click in any cell in the table to select it. To select all cells, right-click in the cell and click **Select All**; or, on the Edit menu, click **Select All**. To copy selected cells to the clipboard, right-click the cells and click **Copy Table**; or, on the Edit menu, point to **Copy** and click **Copy Table**. You can paste the table into any text editor as tab-separated columns.

2.8.3. State Encoding Table

The state encoding table on the **Encoding** tab at the bottom of the State Machine Viewer displays encoding information for each state transition.

To view state encoding information in the State Machine Viewer, you must synthesize your design with the **Start Analysis & Synthesis** command. If you have only elaborated your design with the **Start Analysis & Elaboration** command, the encoding information is not displayed.

2.8.3.1. Select Items in the State Machine Viewer

You can select and highlight each state node and transition in the State Machine Viewer. To select a state transition, click the arc that represents the transition.

When you select a node or transition arc in the state diagram view, the matching state node or equation conditions in the state transition table are highlighted; conversely, when you select a state node or equation condition in the state transition table, the corresponding state node or transition arc is highlighted in the state diagram view.

2.8.4. Switch Between State Machines

A design may contain multiple state machines. To choose which state machine to view, use the **State Machine** selection box located at the top of the State Machine Viewer. Click in the drop-down box and select the necessary state machine.

2.9. Cross-Probing to a Source Design File and Other Intel Quartus Prime Windows

The RTL Viewer, Technology Map Viewer, and State Machine Viewer allow you to cross-probe to the source design file and to various other windows in the Intel 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 Intel 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.

2.10. Cross-Probing to the Netlist Viewers from Other Intel Quartus Prime Windows

You can cross-probe to the RTL Viewer and Technology Map Viewer from other windows in the Intel 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, State Machine Viewer, and Technology Map Viewer, and you can locate nodes in the RTL Viewer and Technology Map Viewer from the following Intel Quartus Prime software windows:

- Project Navigator
- Timing Closure Floorplan
- 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 Intel 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.

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**.

2.11. 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 *Intel Quartus Prime Standard 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

Report Timing (Dialog Box)
In *Intel Quartus Prime Standard Edition User Guide: Timing Analyzer*

2.12. Optimizing the Design Netlist Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • Added link to *Viewing a Timing Path*.
2016.05.03	16.0.0	Removed Schematic Viewer topic.
2015.11.02	15.1.0	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 Intel Quartus Prime software version 10.0, including major user interface changes
November 2009	9.1.0	• Updated devices • Minor text edits

continued...

Document Version	Intel 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 Intel Quartus Prime software version 8.0

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

3. Timing Closure and Optimization

This chapter describes techniques to improve timing performance when designing for Intel devices. The application techniques vary between designs. Applying each technique does not always improve results.

Default settings and options in the Intel Quartus Prime software provide the best trade-off between compilation time, resource utilization, and timing performance. You can adjust these settings to determine whether other settings provide better results for your design.

3.1. 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 Intel 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).

3.2. Critical Paths

Critical paths are timing paths in your design that have a negative slack. 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Related Information

- Reducing Critical Path Delay on page 9
- Displaying Path Reports with the Timing Analyzer on page 60

3.2.1. 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.

3.3. Design Evaluation for Timing Closure

Follow the guidelines in this section when you encounter timing failures in a design. The guidelines show you how to evaluate compilation results of a design and how to address problems. While the guideline does not cover specific examples of restructuring RTL to improve design speed, the analysis techniques help you to evaluate changes to RTL that can help you to close timing.

3.3.1. Review Compilation Results

3.3.1.1. Review Messages

After compiling your design, review the messages in each section of the compilation report.

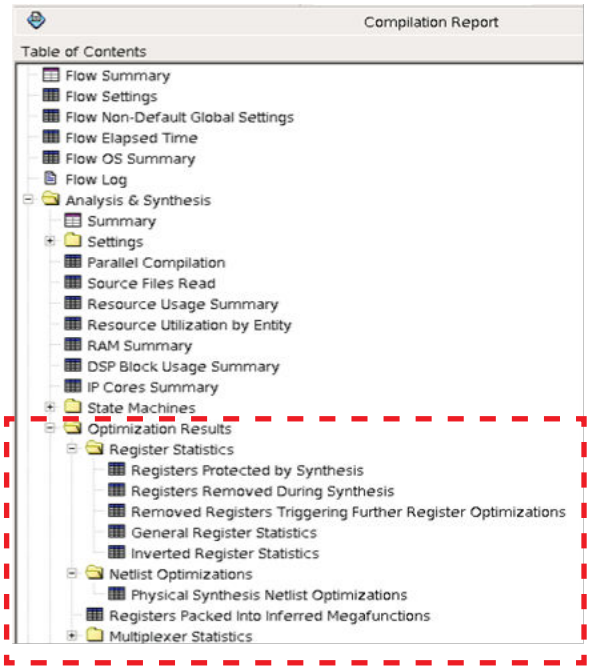
Most designs that fail timing start out with other problems that the Fitter reports as warning messages during compilation. Determine what causes a warning message, and whether to fix or ignore the warning.

After reviewing the warning messages, review the informational messages. Take note of anything unexpected, for example, unconnected ports, ignored constraints, missing files, and assumptions or optimizations that the software made.

3.3.1.2. Evaluate Physical Synthesis Results

If you enable physical synthesis options, the Compiler can duplicate and retime registers, and modify combinatorial logic during synthesis. After compilation, review the Optimization Results reports in the Analysis & Synthesis section. The reports list the optimizations performed by the physical synthesis optimizations, such as register duplication, retiming, and removal. These reports can be found in the Compilation Report panel.

Figure 12. Optimization Results Reports



When you enable physical synthesis, the compilation messages include a information about the physical synthesis algorithm performance improvement. The reported improvement is the sum of the largest improvement in each timing-critical clock domain. Although typically similar, the values for the slack improvements vary per compilation due to the random starting point of compilation algorithms.

3.3.1.3. Evaluate Fitter Netlist Optimizations

The Fitter can also perform optimizations to the design netlist. Major changes include register packing, duplicating or deleting logic cells, inverting signals, or modifying nodes in a general way such as moving an input from one logic cell to another. Find and review these reports in the Netlist Optimizations results of the Fitter section.

3.3.1.4. 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.

3.3.1.5. Evaluate Resource Usage

Evaluate a variety of resources used in the design, including global and non-global signal usage, routing utilization, and clustering difficulty.

3.3.1.5.1. Global and Non-Global Usage

For 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. You can find these reports in the Resource section under Fitter in the **Compilation Report** panel.

The figure shows an example of inefficient use of a global clock. The highlighted line has a single fan-out from a global clock.

Figure 13. Inefficient Use of a 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.

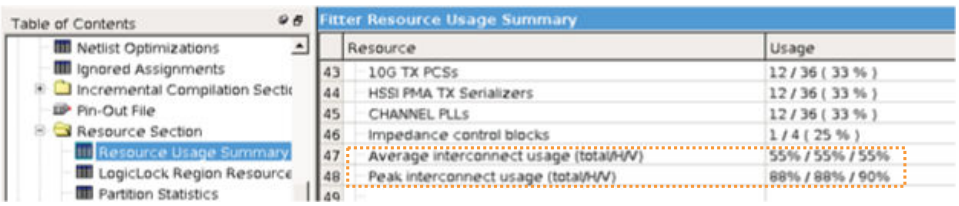
If there is routing congestion in the design, and there are high fan-out non-global nodes in the congested area, consider using global or regional signals to fan-out the nodes, or duplicate the high fan-out registers so that each of the duplicates can have fewer fan-outs.

Use the Chip Planner to locate high fan-out nodes, to report routing congestion, and to determine whether the alternatives are viable.

3.3.1.5.2. Routing Usage

Review routing usage reported in the **Fitter Resource Usage Summary** report.

Figure 14. Fitter Resource Usage Summary Report

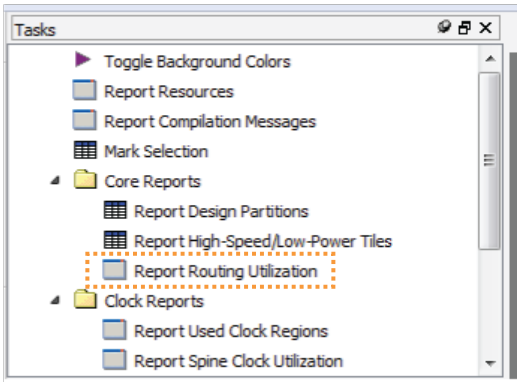


Average interconnect usage reports the average amount of interconnect that is used, out of what is available on the device. **Peak interconnect usage** reports the largest amount of interconnect used in the most congested areas.

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.

The figure shows the **Report Routing Utilization** report.

Figure 15. Report Routing Utilization Report

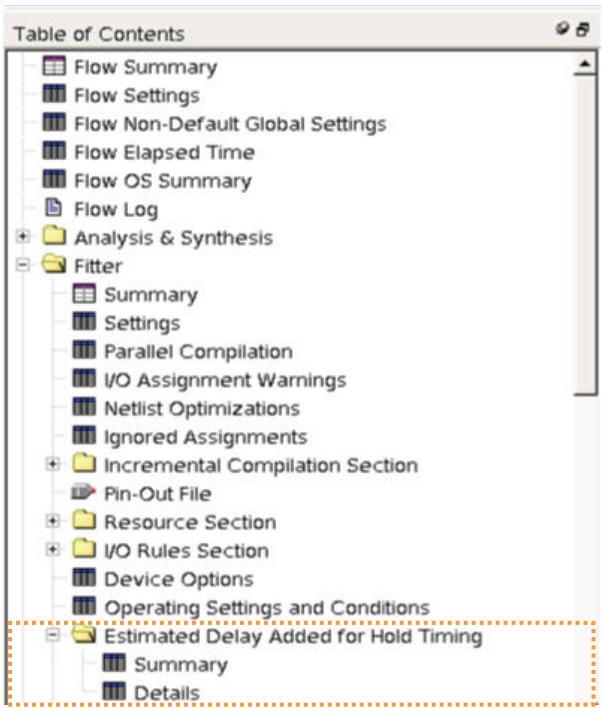


3.3.1.5.3. Wires Added for Hold

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 16. Estimated Delay Added for Hold Timing Report

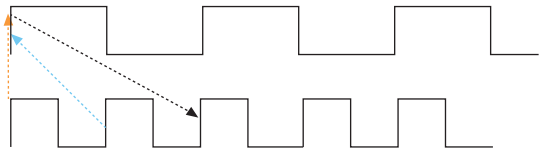


An example of an incorrect constraint which can cause the router to add wire for hold requirements is when there is data transfer from 1x to 2x clocks. Assume the design intent is to allow two cycles per transfer. Data can arrive any time in the two destination clock cycles by adding a multicycle setup 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 17. Timing Requirement Relaxed Waveform



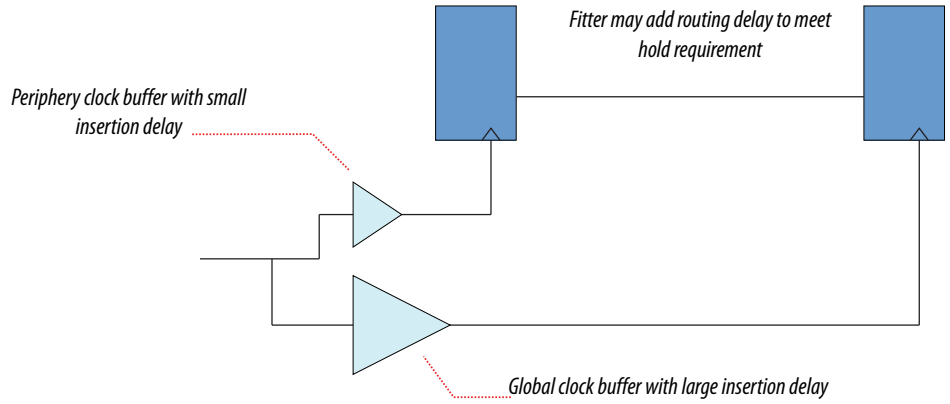
The default hold requirement, shown with the dashed blue line, can force the router to add wire to guarantee that data is delayed by one cycle. To correct the hold requirement, add a multicycle constraint with a hold option.

```
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.

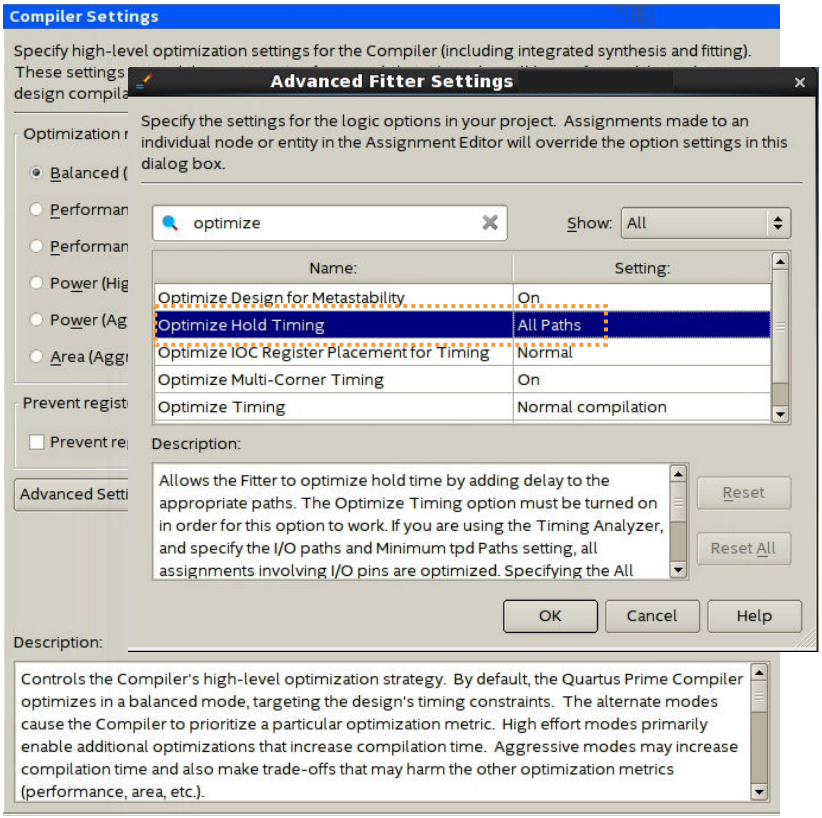
Figure 18. Clock Delay



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.

Figure 19. Optimize Hold Timing Option



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.

3.3.1.6. Evaluate Other Reports and Adjust Settings Accordingly

3.3.1.6.1. 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 (Standard) 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.

3.3.1.6.2. Review Ignored Assignments

The **Compilation Report** includes details of any assignments ignored by the Fitter. Assignments typically get ignored if design names change, but assignments are not updated. Make sure any intended assignments are not being ignored.

3.3.1.6.3. Review Non-Default Settings

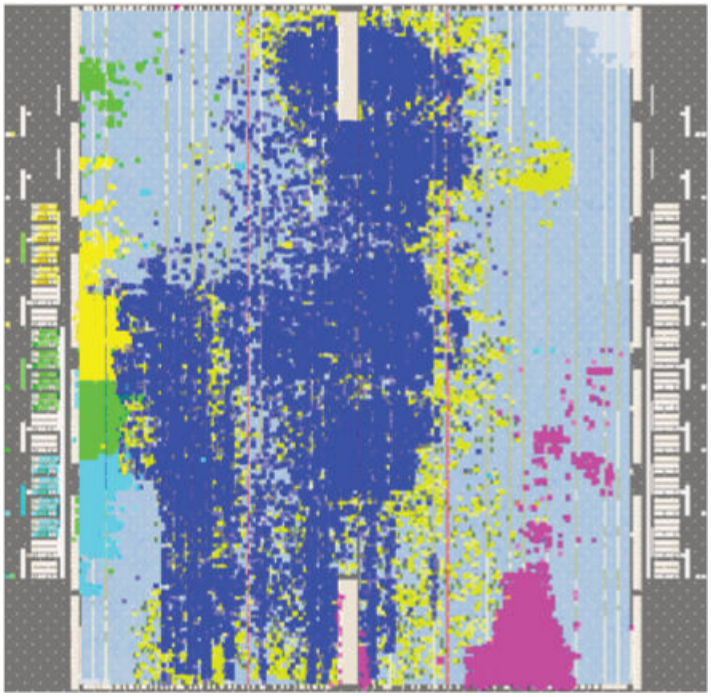
The reports from Synthesis and Fitter show non-default settings used in a compilation. Review the non-default settings to ensure the design benefits from the change.

3.3.1.6.4. Review 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 20. Floorplan with Color-Coded Entities



The following notes describe how you can use the visualization in *Floorplan with Color-Coded Entities* to check timing paths:

- 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.

3.3.1.6.5. Evaluate Placement and Routing

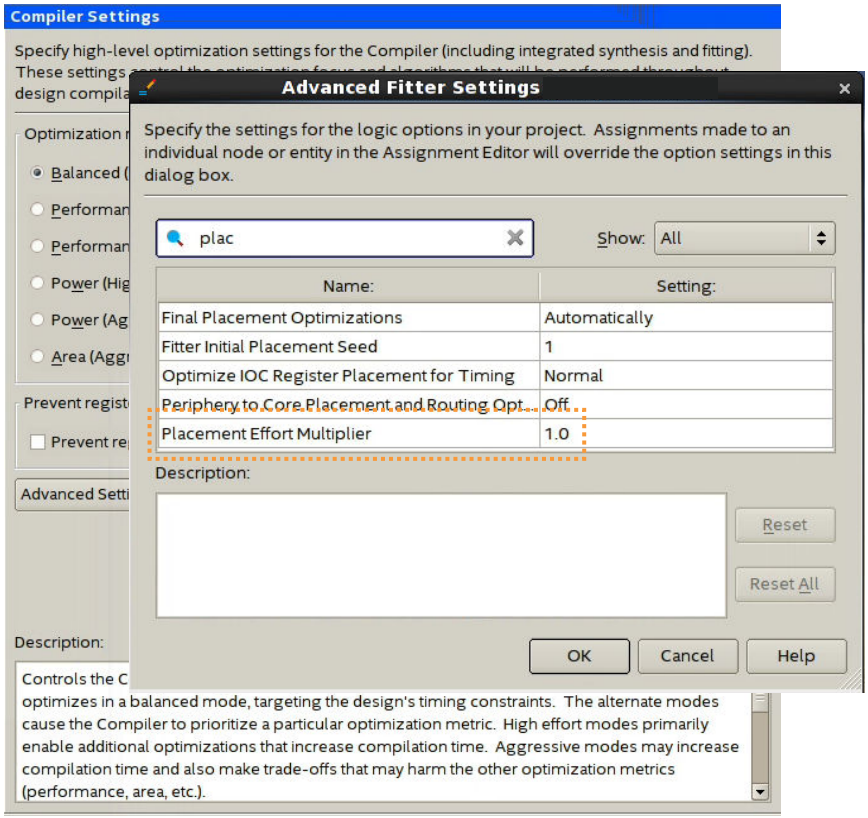
Review duration of parts of compile time in Fitter messages. If routing takes much more time than placement, then meeting timing may be more difficult than the placer predicted.

3.3.1.6.6. 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.

Figure 21. Placement Effort Multiplier



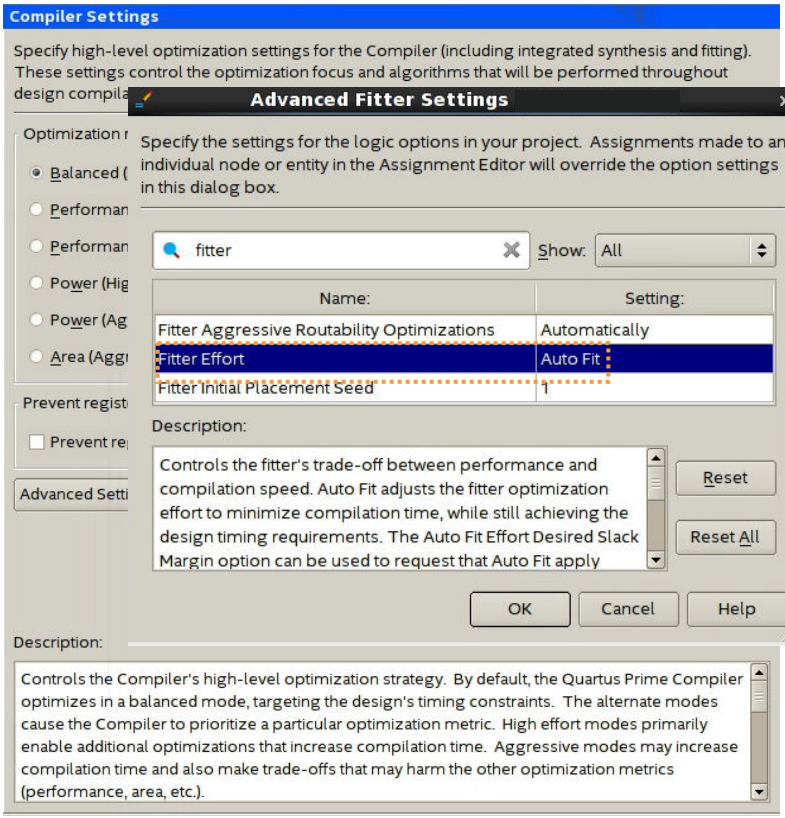
3.3.1.6.7. Adjust Fitter Effort

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)**, which reduces Fitter effort once timing requirements are met. You can optionally select another **Optimization mode** to target performance, power, or resource usage.

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.

Figure 22. Fitter Effort



3.3.1.6.8. Review Timing Constraints

Ensure that clocks are constrained with the correct frequency requirements. Using the `derive_pll_clocks` assignment keeps generated clock settings updated. Timing Analyzer can be useful in reviewing SDC constraints. For example, under **Diagnostic** in the Task panel, the **Report Ignored Constraints** report shows any incorrect names in the design, most commonly caused by changes in the design hierarchy. Use the **Report Unconstrained Paths** report to locate unconstrained paths. Add constraints as necessary so that the design can be optimized.

3.3.1.7. 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.

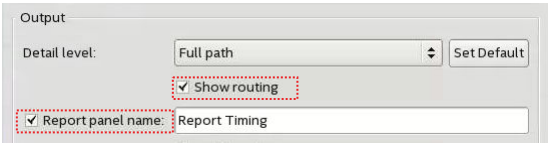
3.3.2. Review Details of Timing Paths

3.3.2.1. Show Timing Path Routing

Showing routing for a path can help uncover unusual routing delays.

In the Timing Analyzer **Report Timing** dialog box, enable the **Report panel name** and **Show routing** options, and click **Report Timing**.

Figure 23. Report Pane and Show Routing Options



The **Extra Fitter Information** tab shows a miniature floorplan with the path highlighted.

You can also locate the path in the Chip Planner to examine routing congestion, and to view whether nodes in a path are placed close together or far apart.

Related Information

Exploring Paths in the Chip Planner on page 110

3.3.2.2. Global Network Buffers

Routing paths allow you to identify global network buffers that fail timing. Buffer locations are named according to the network they drive.

- CLK_CTRL_Gn—for Global driver
- CLk_CTRL_Rn—for Regional driver

Buffers to access the global networks are located in the center of each side of the device. Buffering to route a core logic signal on a global signal network causes insertion delay. Tradeoffs 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 is demoted to local routing.

3.3.2.2.1. Source Location

If the register feeding the global buffer cannot be moved closer, then consider changing either the design logic or the routing type.

3.3.2.2.2. Insertion Delay

If a global signal is required, consider adding half a cycle to timing by using a negative-edge triggered register to generate the signal (top figure) and use a multicycle setup constraint (bottom figure).

Figure 24. Negative-Edge Triggered Register

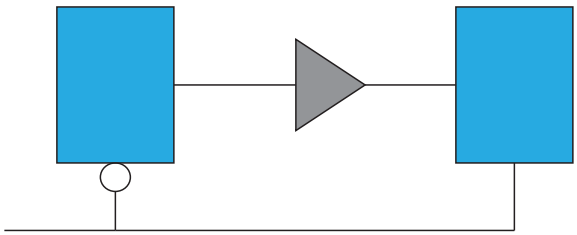
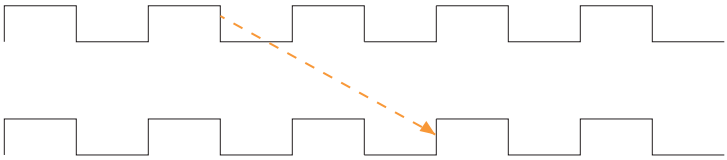


Figure 25. Multicycle Setup Constraint



```
set_multicycle_path -from <generating_register> -setup -end 2
```

3.3.2.2.3. 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.

3.3.2.2.4. Global Networks

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**.

Figure 26. Global Signal Assignment

To	Assignment Name	Value	Enabled
reg_clk	Global Signal	Off	Yes

3.3.2.3. Resets and Global Networks

Reset signals are often routed 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.

3.3.2.4. 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.

3.3.2.5. Logic Depth

The **Statistics** tab in the Timing Analyzer path report shows the levels of logic in a path. If the path fails timing and the number of logic levels is high, consider adding pipelining in that part of the design.

3.3.2.6. 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 names of the converted registers include "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 is close to full, 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.

3.3.2.7. Clocking Architecture

For better timing results, place registers driven by a regional clock in one quadrant of the chip. You can review the clock region boundaries using 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 - 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 (Standard) 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 the Device on page 105

3.3.2.8. Timing Closure Recommendations

The Report Timing Closure Recommendations task in the Timing Analyzer analyzes paths and provides specific recommendations based on path characteristics.

3.3.3. Adjusting and Recompiling

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 (Standard) regions, and re-evaluate paths that fail timing.

Partitioning often does not help timing closure, and must be done at the beginning of the design process. Adding partitions can increase logic utilization if it prevents cross-boundary optimizations, making timing closure harder and increasing compile times.

3.3.3.1. Using Partitions to Achieve Timing Closure

One technique to achieve timing closure is confining failing paths within individual design partitions, such that there are no failing paths passing between partitions. You can then use incremental make changes as necessary to correct the failing paths, and recompile only the affected partitions.

To use this technique:

1. In the Design Partition Planner, load timing data by clicking **View ➤ Show Timing Data**.

   Entities containing nodes on failing paths appear in red in the Design Partition Planner.
2. Extract the entity containing failing paths by dragging it outside of the top-level entity window.
   - If there are no failing paths between the extracted entity and the top-level entity, right-click the extracted entity, and then click **Create Design Partition** to place that entity in its own partition.
3. Keep failing paths within a partition, so that there are no failing paths crossing between partitions.

   If you are unable to isolate the failing paths from an extracted entity so that none are crossing partition boundaries, return the entity to its parent without creating a partition.
4. Find the partition having the worst slack value. For all the other partitions, preserve the contents and set as **Empty**.

For information about preserving the contents of a partition, refer to *Incremental Block-Based Compilation Flow* in the *Intel Quartus Prime Pro Edition User Guide: Block-Based Design*.

5. Adjust the logic in the partition and rerun the Fitter as necessary until the partition meets the timing requirements.
6. Repeat the process for all other design partitions with failing paths.

Related Information

- Using Block-Based Compilation
- Incremental Block-Based Compilation Flow

3.4. Design Analysis

The initial compilation establishes whether the design achieves a successful fit and meets the specified timing requirements. This section describes how to analyze your design results in the Intel Quartus Prime software.

3.4.1. Ignored Timing Constraints

The Intel Quartus Prime software ignores illegal, obsolete, and conflicting constraints.

You can view a list of ignored constraints in the Timing Analyzer GUI by clicking **Reports ➤ Report Ignored Constraints** or by typing the following command to generate a list of ignored timing constraints:

```
report_sdc -ignored -panel_name "Ignored Constraints"
```

Analyze any constraints that the Intel Quartus Prime software ignores. If necessary, correct the constraints and recompile your design before proceeding with design optimization.

You can view a list of ignored assignment in the **Ignored Assignment Report** generated by the Fitter.

Related Information

Creating I/O Requirements

3.4.2. I/O Timing

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 Intel 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

Creating I/O Requirements

3.4.3. 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.

3.4.3.1. Displaying Path Reports with the Timing Analyzer

The Timing Analyzer generates reports with information about all valid register-to-register paths. To view all timing summaries, double-click **Report All Summaries** in the **Tasks** pane.

If any clock domains have failing paths (highlighted in red in the **Report** pane), right-click the clock name listed in the **Clocks Summary** pane and select **Report Timing** to get more details.

When you select a path in the **Summary of Paths** tab, the path detail pane displays all the path information. The **Extra Fitter Information** tab offers visual representation of the path location on the physical device. This can reveal whether the timing failure is distance related, due to the source and destination node being too close or too far.

The **Data Path** tab displays the Data Arrival Path and the Data Required Path. You can determine the path segments contributing the most to the timing violations with the incremental information. The **Waveform** tab shows the signals in the time domain, and plots the slack between arrival data and required data.

The RTL Viewer or Technology Map Viewer provide schematic (gate-level or technology-mapped) representations of the design netlist, and can help you to assess which areas in a design can benefit from reducing the number of logic levels. To locate a timing path in one of the viewers, right-click a path in the timing report, point to **Locate**, and select either **Locate in RTL Viewer** or **Locate in Technology Map Viewer**. You can also investigate the physical layout of a path in detail with the Chip Planner.

Related Information

- When to Use the Netlist Viewers: Analyzing Design Problems
- Generating Timing Reports

3.4.3.2. 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 Intel 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 110
- Design Evaluation for Timing Closure on page 44

3.4.3.3. 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 27. 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.
- 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_clock_transfers

3.4.3.4. Tips for Analyzing Paths from/to the Source and Destination of Critical Path

When analyzing the failing paths in a design, it is often helpful to get a fuller picture of the interactions around the paths.

To understand what may be pulling on a critical path, the following `report_timing` command can be useful.

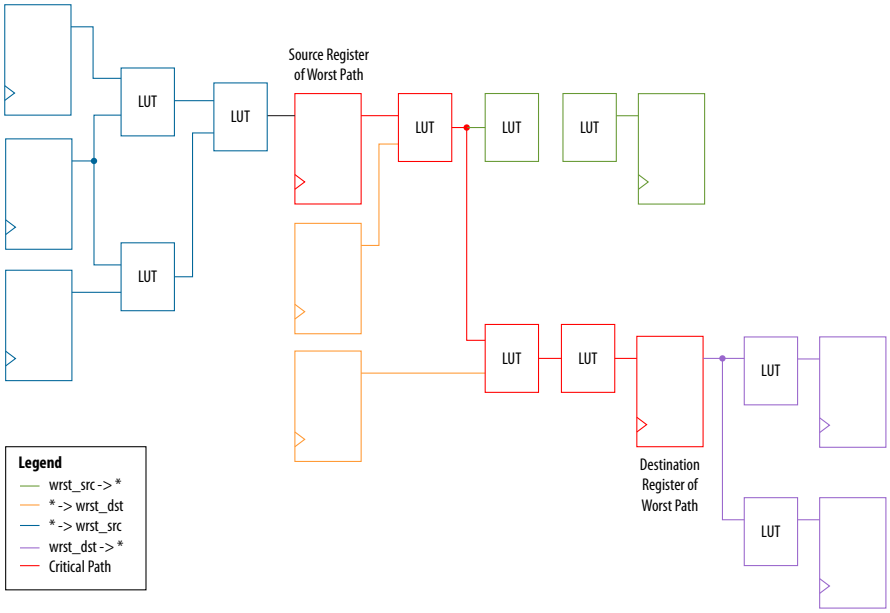
1. In the project directory, run the `report_timing` command to find the nodes in a critical path.
2. Copy the code below in a `.tcl` file, and replace the first two variable with the node names from the **From Node** and **To Node** columns of the worst path. The script analyzes the path between the 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 fanouts.

   The figure shows a simplified example of what these reports analyzed.

Figure 28. 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.

3.4.3.5. 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

3.4.3.6. 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 Intel 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.

3.5. Timing Optimization

Use the following guidelines if your design does not meet its timing requirements.

3.5.1. Displaying Timing Closure Recommendations for Failing Paths

Use the **Timing Closure Recommendations** report to get specific recommendations about failing paths in your design and changes that you can make to potentially fix the failing paths.

1. In the **Tasks** pane of the Timing Analyzer, select the **Report Timing Closure Recommendations** task to open the **Report Timing Closure Recommendations** dialog box.
2. Select paths based on the clock domain, filter by nodes on path, and choose the number of paths to analyze.
3. After running the **Report Timing Closure Recommendations** task in the Timing Analyzer, examine the reports in the **Report Timing Closure Recommendations** folder in the **Report** pane of the Timing Analyzer GUI. Each recommendation has star symbols (*) associated with it. Recommendations with more stars are more likely to help you close timing on your design.

The reports give you the most probable causes of failure for each analyzed path, and show recommendations that may help you fix the failing paths.

The reports are organized into sections, depending on the type of issues found in the design, such as large clock skew, restricted optimizations, unbalanced logic, skipped optimizations, coding style that has too many levels of logic between registers, or region or partition constraints specific to your project.

For detailed analysis of the critical paths, run the `report_timing` command on specified paths. In the **Extra Fitter Information** tab of the **Path** report panel, you can see detailed fitter-related information that may help you visualize the issue.

Related Information

- Displaying Timing Closure Recommendations for Failing Paths on page 64
- Report Timing Closure Recommendations Dialog Box

3.5.2. Timing Optimization Advisor

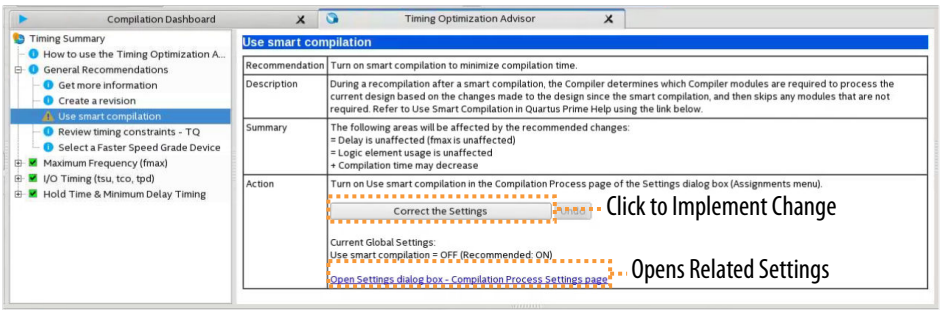
While the Timing Analyzer **Report Timing Closure Recommendations** task gives specific recommendations to fix failing paths, the Timing Optimization Advisor gives more general recommendations to improve timing performance for a design.

The Timing Optimization Advisor guides you in making settings that optimize your design to meet your timing requirements. To run the Timing Optimization Advisor click **Tools ➤ Advisors ➤ Timing Optimization Advisor**. This advisor describes many of the suggestions made in this section.

When you open the Timing Optimization Advisor after compilation, you can find recommendations to improve the timing performance of your design. If suggestions in these advisors contradict each other, evaluate these options and choose the settings that best suit the given requirements.

The example shows the Timing Optimization Advisor after compiling a design that meets its frequency requirements, but requires setting changes to improve the timing.

Figure 29. Timing Optimization Advisor



When you expand one of the categories in the Timing Optimization Advisor, such as **Maximum Frequency (fmax)** or **I/O Timing (tsu, tco, tpd)**, the recommendations appear in stages. These stages show the order in which to apply the recommended settings.

The first stage contains the options that are easiest to change, make the least drastic changes to your design optimization, and have the least effect on compilation time.

Icons indicate whether each recommended setting has been made in the current project. In the figure, the checkmark icons in the list of recommendations for Stage 1 indicates recommendations that are already implemented. The warning icons indicate recommendations that are not followed for this compilation. The information icons indicate general suggestions. For these entries, the advisor does not report whether these recommendations were followed, but instead explains how you can achieve better performance. For a legend that provides more information for each icon, refer to the "How to use" page in the Timing Optimization Advisor.

Each recommendation provides a link to the appropriate location in the Intel Quartus Prime GUI where you can change the settings. For example, consider the **Synthesis Netlist Optimizations** page of the **Settings** dialog box or the **Global Signals category** in the Assignment Editor. This approach provides the most control over which settings are made and helps you learn about the settings in the software. When available, you can also use the **Correct the Settings** button to automatically make the suggested change to global settings.

For some entries in the Timing Optimization Advisor, a button allows you to further analyze your design and see more information. The advisor provides a table with the clocks in the design, indicating whether they have been assigned a timing constraint.

3.5.3. 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

3.5.3.1. Optimize Hold Timing

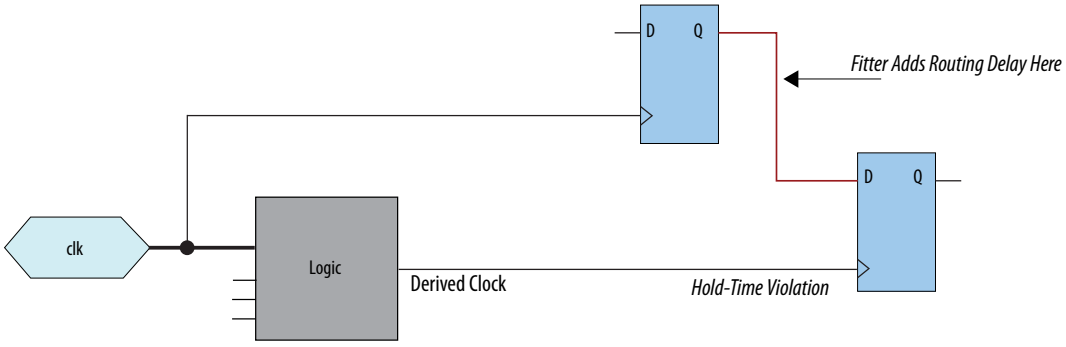
The **Optimize Hold Timing** option directs the Intel Quartus Prime software to optimize minimum delay timing constraints. Check your device information to determine whether the Intel Quartus Prime software optimizes hold timing for all paths or only for I/O paths and minimum t_{PD} paths.

When you turn on **Optimize Hold Timing** in the **Advanced Fitter Settings** dialog box, the Intel 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 30. 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

Recommended Design Practices

3.5.3.2. 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 8. 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**.

3.5.4. 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.

You can apply the suggestions this section to all Intel FPGA families and to the family of CPLDs.

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 9. 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

continued...

Order	Technique	Affects t_{SU}	Affects t_{CO}
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	**Use the Fast Regional Clock** (refer to *Change How Hold Times are Optimized for MAX® II Devices*)	N/A	Yes
11	For MAX II or MAX V family devices, set **Guarantee I/O Paths Have Zero Hold Time at Fast Corner** to **OFF**, or **When T_{SU} and T_{PD} Constraints Permit** (refer to *Change How Hold Times are Optimized for MAX II Devices*)	Yes	N/A
12	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

Optimize IOC Register Placement for Timing Logic Option on page 69
Fast Input, Output, and Output Enable Registers on page 70
Programmable Delays on page 70
Use PLLs to Shift Clock Edges on page 71
Use Fast Regional Clock Networks and Regional Clocks Networks on page 71
Spine Clock Limitations on page 72
Change How Hold Times are Optimized for Devices on page 72

Related Information

Required Settings for Initial Compilation on page 7

3.5.4.1. 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.

Note: The option does not apply to series devices because they do not contain I/O registers.

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 Intel Quartus Prime Fitter optimizations.

- Have combinational logic between the register and the pin
- Are part of a carry or cascade 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

3.5.4.2. 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.

In series devices, which have no I/O registers, these assignments lock the register into the LAB adjacent to the I/O pin if there is a pin location assignment for that I/O pin.

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 Intel 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 (Standard) 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.

In series devices, the Fitter duplicates the register and places it in each distinct LAB location that is next to an I/O pin with a pin location assignment.

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 Intel Quartus Prime Help.

Related Information

- Fast Input Register logic option
- Fast Output Register logic option
- Fast Output Enable Register logic option
- Fast OCT Register logic option

3.5.4.3. 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.

For Arria®, Cyclone®, MAX II, MAX V, and Stratix® series devices, the Intel 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 Intel 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 Intel Quartus Prime Help topics:

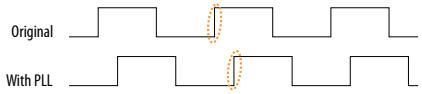
Related Information

- Input Delay from Pin to Input Register logic option
- Input Delay from Pin to Internal Cells logic option
- Output Enable Pin Delay logic option
- Delay from Output Register to Output Pin logic option
- Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations logic option

3.5.4.4. 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 31. 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**.

3.5.4.5. 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.

3.5.4.6. Spine Clock Limitations

In projects with high clock routing demands, limitations in the Intel Quartus Prime software can cause spine clock errors. These errors are often seen with designs using multiple memory interfaces and high-speed serial interface (HSSI) channels, especially 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 (Standard) regions, or if the Logic Lock (Standard) regions are not aligned to your clock region boundaries, create additional Logic Lock (Standard) regions and further constrain your logic.
- If Periphery features ignore Logic Lock (Standard) region assignment, possibly because the global promotion process is not functioning properly. 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 the Device on page 105
- Layers Settings and Editing Modes on page 103
- Report Spine Clock Utilization dialog box (Chip Planner)

3.5.4.7. Change How Hold Times are Optimized for Devices

For devices, you can use the **Guarantee I/O Paths Have Zero Hold Time at Fast Corner** option to control how hold time is optimized by the Intel Quartus Prime software.

3.5.5. 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

Recommended Design Practices

3.5.5.1. 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 (Standard) or location assignments.

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 Intel 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

3.5.5.2. 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 warning messages from your initial compilation and check for ignored timing assignments.
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
- Change State Machine Encoding
- Prevent Shift Register Inference
- Use Other Synthesis Options Available in Your Synthesis Tool

5. To optimize for performance using physical synthesis, apply the following options:
   - Enable physical synthesis
   - Perform automatic asynchronous signal pipelining
   - Perform register duplication
   - Perform register retiming
   - Perform logic to memory mapping
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 (Standard) 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

Initial Compilation: Required Settings

3.5.5.3. Physical Synthesis Optimizations

The Intel 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 Intel 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 Intel Quartus Prime software can remap the circuit to improve performance, use the **Perform WYSIWYG Primitive Resynthesis** option. This option directs the Intel 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 Intel Quartus Prime technology mapper 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 Intel Quartus Prime compilation, physical synthesis optimizations make placement-specific changes to the netlist that improve speed performance results for the specific Intel device.

Note: If you want the performance gain from physical synthesis only on parts of your design, you can apply the physical synthesis options on specific instances with the Assignment Editor.

Related Information

- Perform WYSIWYG Primitive Resynthesis Logic Option
- Optimization Technique Logic Option

3.5.5.4. Turn Off Extra-Effort Power Optimization Settings

If power optimization settings are set to **Extra Effort**, your design performance can be affected. If timing performance is more important than power, set the power optimization setting to **Normal**.

Related Information

- Power Optimization
- Power Optimization Logic Option

3.5.5.5. Optimize Synthesis for Speed, 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.

You can specify logic options for specific modules in your design with the Assignment Editor while leaving the default **Optimization Technique** setting at **Balanced** (for the best trade-off between area and speed for certain device families) or **Area** (if area is an important concern). You can also use the **Speed Optimization Technique for Clock Domains** option in the Assignment Editor to specify that all combinational logic in or between the specified clock domains are optimized for speed.

To achieve best performance with push-button compilation, follow the recommendations in the following sections for other synthesis settings. You can use DSE II to experiment with different Intel Quartus Prime synthesis options to optimize your design for the best performance.

Related Information

Optimization Technique Logic Option

3.5.5.6. 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.

Note: If you are using Intel Quartus Prime incremental compilation, you cannot flatten your design across design partitions. Incremental compilation always preserves the hierarchical boundaries between design partitions. Follow Intel's recommendations for design partitioning, such as registering partition boundaries to reduce the effect of cross-boundary optimizations.

3.5.5.7. 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.

3.5.5.8. Change State Machine Encoding

State machines can be encoded using various techniques. One-hot encoding, which uses one register for every state bit, usually provides the best performance. If your design contains state machines, changing the state machine encoding to one-hot can improve performance at the cost of area.

Related Information

State Machine Processing Logic Option online help

3.5.5.9. Duplicate Logic for Fan-Out Control

Oftentimes, timing failures occur not because of the high fan-out registers, but because of the location of those registers. Duplicating registers, where source and destination registers are physically close, can help improve slack on critical paths.

Synthesis tools support options or attributes that specify the maximum fan-out of a register. When using Intel Quartus Prime integrated 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 to

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 Intel 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.

3.5.5.10. 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.

3.5.5.11. 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.

3.5.5.12. 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

Design Space Explorer II on page 12

3.5.5.13. 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

3.5.6. Logic Lock (Standard) Assignments

Using Logic Lock (Standard) assignments to improve timing performance is only recommended for older devices, such as the MAX II family. For other device families, especially for larger devices such as Arria and Stratix series devices, do not use Logic Lock (Standard) assignments to improve timing performance. For these devices, use the feature for performance preservation and to floorplan your design.

Logic Lock (Standard) assignments do not always improve the performance of the design. In many cases, you cannot improve upon results from the Fitter by making location assignments. If there are existing Logic Lock (Standard) assignments in your design, remove the assignments if your design methodology permits it. Recompile the design, and then check if the assignments are making the performance worse.

When making Logic Lock (Standard) assignments, it is important to consider how much flexibility to give the Fitter. Logic Lock (Standard) assignments provide more flexibility than hard location assignments. Assignments that are more flexible require higher Fitter effort, but reduce the chance of design overconstraint.

The following types of Logic Lock (Standard) assignments are available, listed in the order of decreasing flexibility:

- Auto size, floating location regions
- Fixed size, floating location regions
- Fixed size, locked location regions

If you are unsure about the best size or location of a Logic Lock (Standard) region, the **Auto/Floating** options are useful for your first pass. After you determine where a Logic Lock (Standard) region must go, modify the Fixed/Locked regions, as Auto/Floating Logic Lock (Standard) regions can hurt your overall performance. To determine what to put into a Logic Lock (Standard) region, refer to the timing analysis results and analyze the critical paths in the Chip Planner. The register-to-register timing paths in the Timing Analyzer section of the Compilation Report help you recognize patterns.

Related Information

Analyzing and Optimizing the Design Floorplan on page 102

3.5.6.1. Hierarchy Assignments

For a design with the hierarchy shown in the figure, which has failing paths in the timing analysis results similar to those shown in the table, mod_A is probably a problem module. In this case, a good strategy to fix the failing paths is to place the mod_A hierarchy block in a Logic Lock (Standard) region so that all the nodes are closer together in the floorplan.

Figure 32. Design Hierarchy

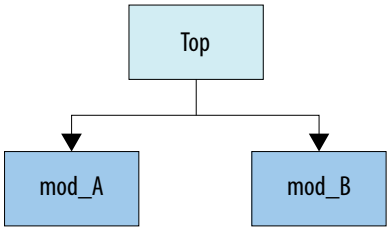


Table 10. Failing Paths in a Module Listed in Timing Analysis

From	To
|mod_A|reg1	|mod_A|reg9
|mod_A|reg3	|mod_A|reg5
|mod_A|reg4	|mod_A|reg6
|mod_A|reg7	|mod_A|reg10
|mod_A|reg0	|mod_A|reg2

Hierarchical Logic Lock (Standard) regions are also important if you are using an incremental compilation flow. Place each design partition for incremental compilation in a separate Logic Lock (Standard) region to reduce conflicts and ensure good results as the design develops. You can use the auto size and floating location regions to find a good design floorplan, but fix the size and placement to achieve the best results in future compilations.

Related Information

Analyzing and Optimizing the Design Floorplan on page 102

3.5.7. Location Assignments

If a small number of paths are failing to meet their timing requirements, you can use hard location assignments to optimize placement.

Location assignments are less flexible for the Intel Quartus Prime Fitter than Logic Lock (Standard) assignments. Additionally, if you are familiar with your design, you can enter location constraints in a way that produces better results.

Note: Improving fitting results, especially for larger devices, such as Arria and Stratix series devices, can be difficult. Location assignments do not always improve the performance of the design. In many cases, you cannot improve upon the results from the Fitter by making location assignments.

3.5.8. 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 Intel 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.

If the MTBF of your design is low, refer to the Metastability Optimization section in the Timing Optimization Advisor, which suggests various settings that can help optimize your design in terms of metastability.

This chapter describes how to enable metastability analysis and identify the register synchronization chains in your design, provides details about metastability reports, and provides additional guidelines for managing metastability.

Related Information

- Understanding Metastability in FPGAs
- Managing Metastability with the Intel Quartus Prime Software
  In *Intel Quartus Prime Standard Edition User Guide: Design Recommendations*

3.6. 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.

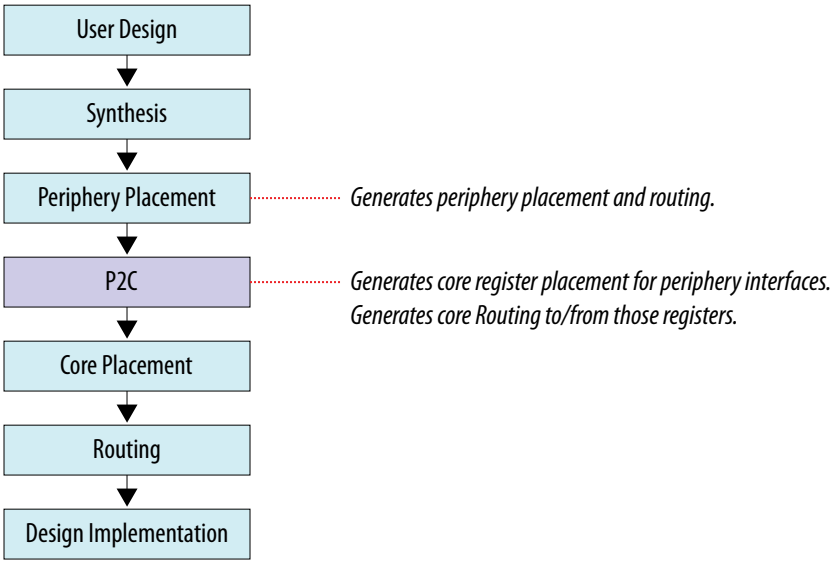
Transfers between external interfaces (for example, high-speed I/O or serial interfaces) and the FPGA often require routing many connections with tight setup and hold timing requirements. When this option is turned on, the Fitter performs P2C

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 33. 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 81

Setting Periphery to Core Optimizations in the Assignment Editor on page 82

Viewing Periphery to Core Optimizations in the Fitter Report on page 82

3.6.1. 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 Intel 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.

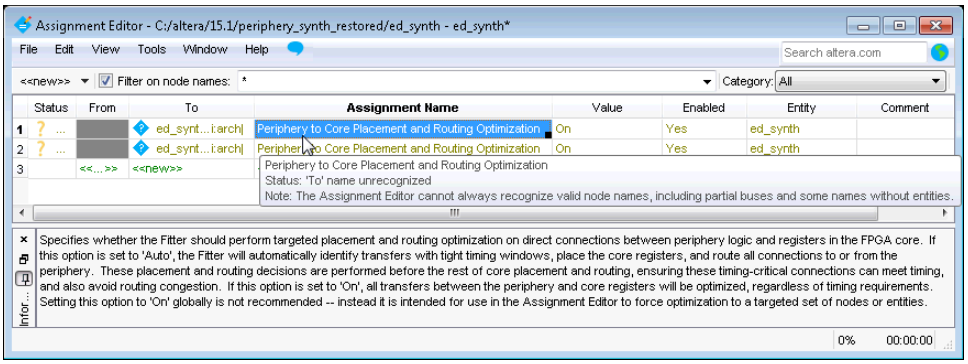
c. Select **Off** to disable periphery to core path optimization in your design.

3.6.2. 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 Intel 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 Intel 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.



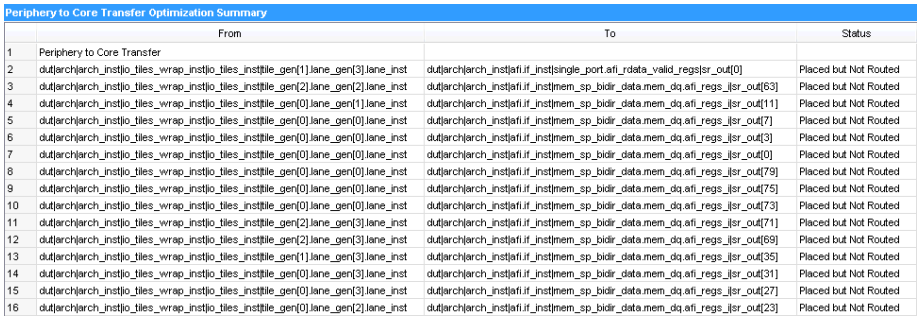
3.6.3. Viewing Periphery to Core Optimizations in the Fitter Report

The Intel Quartus Prime software generates a periphery to core placement and routing optimization summary in the **Fitter (Place & Route)** report after compilation.

1. Compile your Intel 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 11. 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_l|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_l|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_l|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_l|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_l|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_l|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_l|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_l|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_l|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_l|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_l|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_l|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_l|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_l|sr_out[23]	Placed but Not Routed

3.7. 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 Intel 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

- Intel Quartus Prime Standard Edition Settings File Reference Manual
  For information about all settings and constraints in the Intel Quartus Prime software.
- Tcl Scripting
- Command Line Scripting

3.7.1. Initial Compilation Settings

Use the Intel 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 12. 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 13. 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

3.7.2. I/O Timing Optimization Techniques

The table lists the `.qsf` file variable name and applicable values for the I/O timing optimization settings.

Table 14. 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

3.7.3. 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 15. 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
Perform Physical Synthesis for Combinational Logic (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_COMBO_LOGIC	ON, OFF	Global, Instance
Perform Register Duplication (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION	ON, OFF	Global, Instance
Perform Register Retiming (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_REGISTER_RETIMING	ON, OFF	Global, Instance
Perform Automatic Asynchronous Signal Pipelining (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING	ON, OFF	Global, Instance
Physical Synthesis Effort (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_EFFORT	NORMAL, EXTRA, FAST	Global
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

Related Information

Register-to-Register Timing Optimization Techniques on page 72

3.8. Timing Closure and Optimization Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
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	• Initial release in Intel Quartus Prime Standard Edition User Guide. • Removed duplicated topic: *Resource Utilization Optimization Techniques*. The topic is now in the *Area Optimization* chapter.
2017.11.06	17.1.0	• Moved Topic: Design Evaluation for Timing Closure after Initial Compilation: Optional Fitter Settings. • 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* .
2016.05.02	16.0.0	• Stated limitations 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.

continued...

Document Version	Intel Quartus Prime Version	Changes
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.
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

continued...

Document Version	Intel Quartus Prime Version	Changes
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

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

4. Area Optimization

This chapter describes techniques to reduce resource usage when designing for Intel devices.

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, digital signal processing (DSP) blocks, and phase-locked loops (PLLs).

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

In the **Fitter** section of the compilation report, reports under **Resource Section** provide detailed resource information.

The **Fitter Resource Usage Summary** report breaks down the logic utilization information and provides additional resource information, including the number of bits in each type of memory block. This panel also contains a summary of the usage of global clocks, PLLs, DSP blocks, and other device-specific resources.

Related Information

Fitter Resources Reports

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

4.1.3. Analysis and Synthesis Reports

For designs synthesized with the Intel 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

Synthesis Optimization Results Reports

4.1.4. Compilation Messages

If the reports show routing resource usage lower than 100% but the design does not fit, either routing 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.

If the Fitter finishes unsuccessfully and runs much faster than on similar designs, a resource might be over-utilized or there might be an illegal assignment.

If the Intel Quartus Prime software takes too long to run when compared to similar designs possibly the Compiler is not able to find valid placement or route. In the Compilation Report, look for errors and warnings that indicate these types of problems.

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 Messages

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 43

4.2.1. Using the Resource Optimization Advisor

The Resource Optimization Advisor provides guidance in determining settings that optimize resource usage. To run the Resource Optimization Advisor click **Tools ➤ Advisors ➤ Resource Optimization Advisor**.

The Resource Optimization Advisor provides step-by-step advice about how to optimize resource usage (logic element, memory block, DSP block, I/O, and routing) of your design. Some of the recommendations in these categories might conflict with each other. Intel recommends evaluating the options and choosing the settings that best suit your requirements.

Related Information

Resource Optimization Advisor Command Tools Menu

4.2.2. 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*.

4.2.3. I/O Pin Utilization or Placement

Resolve I/O resource problems with these guidelines.

4.2.3.1. Guideline: Use I/O Assignment Analysis

To help with pin placement, click **Processing ➤ Start ➤ Start I/O Assignment Analysis**. The **Start I/O Assignment Analysis** command allows you to check your I/O assignments early in the design process. You can use this command to check the legality of pin assignments before, during, or after compilation of your design. If design files are available, you can use this command to accomplish more thorough legality checks on your design's I/O pins and surrounding logic. These checks include proper reference voltage pin usage, valid pin location assignments, and acceptable mixed I/O standards.

Common issues with I/O placement relate to the fact that differential standards have specific pin pairings and certain I/O standards might be supported only on certain I/O banks.

If your compilation or I/O assignment analysis results in specific errors relating to I/O pins, follow the recommendations in the error message. Right-click the message in the **Messages** window and click **Help** to open the Intel Quartus Prime Help topic for this message.

4.2.3.2. 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 Intel 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

Managing Device I/O Pins
>In *Intel Quartus Prime Standard Edition User Guide: Design Constraints*

4.2.4. 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.

4.2.4.1. Guideline: Optimize Source Code

If your design does not fit because of logic utilization, then evaluate and modify the design at the source. 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 Intel 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
- Recommended HDL Coding Styles
  In *Intel Quartus Prime Standard Edition User Guide: Design Recommendations*

4.2.4.2. 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 Intel Quartus Prime integrated synthesis, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** and select **Balanced** or **Area** for the **Optimization Technique**.
- 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.
- You can also turn on the **Speed Optimization Technique for Clock Domains** logic option to optimize for speed all combinational logic in or between the specified clock domains.
- 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 Intel 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 Intel Quartus Prime software provides additional attributes and options that can help improve the quality of the synthesis results.

Related Information

Intel Quartus Prime Integrated Synthesis

4.2.4.3. 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
- Recommended HDL Coding Styles
  For design guidelines to achieve optimal resource utilization for multiplexer designs

4.2.4.4. 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

4.2.4.5. Guideline: Use Register Packing

The **Auto Packed Registers** option implements the functions of two cells into one logic cell by combining the register of one cell in which only the register is used with the LUT of another cell in which only the LUT is used.

Related Information

Auto Packed Registers logic option
For more information about the Auto Packed Registers logic option

4.2.4.6. 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 (Standard) 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 (Standard) 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 (Standard) 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 (Standard) 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 (Standard) assignments in the Chip Planner, in the Logic Lock (Standard) 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 102

4.2.4.7. 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 Intel 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.

If you are using Intel Quartus Prime incremental compilation, you cannot flatten your design across design partitions. Incremental compilation always preserves the hierarchical boundaries between design partitions, and the synthesis does not flatten it across partitions. Follow Intel's recommendations for design partitioning, such as registering partition boundaries to reduce the effect of cross-boundary optimizations.

4.2.4.8. 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 Intel 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.

Consider the Resource Utilization by Entity report in the report file and determine whether there is an unusually high register count in any of the modules. Some coding styles prevent the Intel 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.

Related Information

- Inferring Shift Registers in HDL Code
- Fitter Resource Utilization by Entity Report

4.2.4.9. 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 Intel 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

4.2.4.10. Guideline: Retarget or Balance DSP Blocks

A design might not fit because it requires too many DSP blocks. 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 Intel 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 Intel Quartus Prime integrated 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 Intel 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 Intel 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.

4.2.4.11. 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.

4.2.5. Routing

Resolve routing resource problems with these guidelines.

4.2.5.1. 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

4.2.5.2. 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

4.2.5.3. 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.

4.2.5.4. 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 (Standard) 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 (Standard) 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 (Standard) 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 (Standard) 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 (Standard) assignments in the Chip Planner, in the Logic Lock (Standard) 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 102

4.2.5.5. Guideline: Optimize Synthesis for Area, Not Speed

In some cases, resynthesizing the design to improve the area utilization can also improve the routability of the design. First, ensure that you have set your device and timing constraints correctly in your synthesis tool. Ensure that you do not over constrain the timing requirements for the design, particularly when the area utilization

of the design is a concern. Synthesis tools generally 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 Intel Quartus Prime integrated synthesis, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)** and select **Balanced** or **Area** for the **Optimization Technique**.
- If you want to reduce area for specific modules in your design using the **Area** or **Speed** setting while leaving the default **Optimization Technique** setting at **Balanced**, use the Assignment Editor.
- You can also use the **Speed Optimization Technique for Clock Domains** logic option to specify that all combinational logic in or between the specified clock domain(s) is optimized for speed.
- In some synthesis tools, not specifying an f_{MAX} requirement can result in lower resource utilization.

Optimizing for area or speed can affect the register-to-register timing performance.

Note: In the Intel 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 Intel Quartus Prime software provides additional attributes and options that can help improve the quality of your synthesis results.

Related Information

Intel Quartus Prime Integrated Synthesis

4.2.5.6. 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, 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.

4.2.5.7. 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 Intel 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

- Tcl Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*
- Command Line Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*

4.3.1. Initial Compilation Settings

Use the Intel 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 16. 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 17. Resource Utilization Optimization Settings

Setting Name	.qsf File Variable Name	Values	Type
Auto Packed Registers[1]	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
Perform Physical Synthesis for Combinational Logic for Reducing Area (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_COMBO_LOGIC_FOR_AREA	ON, OFF	Global, Instance
Perform Physical Synthesis for Mapping Logic to Memory (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_MAP_LOGIC_TO_MEMORY_FOR AREA	ON, OFF	Global, Instance
Optimization Technique	<device family name>_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
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

(1) Allowed values for this setting depend on the device family that you select.

4.4. Area Optimization Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • Divided topic: *Resource Utilization* into topics: *Resource Utilization Information*, *Flow Summary Report*, *Fitter Reports*, *Analysis and Synthesis Reports*, and *Compilation Messages*.
2018.07.03	18.0.0	Fixed typo and added links in topic *Guideline: Retarget Memory Blocks*.
2017.05.08	17.0.0	• Revised topics: *Resolving Resource Utilization Issues*, *Guideline: Optimize Synthesis for Area, Not Speed*
2016.05.02	16.0.0	• Stated limitations about deprecated physical synthesis options.
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 Synthesis Optimizations to Compiler Settings.
June 2014	14.0.0	• Removed Cyclone III and Stratix III devices references. • Removed Macrocell-Based CPLDs related information. • Updated template.
May 2013	13.0.0	Initial release.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

5. Analyzing and Optimizing the Design Floorplan

As FPGA designs grow larger in density, the ability to analyze the design for performance, routing congestion, and logic placement is critical to meet the design requirements. This chapter discusses how the Chip Planner and Logic Lock (Standard) regions help you improve your design's floorplan.

Design floorplan analysis helps to close timing, and ensures optimal performance in highly complex designs. With analysis capability, the Intel Quartus Prime Chip Planner helps you close timing quickly on your designs. You can use the Chip Planner together with Logic Lock (Standard) regions to compile your designs hierarchically and assist with floorplanning. Additionally, use partitions to preserve placement and routing results from individual compilation runs.

You can perform design analysis, as well as create and optimize the design floorplan with the Chip Planner. To make I/O assignments, use the Pin Planner.

Related Information

Managing Device I/O Pins

5.1. Design Floorplan Analysis in the Chip Planner

The Chip Planner simplifies floorplan analysis by providing visual display of chip resources. With the Chip Planner, you can view post-compilation placement, connections, and routing paths.

The Chip Planner allows you to:

- Make assignment changes, such as creating and deleting resource assignments.
- Perform post-compilation changes such as creating, moving, and deleting logic cells and I/O atoms.
- Perform power and design analyses.
- Implement ECOs.
- Change connections between resources and make post-compilation changes to the properties of logic cells, I/O elements, PLLs, RAMs, and digital signal processing (DSP) blocks.

The Chip Planner showcases:

- Logic Lock (Standard) regions
- Relative resource usage
- Detailed routing information
- Fan-in and fan-out connections between nodes
- Timing paths between registers
- Delay estimates for paths
- Routing congestion information

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

5.1.1. Starting the Chip Planner

To start the Chip Planner, select **Tools ➤ Chip Planner**. You can also start the Chip Planner by the following methods:

- Click the Chip Planner icon on the Intel Quartus Prime software toolbar.
- In the following tools, right-click any chip resource and select **Locate ➤ Locate in Chip Planner**:
  - Design Partition Planner
  - Compilation Report
  - **Logic Lock (Standard) Regions Window**
  - Technology Map Viewer
  - **Project Navigator** window
  - RTL source code
  - Node Finder
  - Simulation Report
  - RTL Viewer
  - Report Timing panel of the Timing Analyzer

5.1.2. Chip Planner GUI Components

5.1.2.1. Chip Planner Toolbar

The Chip Planner toolbar provides powerful tools for visual design analysis. You can access Chip Planner commands either from the **View** or the **Shortcut** menu, or by clicking the icons in the toolbars.

5.1.2.2. Layers Settings and Editing Modes

The Chip Planner allows you to control the display of resources. To determine the operations that you can perform, use the **Editing Mode** .

Layers Settings Pane

With the **Layers Settings** pane, you can manage the graphic elements that the Chip Planner displays.

You open the **Layers Settings** pane by clicking **View ➤ Layers Settings**. The **Layers Settings** pane offers layer presets, which group resources that are often used together. The **Basic**, **Detailed**, and **Floorplan Editing** default presets are useful for general assignment-related activities. You can also create custom presets tailored to your needs. The **Design Partition Planner** preset is optimized for specific activities.

Editing Mode

The Chip Planner's **Editing Mode** determines the operations that you can perform. The **Assignment** editing mode allows you to make assignment changes that are applied by the Fitter during the next place and route operation. The **ECO** editing mode allows you to make post-compilation changes, commonly referred to as engineering change orders (ECOs).

Select the editing mode appropriate for the work that you want to perform, and a preset that displays the resources that you want to view, in a level of detail appropriate for your design.

Related Information

- Viewing Architecture-Specific Design Information on page 105
- Layers Settings Dialog Box

5.1.2.3. Locate History Window

As you optimize your design floorplan, you might have to locate a path or node in the Chip Planner more than once. The **Locate History** window lists all the nodes and paths you have displayed using a **Locate in Chip Planner** command, providing easy access to the nodes and paths of interest to you.

If you locate a required path from the **Timing Analyzer Report Timing** pane, the **Locate History** window displays the required clock path. If you locate an arrival path from the **Timing Analyzer Report Timing** pane, the **Locate History** window displays the path from the arrival clock to the arrival data. Double-clicking a node or path in the **Locate History** window displays the selected node or path in the Chip Planner.

5.1.2.4. Chip Planner Floorplan Views

The Chip Planner uses a hierarchical zoom viewer that shows various abstraction levels of the targeted Intel device. As you zoom in, the level of abstraction decreases, revealing more details about your design.

Bird's Eye View

The Bird's Eye View displays a high-level picture of resource usage for the entire chip and provides a fast and efficient way to navigate between areas of interest in the Chip Planner.

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, and you want to quickly navigate between resource elements without losing your frame of reference.

Properties Window

The **Properties** window displays detailed properties of the objects (such as atoms, paths, Logic Lock (Standard) regions, or routing elements) currently selected in the Chip Planner. To display the **Properties** window, right-click the object and select **View ➤ Properties**.

Related Information

Bird's Eye View Window

5.1.3. Viewing Architecture-Specific Design Information

The Chip Planner allows you to view architecture-specific information related to your design. By enabling the options in the **Layers Settings** pane, 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.

In addition, you can modify the following device properties with the Chip Planner:

- LEs and ALMs
- I/O cells
- PLLs
- Registers in RAM and DSP blocks
- Connections between elements
- Placement of elements

  For more information about LEs, ALMs, and other resources of an FPGA device, refer to the relevant device handbook.

Related Information

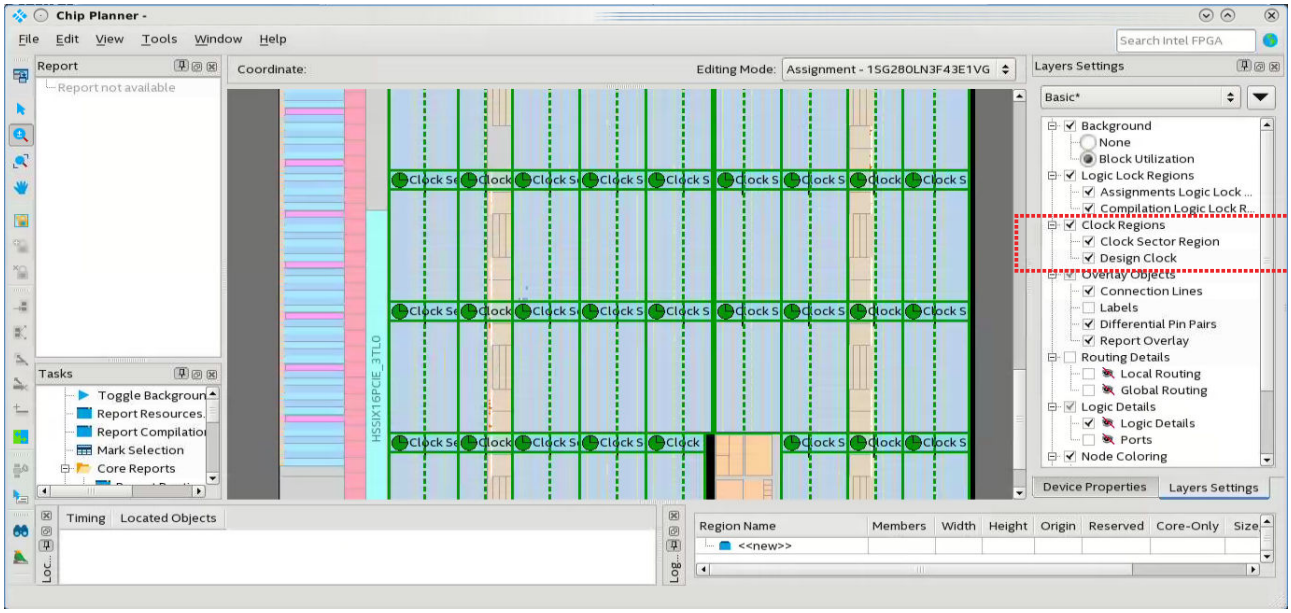
- Layers Settings and Editing Modes on page 103
- Layers Settings Dialog Box

5.1.4. Viewing Available Clock Networks in the Device

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.

This global clock display feature is available for Arria V, Intel Arria 10, Cyclone V, Stratix IV, and Stratix V device families.

Figure 34. Clock Regions



- 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.
- 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.
- 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 72
- Layers Settings and Editing Modes on page 103
- Report Spine Clock Utilization dialog box (Chip Planner)

5.1.5. Viewing Routing Congestion

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 111

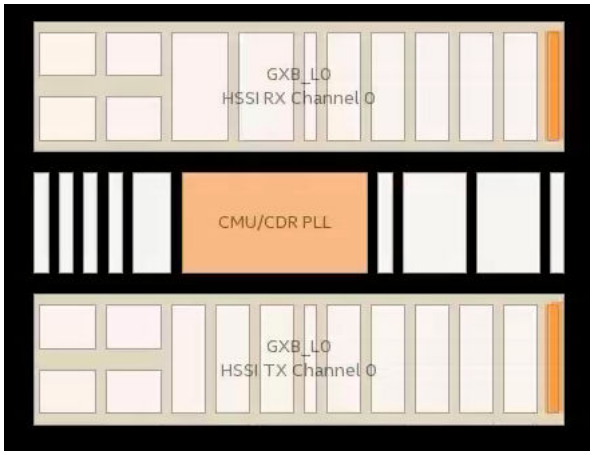
5.1.6. Viewing I/O Banks

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.

5.1.7. Viewing High-Speed Serial Interfaces (HSSI)

For selected device families, tThe 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 35. Intel Arria 10 HSSI Channel Blocks



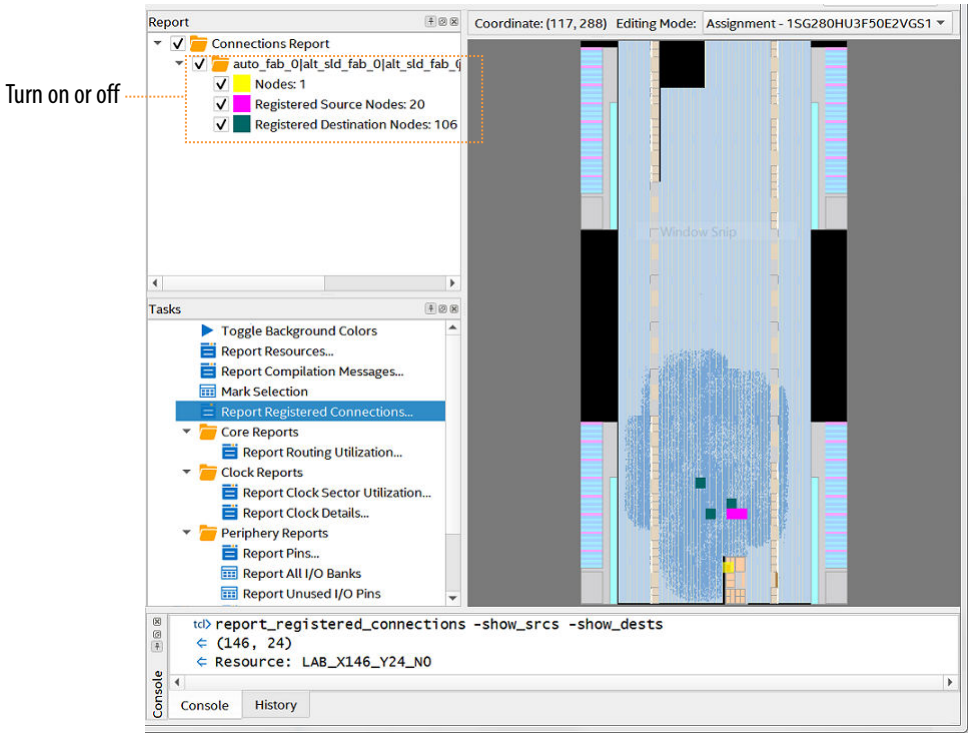
5.1.8. Viewing the Source and Destination of Placed Nodes

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 36. Report Registered Connections



Related Information

- Viewing Fan-In and Fan-Out Connections of Placed Resources on page 109
- Expand Connections Command (View Menu)

5.1.9. Viewing Fan-In and Fan-Out Connections of Placed Resources

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** icon or the **Generate Fan-Out Connections** icon.
2. To remove other connections that appear on the Chip Planner view, click the **Clear Unselected Connections** icon.

You can also perform this actions from the Chip Planner **View** menu.

Related Information

- Viewing the Source and Destination of Placed Nodes on page 108
- Expand Connections Command (View Menu)

5.1.10. Generating Immediate Fan-In and Fan-Out Connections

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** icon in the Chip Planner toolbar.

5.1.11. 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.

5.1.11.1. Analyzing Connections for a Path

To determine the elements forming a selected path or connection in the Chip Planner, click the **Expand Connections** icon in the Chip Planner toolbar.

Related Information

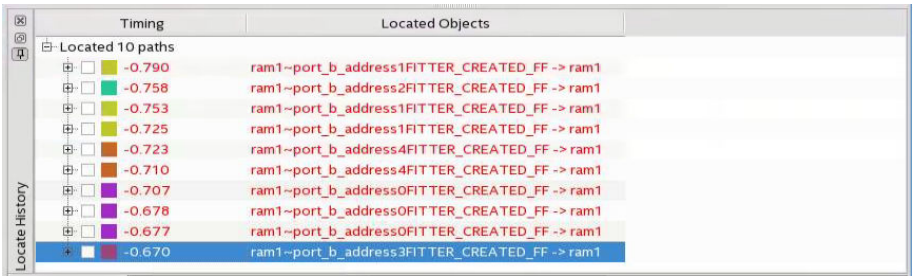
Expand Connections Command (View Menu)

5.1.11.2. 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 37. Path List in the Locate History Window



Related Information

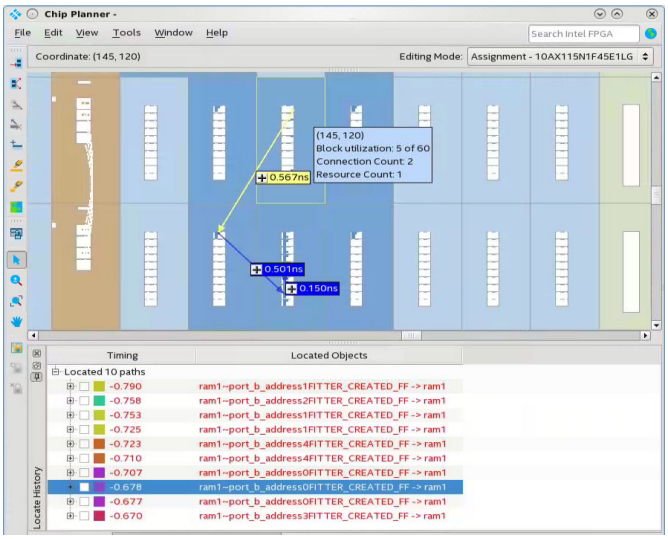
Displaying Path Reports with the Timing Analyzer on page 60

5.1.11.3. 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 icon 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.

For example, you can view the delay between two logic resources or between a logic resource and a routing resource.

Figure 38. Show Delays Associated in a Timing Analyzer Path



5.1.11.4. 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.

Figure 39. Show Physical Routing

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**.

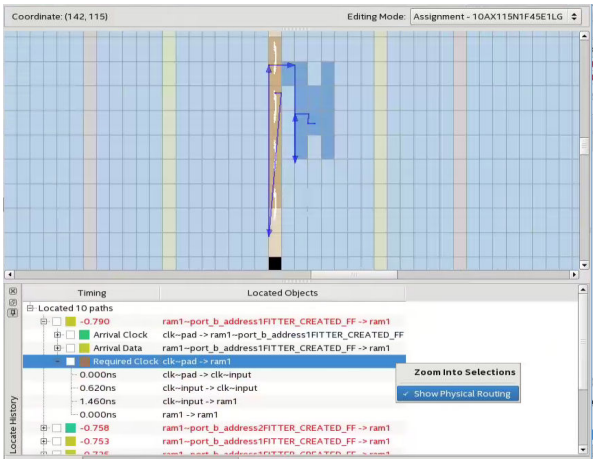
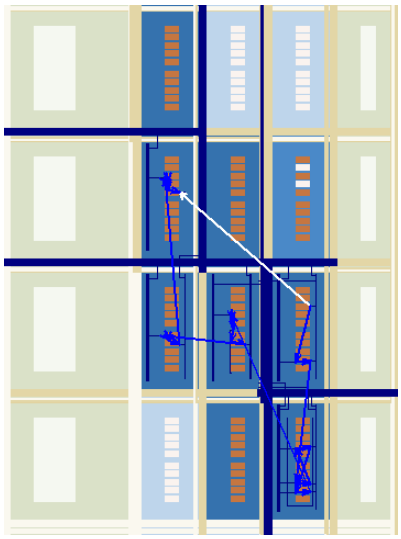


Figure 40. Highlight Routing

To see the rows and columns where the Fitter routed the path, right-click a path and select **Highlight Routing**.



Related Information

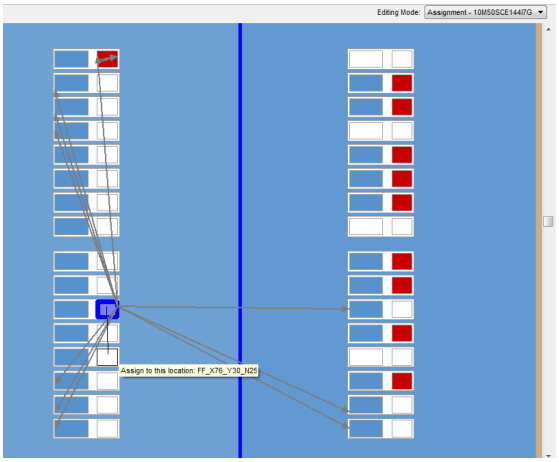
Viewing Routing Congestion on page 106

5.1.12. Viewing Assignments in the Chip Planner

You can view location assignments in the Chip Planner by using the Assignment editing mode and the **Floorplan Editing** preset in the **Layers Settings** pane.

The Chip Planner displays assigned resources in a predefined color (gray, by default).

Figure 41. Viewing Assignments in the Chip Planner



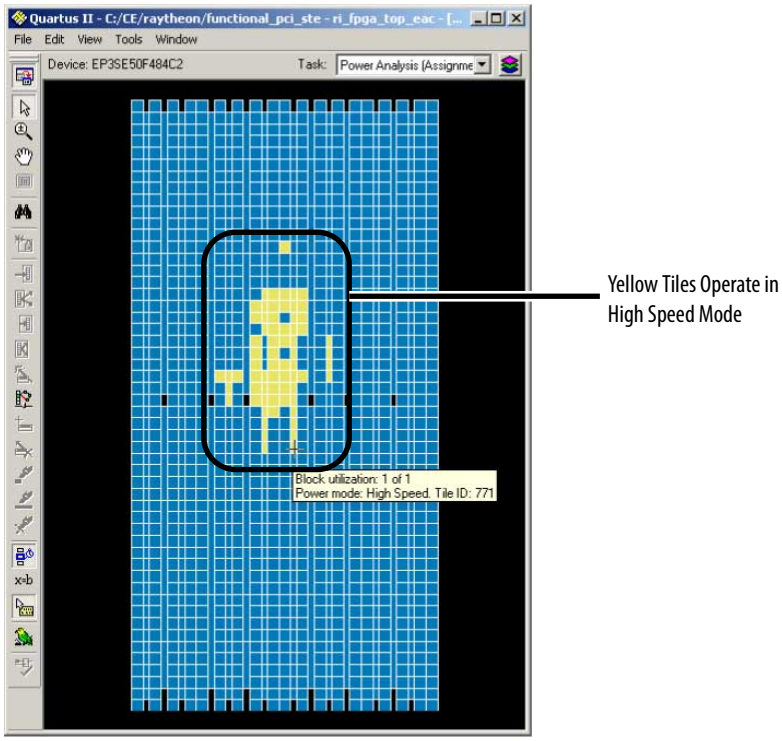
To create or move an assignment, or to make node and pin location assignments to Logic Lock (Standard) regions, drag the selected resource to a new location. The Fitter applies the assignments that you create during the next place-and-route operation.

5.1.13. 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 Intel Quartus Prime software. These ALMs are grouped together to form larger blocks, called "tiles".

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.

Figure 42. Viewing High-Speed and Low Power Tiles in a Stratix Device



Related Information

AN 514: Power Optimization in Stratix IV FPGAs

5.1.14. 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.

5.2. Logic Lock (Standard) Regions

Logic Lock (Standard) regions are floorplan location constraints. When you assign instances or nodes to a Logic Lock (Standard) region, you direct the Fitter to place those instances or nodes within the region. A floorplan can contain multiple Logic Lock (Standard) regions.

You can use the Design Partition Planner in conjunction with Logic Lock (Standard) regions to create a floorplan for your design.

Related Information

Creating Logic Lock (Standard) Regions on page 115

5.2.1. Attributes of a Logic Lock (Standard) Region

The following table lists the attributes of a Logic Lock (Standard) region. In the Intel Quartus Prime software, the Logic Lock (Standard) Regions window displays the attributes of all the Logic Lock (Standard) regions in the design.

Table 18. Attributes of Logic Lock (Standard) Regions

Name	Value	Behavior
Size	Auto | Fixed	**Auto** allows the Intel Quartus Prime software to determine the appropriate size of a region given its contents. **Fixed** regions have a shape and size that you define.
Width	Number of columns	Specifies the width of the Logic Lock (Standard) region.
Height	Number of rows	Specifies the height of the Logic Lock (Standard) region.
State	Floating | Locked	**Floating** allows the Intel Quartus Prime software to determine the location of the region on the device. Floating regions appear with a dashed boundary in the floorplan. **Locked** allows you to specify the location of the region. Locked regions appear with a solid boundary in the floorplan. A **Locked** region must have a **Fixed** size.
Origin	Any Floorplan Location | Undetermined	Specifies the location of the Logic Lock (Standard) region on the floorplan. The origin is at the lower left corner of the Logic Lock (Standard) region.
Reserved	Off | On	Prevents the Fitter from placing other logic in the region.

Related Information

Logic Lock (Standard) Regions Window on page 124

5.2.2. Creating Logic Lock (Standard) Regions

You can define a Logic Lock (Standard) region by its height, width, and location; Alternatively, you can specify the size or location of a region, or both, or the Intel Quartus Prime software can generate these properties automatically. The Intel Quartus Prime software bases the size and location of a region on the contents of the region and the timing requirements of the module.

The Intel Quartus Prime software displays Logic Lock (Standard) regions with colors indicating the percentage of resources available in the region. An orange Logic Lock (Standard) region indicates a nearly full Logic Lock (Standard) region.

Intel Quartus Prime software cannot automatically define the size of a region if the location is **Locked**. Therefore, if you want to specify the exact location of the region, you must also specify the size.

5.2.2.1. Creating Logic Lock (Standard) Regions with the Chip Planner

1. Click **View ➤ Logic Lock (Standard) Regions ➤ Create Logic Lock (Standard) Region**
2. Click and drag on the Chip Planner floorplan to create a region of your preferred location and size

After you create the region, you can define the region shape and then assign a single entity to the region. The order that you assign the entity or define the shape does not matter.

5.2.2.2. Creating Logic Lock (Standard) Regions with the Project Navigator

1. Perform either a full compilation or analysis and elaboration on the design.
2. If the Project Navigator is not already open, click **View ➤ Utility Windows ➤ Project Navigator**. The Project Navigator displays the hierarchy of the design.
3. With the design hierarchy fully expanded, right-click any design entity, and click **Create New Logic Lock (Standard) Region**.
4. Assign the entity to the new region.

The new region has the same name as the entity.

5.2.2.3. Creating Logic Lock (Standard) Regions with the Logic Lock (Standard) Regions Window

1. Click **Assignments ➤ Logic Lock (Standard) Regions Window**.
2. In the **Logic Lock (Standard) Regions** window, click **<<new>>**.

After you create the region, you can define the region shape and then assign a single entity to the region. The order that you assign the entity or define the shape does not matter.

Related Information

Logic Lock (Standard) Regions Window on page 124

5.2.2.4. Defining Routing Regions

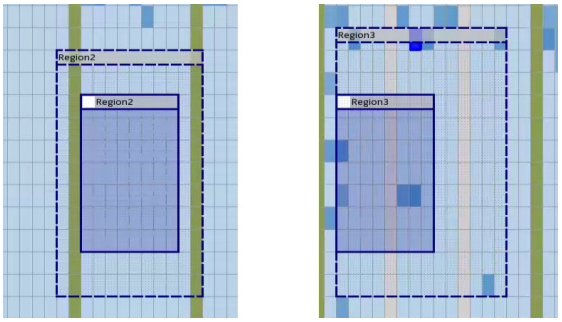
A routing region is an element of a Logic Lock region that specifies the routing area. A routing region must encompass the existing Logic Lock placement region. Routing regions cannot be set as reserved. To define the routing region, double-click the **Routing Region** cell in the **Logic Lock (Standard) Regions** window, and select an option from the drop-down menu.

Valid routing region options are:

Table 19. Routing Region Options

Option	Description
Unconstrained (default)	Allows the fitter to use any available routes on the device.
Whole Chip	Same as Unconstrained, but writes the constraint in the Intel Quartus Prime settings file (`.qsf`).
Fixed with Expansion	Follows the outline of the placement region. The routing region scales by a number of rows/cols larger than the placement region.
Custom	Allows you to make 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.

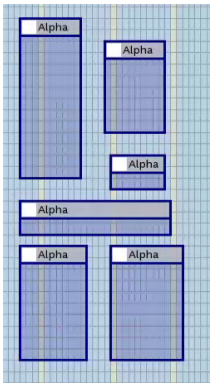
Figure 43. Routing Regions



5.2.2.5. Noncontiguous Logic Lock (Standard) Regions

You can create disjointed regions by using the Logic Lock (Standard) region manipulation tools. Noncontiguous regions act as a single Logic Lock (Standard) region for all Logic Lock (Standard) region attributes.

Figure 44. Noncontiguous Logic Lock (Standard) Region



Related Information

Merging Logic Lock (Standard) Regions on page 118

5.2.2.6. Considerations on Using Auto Sized Regions

If you use **Auto** Sized Logic Lock (Standard) regions, take into account:

- **Auto/Floating** regions cannot be reserved.
- Verify that your Logic Lock (Standard) 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** Sized Logic Lock (Standard) 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 (Standard) 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 (Standard) region.
  - If a partition is preserved at placement, routed, or final; a Logic Lock (Standard) region is not an effective placement boundary, because the location of the partition's logic is fixed.
  - However, if the Logic Lock (Standard) 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 (Standard) region meets your specification, you can:
  - Convert the Logic Lock (Standard) region to **Fixed** Size.
  - Leave the Logic Lock (Standard) region with **Auto** Sized attribute and use the region as a "keep together" type of function.
  - If the Logic Lock (Standard) region is also a partition, you can preserve the place and route through the partition and remove the Logic Lock (Standard) region entirely.

5.2.3. Customizing the Shape of Logic Lock Regions

To create custom shaped Logic Lock regions, you can perform logic operations. Non-rectangular Logic Lock (Standard) 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 (Standard) shapes for 17.1.

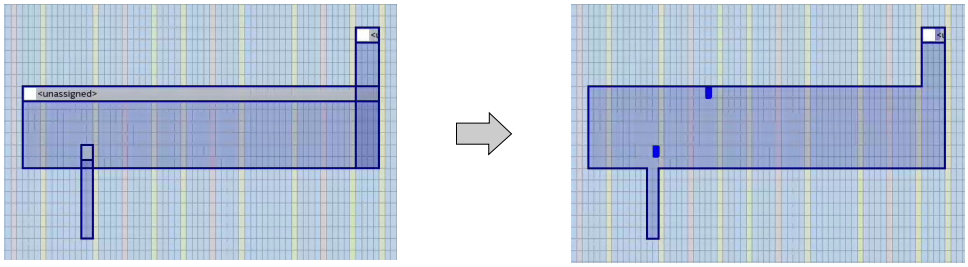
5.2.3.1. Merging Logic Lock (Standard) Regions

To merge two or more Logic Lock (Standard) 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 (Standard) regions and select **Logic Lock (Standard) Regions ➤ Merge Logic Lock (Standard) Region**. The individual regions that you select merge to create a single new region.

Note: By default, the new Logic Lock (Standard) region has the same name as the component region containing the greatest number of resources; however, you can rename the new region. In the **Logic Lock (Standard) Regions Window**, the new region is shown as having a **Custom Shape**.

Figure 45. Using the Merge Logic Lock (Standard) Region command



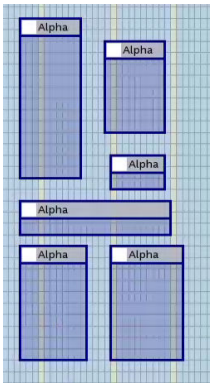
Related Information

Creating Logic Lock (Standard) Regions on page 115

5.2.3.2. Noncontiguous Logic Lock (Standard) Regions

You can create disjointed regions by using the Logic Lock (Standard) region manipulation tools. Noncontiguous regions act as a single Logic Lock (Standard) region for all Logic Lock (Standard) region attributes.

Figure 46. Noncontiguous Logic Lock (Standard) Region



Related Information

Merging Logic Lock (Standard) Regions on page 118

5.2.4. Placing Logic Lock (Standard) Regions

A fixed region must contain all resources required by the design block assigned to the region. Although the Intel Quartus Prime software can automatically place and size Logic Lock (Standard) regions to meet resource and timing requirements, you can manually place and size regions to meet your design requirements.

If you manually place or size a Logic Lock (Standard) region:

- Logic Lock (Standard) regions with pin assignments must be placed on the periphery of the device, adjacent to the pins. You must also include the I/O block within the Logic Lock (Standard) Region.
- Floating Logic Lock (Standard) regions can overlap with their ancestors or descendants, but not with other floating Logic Lock (Standard) regions.

5.2.5. Placing Device Resources into Logic Lock (Standard) Regions

You can assign an entity in the design to only one Logic Lock (Standard) 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 (Standard) region boundary includes part of a device resource, the Intel Quartus Prime software allocates the entire resource to that Logic Lock (Standard) region. When the Intel Quartus Prime software places a floating auto-sized region, it places the region in an area that meets the requirements of the contents of the Logic Lock (Standard) region.

To add an instance using the **Logic Lock Region** window, right-click the region and select **Logic Lock Properties ➤ Add**. Alternatively, in the Intel Quartus Prime software you can drag entities from the Hierarchy viewer into a Logic Lock (Standard) region's name field in the Logic Lock (Standard) Regions Window.

5.2.5.1. Empty Logic Lock Regions

Intel Quartus Prime allows you to have Logic Lock regions with no members. Empty regions are a tool to manage space in the FPGA for future logic. This technique only works when you set the regions to **Reserved**

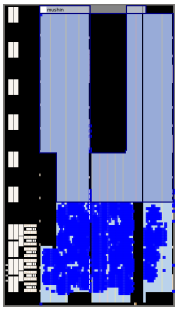
Some reasons to use empty Logic Lock regions are:

- Preliminary floorplanning.
- Complex incremental builds.
- Team based design and interconnect logic.
- Confining logic placements.

Since Logic Lock regions do not reserve any routing resources, the Fitter may use the 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 populated Logic Lock Region.

Figure 47. Logic Placed Outside of an Empty region



The figure shows an empty Logic Lock region and the logic around it. However, some IOs, HSSIO, and PLLs are in the empty region. This placement happens because the output port connects to the IO, and the IO is always part of the root_partition (top-level partition).

5.2.5.2. Pin Assignment

A Logic Lock (Standard) region incorporates all device resources within its boundaries, including memory and pins. The Intel Quartus Prime Standard Edition software automatically includes pins when you assign an instance to a region. You can manually exclude pins with a **Core Only** assignment.

Note: Pin assignments to Logic Lock (Standard) regions are effective only in fixed and locked regions. Pin assignments to floating regions do not influence the placement of the region.

You can assign an entity in the design to only one Logic Lock (Standard) region, but the entity can inherit regions by hierarchy. This hierarchy allows a reserved region to have a subregion without reserving the resources in the subregion.

When the Intel Quartus Prime software places a floating auto-sized region, it places the region in an area that meets the requirements of the contents of the Logic Lock (Standard) region.

5.2.5.3. Reserved Logic Lock (Standard) Regions

The **Reserved** attribute instructs the Fitter to only place the entities and nodes that you specifically assigned to the Logic Lock (Standard) region in the Logic Lock (Standard) region.

The Intel Quartus Prime software honors all entity and node assignments to Logic Lock (Standard) regions. Occasionally entities and nodes do not occupy an entire region, which leaves some of the region's resources unoccupied.

To increase the region's resource utilization and performance, Intel Quartus Prime software by default fills the unoccupied resources with other nodes and entities that have not been assigned to another region. To prevent this behavior, turn on **Reserved** on the**Logic Lock (Standard) Region Properties ➤ General** tab.

5.2.5.4. Excluded Resources

The Excluded Resources feature allows you to easily exclude specific device resources such as DSP blocks or M4K memory blocks from a Logic Lock (Standard) region.

For example, you can assign a specific entity to a Logic Lock (Standard) region but allow the DSP blocks of that entity to be placed anywhere on the device. Use the Excluded Resources feature on a per-Logic Lock (Standard) region member basis.

To exclude certain device resources from an entity, in the **Logic Lock (Standard) Region Properties** dialog box, highlight the entity in the **Design Element** column, and click **Edit**. In the **Edit Node** dialog box, under **Excluded Element Types**, click the **Browse** button. In the **Excluded Resources Element Types** dialog box, you can select the device resources you want to exclude from the entity. When you have selected the resources to exclude, the **Excluded Resources** column is updated in the **Logic Lock (Standard) Region Properties** dialog box to reflect the excluded resources.

Note: The Excluded Resources feature prevents certain resource types from being included in a region, but it does not prevent the resources from being placed inside the region unless you set the region's **Reserved** property to **On**. To indicate to the Fitter that certain resources are not required inside a Logic Lock (Standard) region, define a resource filter.

5.2.5.5. Logic Lock (Standard) Assignment Precedence

You can encounter conflicts during the assignment of entities and nodes to Logic Lock (Standard) regions. For example, an entire top-level entity might be assigned to one region and a node within this top-level entity assigned to another region.

To resolve conflicting assignments, the Intel Quartus Prime software maintains an order of precedence for Logic Lock (Standard) assignments. The following order of precedence, from highest to lowest, applies:

1. Exact node-level assignments
2. Path-based and wildcard assignments
3. Hierarchical assignments

Note: To open the **Priority** dialog box, select **Logic Lock (Standard) Regions Properties ➤ General ➤ Priority**. You can change the priority of path-based and wildcard assignments with the **Up** and **Down** buttons in the **Priority** dialog box. To prioritize assignments between regions, you must select multiple Logic Lock (Standard) regions and then open the **Priority** dialog box from the **Logic Lock (Standard) Regions Properties** dialog box.

5.2.5.6. Virtual Pins

A virtual pin is an I/O element that the Compiler temporarily maps to a logic element, and not to a pin during compilation. The software implements virtual pins as LUTs. To assign a Virtual Pin, use the Assignment Editor. You can create virtual pins by assigning the **Virtual Pin** logic option to an I/O element.

When you apply the **Virtual Pin** assignment to an input pin, the pin no longer appears as an FPGA pin; 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.

Note: The **Virtual Pin** logic option must be assigned to an input or output pin. If you assign this option to a bidirectional pin, tri-state pin, or registered I/O element, Analysis & 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.

In the top-level design, you connect these virtual pins to an internal node of another module. By making assignments to virtual pins, you can place those pins in the same location or region on the device as that of the corresponding internal nodes in the top-level module. You can use the **Virtual Pin** option when compiling a Logic Lock (Standard) module with more pins than the target device allows. 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.

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 128
- Managing Device I/O Pins
- Node Finder Command (View Menu)

5.2.6. Hierarchical (Parent and Child) Logic Lock (Standard) Regions

To further constrain module locations, you can define a hierarchy for a group of regions by declaring parent and child regions.

The Intel Quartus Prime software places a child region completely within the boundaries of its parent region; a child region must be placed entirely within the boundary of its parent. Additionally, parent and child regions allow you to further improve the performance of a module by constraining nodes in the critical path of a module.

To make one Logic Lock (Standard) region a child of another Logic Lock (Standard) region, in the Logic Lock (Standard) Regions window, select the new child region and dragging the new child region into its new parent region.

Note: The Logic Lock (Standard) region hierarchy does not have to be the same as the design hierarchy.

You can create both auto-sized and fixed-sized Logic Lock (Standard) regions within a parent Logic Lock (Standard) region; however, the parent of a fixed-sized child region must also be fixed-sized. The location of a locked parent region is locked relative to the device; the location of a locked child region is locked relative to its parent region. If you change the parent's location, the locked child's origin changes, but maintains the same placement relative to the origin of its parent. The location of a floating child region can float within its parent. Complex region hierarchies might result in some LABs not being used, effectively increasing the resource utilization in the device. Do not create more levels of hierarchy than you need.

5.2.7. Additional Intel Quartus Prime Logic Lock (Standard) Design Features

To complement the **Logic Lock (Standard) Regions Window**, the Intel Quartus Prime software has additional features to help you design with Logic Lock (Standard) regions.

5.2.7.1. Analysis and Synthesis Resource Utilization by Entity

The Compilation Report contains an **Analysis and Synthesis Resource Utilization by Entity** section, which reports resource usage statistics, including entity-level information. You can use this feature to verify that any Logic Lock (Standard) region you manually create contains enough resources to accommodate all the entities you assign to it.

5.2.7.2. Intel Quartus Prime Revisions Feature

When you evaluate different Logic Lock (Standard) regions in your design, you might want to experiment with different configurations to achieve your desired results. The Intel Quartus Prime Revisions feature allows you to organize the same project with different settings until you find an optimum configuration.

To use the Revisions feature, choose **Project ➤ Revisions**. You can create a revision from the current design or any previously created revisions. Each revision can have an associated description. You can use revisions to organize the placement constraints created for your Logic Lock (Standard) regions.

5.2.8. Logic Lock (Standard) Regions Window

The Logic Lock (Standard) Regions Window provides a summary of all Logic Lock (Standard) regions defined in your design. Use the Logic Lock (Standard) Regions Window to create, assign elements, and modify properties of a Logic Lock (Standard) region.

Open the Logic Lock (Standard) Regions Window in the Chip Planner by clicking **View ➤ Logic Lock (Standard) Window**, and in Intel Quartus Prime by clicking **Assignments ➤ Logic Lock (Standard) Window**.

Figure 48. Logic Lock (Standard) Regions Window

The Logic Lock (Standard) Regions Window also has a recommendations toolbar; select a Logic Lock (Standard) region from the drop-down list in the recommendations toolbar to display the relevant suggestions to optimize that Logic Lock (Standard) region.

The Intel Quartus Prime software automatically creates a Logic Lock (Standard) region that encompasses the entire device. This default region is labeled `Root_Region`, and is locked and fixed.

You can customize the Logic Lock (Standard) Regions Window by dragging and dropping the columns to change their order; you can also show and hide optional columns by right-clicking any column heading and then selecting the appropriate columns in the shortcut menu.

Logic Lock (Standard) Regions Properties Dialog Box

Use the **Logic Lock (Standard) Regions Properties** dialog box to view and modify detailed information about your Logic Lock (Standard) region, such as which entities and nodes are assigned to your region, and which resources are required.

To open the **Logic Lock (Standard) Regions Properties** dialog box, right-click the region and select **Logic Lock (Standard) Regions Properties...**.

Related Information

- Attributes of a Logic Lock (Standard) Region on page 115
- Creating Logic Lock (Standard) Regions with the Logic Lock (Standard) Regions Window on page 116
- Logic Lock (Standard) Regions Window

5.3. Using Logic Lock (Standard) Regions in the Chip Planner

You can easily create Logic Lock (Standard) regions in the Chip Planner and assign resources to them.

5.3.1. Viewing Connections Between Logic Lock (Standard) Regions in the Chip Planner

You can view and edit Logic Lock (Standard) regions using the Chip Planner. To view and edit Logic Lock (Standard) regions, use **Floorplan Editing** in the **Layers Settings** window, or any layers setting mode that has the **User-assigned Logic Lock (Standard) regions** setting enabled.

The Chip Planner shows the connections between Logic Lock (Standard) regions. By default, you can view each connection as an individual line. You can choose to display connections between two Logic Lock (Standard) 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 Intel Quartus Prime Help.

5.3.2. Using Logic Lock (Standard) Regions with the Design Partition Planner

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 usually places closely connected entities in the same area of the device; however, you can use Logic Lock (Standard) regions, together with the Design Partition Planner and the Chip Planner, to help ensure that logically connected entities retain optimal placement from one compilation to the next.

You can view the logical connectivity between entities with the Design Partition Planner, and the physical placement of those entities with the Chip Planner. In the Design Partition Planner, you can identify entities that are highly interconnected, and place those entities in a partition. In the Chip Planner, you can create Logic Lock (Standard) regions and assign each partition to a Logic Lock (Standard) region, thereby preserving the placement of the entities.

5.4. Scripting Support

You can run procedures and specify the settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt.

Related Information

- Tcl Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*
- Command Line Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*

5.4.1. Initializing and Uninitializing a Logic Lock (Standard) Region

You must initialize the Logic Lock (Standard) data structures before creating or modifying any Logic Lock (Standard) regions and before executing any of the Tcl commands listed below.

Use the following Tcl command to initialize the Logic Lock (Standard) data structures:

```
initialize_logiclock
```

Use the following Tcl command to uninitialize the Logic Lock (Standard) data structures before closing your project:

```
uninitialize_logiclock
```

5.4.2. Creating or Modifying Logic Lock (Standard) Regions

Use the following Tcl command to create or modify a Logic Lock (Standard) region:

```
set_logiclock -auto_size true -floating true -region <my_region-name>
```

Note: The command in the above example sets the size of the region to auto and the state to floating.

If you specify a region name that does not exist in the design, the command creates the region with the specified properties. If you specify the name of an existing region, the command changes all properties you specify and leaves unspecified properties unchanged.

Related Information

Creating Logic Lock (Standard) Regions on page 115

5.4.3. Obtaining Logic Lock (Standard) Region Properties

Use the following Tcl command to obtain Logic Lock (Standard) region properties. This example returns the height of the region named `my_region`:

```
get_logiclock -region my_region -height
```

5.4.4. Assigning Logic Lock (Standard) Region Content

Use the following Tcl commands to assign or change nodes and entities in a Logic Lock (Standard) region. This example assigns all nodes with names matching `fifo*` to the region named `my_region`.

```
set_logiclock_contents -region my_region -to fifo*
```

You can also make path-based assignments with the following Tcl command:

```
set_logiclock_contents -region my_region -from fifo -to ram*
```

5.4.5. Save a Node-Level Netlist for the Entire Design into a Persistent Source File

Make the following assignments to cause the Intel Quartus Prime Fitter to save a node-level netlist for the entire design into a `.vqm` file:

```
set_global_assignment-name LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT ON
set_global_assignment-name LOGICLOCK_INCREMENTAL_COMPILE_FILE <file name>
```

Any path specified in the file name is relative to the project directory. For example, specifying `atom_netlists/top.vqm` places `top.vqm` in the **atom_netlists** subdirectory of your project directory.

A `.vqm` file is saved in the directory specified at the completion of a full compilation.

Note: The saving of a node-level netlist to a persistent source file is not supported for designs targeting newer devices such as MAX V , Stratix IV, or Stratix V.

5.4.6. Setting Logic Lock (Standard) Assignment Priority

Use the following Tcl code to set the priority for a Logic Lock (Standard) region's members. This example reverses the priorities of the Logic Lock (Standard) region in your design.

```
set reverse [list]
for each member [get_logiclock_member_priority] {
    set reverse [insert $reverse 0 $member]
{
set_logiclock_member_priority $reverse
```

5.4.7. 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

- Virtual Pins on page 122
- Managing Device I/O Pins
- Node Finder Command (View Menu)

5.5. Analyzing and Optimizing the Design Floorplan Revision History

The following revision history applies to this chapter:

Table 20. Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • 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* to *Logic Lock (Standard)*.
2017.05.08	17.0.0	• Chapter reorganization and content update. • Added figures: Clock Regions, Creating a Hole in a LogicLock Region, Noncontiguous LogicLock Region, Routing Regions, Logic Placed Outside of an Empty 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*.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.

continued...

Document Version	Intel Quartus Prime Version	Changes
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

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

6. Netlist Optimizations and Physical Synthesis

The Intel Quartus Prime software offers netlist and physical synthesis optimizations that improve performance of your design. Click to enable physical synthesis options during fitting. This chapter also provides guidelines for applying netlist and physical synthesis options, and for preserving compilation results through back-annotation.

Table 21. Netlist Optimization and Physical Synthesis Options

Options	Location/Description
Enable physical synthesis options.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**. Physical synthesis optimizations apply at different stages of the compilation flow, either during synthesis, fitting, or both.
Enable netlist optimization options.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. Netlist optimizations operate with the atom netlist of your design, which describes a design in terms of specific primitives. An atom netlist file can be an Electronic Design Interchange Format (`.edf`) file or a Verilog Quartus Mapping (`.vqm`) file generated by a third-party synthesis tool. Intel Quartus Prime synthesis generates and internally uses the atom netlist internally

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 (Standard) based optimization flow (for legacy devices), disable physical synthesis options.

6.1. Physical Synthesis Optimizations

The Intel 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.

Some physical synthesis options affect only registered logic, while others affect only combinational logic. Select options based on whether you want to keep the registers intact. For example, if your verification flow involves formal verification, you might want to keep the registers intact.

The following sections describe the physical synthesis optimizations available in the Intel Quartus Prime software, and how they can help improve performance and fitting for the selected device.

Related Information

Compiler Settings Page (Settings Dialog Box)

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

6.1.1. Enabling Physical Synthesis Optimization

Physical synthesis optimization improves circuit performance by performing combinational and sequential optimization and register duplication.

To enable physical synthesis options:

1. Click **Assignments ➤ Settings ➤ Compiler Settings**.
2. To enable physical synthesis, click **Advanced Settings (Fitter)**, and then enable **Perform Physical Synthesis for Combinational Logic for Performance** and **Perform Physical Synthesis for Combinational Logic for Fitting**.
3. View physical synthesis results in the **Netlist Optimizations** report.

6.1.2. Physical Synthesis Options

The Intel 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 22. Physical Synthesis Options

Option	Description
Perform asynchronous signal pipelining (no Intel Arria 10 support)	Automatically inserts pipeline stages for asynchronous clear and asynchronous load signals during fitting to increase circuit performance. This option is useful for asynchronous signals that are failing recovery and removal timing because they feed registers using a high-speed clock. You can use this option if asynchronous control signal recovery and removal times are not achieving requirements. This option adds registers and potential latency to nets driving the asynchronous clear or asynchronous load ports of registers. The additional register delays can change the behavior of the signal in the design; therefore, you should use this option only if additional latency on the reset signals does not violate any design requirements. This option also prevents the promotion of signals to global routing resources.
Perform Register Duplication for Performance (no Intel Arria 10 support)	Duplicates registers based on Fitter placement information to reduce the delay of one path without degrading the delay of another. You can also duplicate combinational logic when you enable this option. The Fitter can place the new logic cell closer to critical logic without affecting the other fan-out paths of the original logic cell. This setting does not apply to logic cells that are part of a chain, drive global signals, are constrained to a single LAB, or the **Netlist Optimizations** option set to **Never Allow**.
Perform Register Retiming for Performance (no Arria 10 support)	Enables the movement of registers across combinational logic, allowing the Quartus Prime software to trade off the delay between timing-critical paths and non-critical paths.
Perform Physical synthesis for combinational logic for Performance (no Intel Arria 10 support)	Performs physical synthesis optimizations on combinational logic during synthesis and fitting to increase circuit performance. Swaps the look-up table (LUT) ports within LEs so that the critical path has fewer layers through which to travel. Also allows the duplication of LUTs to enable further optimizations on the critical path.
Physical Synthesis for Combinational Logic for Fitting (no Intel Arria 10 support)	Reduces delay along critical paths. This option swaps the look-up table (LUT) ports within LEs so that the critical path has fewer layers through which to travel. The option also allows the duplication of LUTs to enable further optimizations on the critical path. The option causes registers that do not have a **Power-Up Level logic** option setting to power up with a don't care logic level (X). When the **Power-Up Don't Care** option is turned on, the Compiler determines when it is beneficial to change the power-up level of a register to minimize the area of the design. A power-up state of zero is maintained unless there is an immediate area advantage. The registers contained in the affected logic cells are not modified. Inputs into memory blocks, DSP blocks, and I/O elements

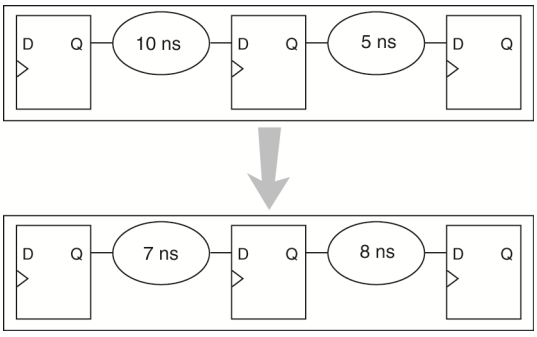
continued...

Option	Description
	(IOEs) are not swapped. This setting does not apply to logic cells that are part of a chain, drive global signals, are constrained to a single LAB, or the **Netlist Optimizations** option set to **Never Allow**.
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.
Physical Synthesis Effort Level (no Intel Arria 10 support)	Specifies the amount of effort, in terms of compile time, physical synthesis should use. Compared to the **Default** setting, a setting of **Extra** uses extra compile time to try to gain extra circuit performance. Conversely, a setting of **Fast** uses less compile time but may reduce the performance gain that physical synthesis is able to achieve.
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.
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.
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.

6.1.3. Perform Register Retiming for Performance

The **Perform Register Retiming for Performance** option enables the movement of registers across combinational logic, allowing the Intel Quartus Prime software to trade off the delay between timing-critical paths and non-critical paths. Register retiming can be done during Intel Quartus Prime integrated synthesis or during the Fitter stages of design compilation.

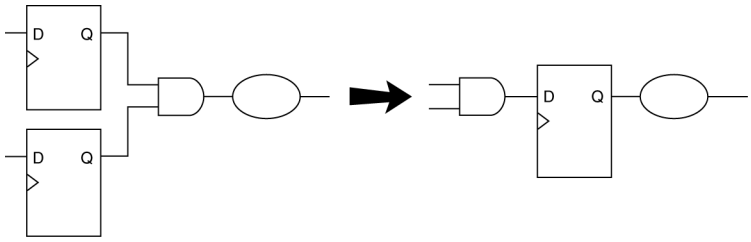
Figure 49. Reducing Critical Delay by Moving the Register Relative to Combinational Logic



Retiming can create multiple registers at the input of a combinational block from a register at the output of a combinational block. In this case, the new registers have the same clock and clock enable. The asynchronous control signals and power-up level

are derived from previous registers to provide equivalent functionality. Retiming can also combine multiple registers at the input of a combinational block to a single register.

Figure 50. Combining Registers with Register Retiming



To move registers across combinational logic to balance timing, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**. Specify your preferred option under **Optimize for performance (physical synthesis)** and **Effort level** .

6.1.4. Preventing Register Movement During Retiming

If you want to prevent register movement during register retiming, you can set the **Netlist Optimizations** logic option to **Never Allow**. You can apply this option to either individual registers or entities in the design using the Assignment Editor.

In digital circuits, synchronization registers are instantiated on cross clock domain paths to reduce the possibility of metastability. The Intel Quartus Prime software detects such synchronization registers and does not move them, even if register retiming is turned on.

The following sets of registers are not moved during register retiming:

- Both registers in a direct connection from input pin-to-register-to-register if both registers have the same clock and the first register does not fan-out to anywhere else. These registers are considered synchronization registers.
- Both registers in a direct connection from register-to-register if both registers have the same clock, the first register does not fan out to anywhere else, and the first register is fed by another register in a different clock domain (directly or through combinational logic). These registers are considered synchronization registers.

The Intel Quartus Prime software does not perform register retiming on logic cells that have the following properties:

- Are part of a cascade chain
- Contain registers that drive asynchronous control signals on another register
- Contain registers that drive the clock of another register
- Contain registers that drive a register in another clock domain
- Contain registers that are driven by a register in another clock domain

Note: The Intel Quartus Prime software does not usually retime registers across different clock domains; however, if you use the Classic Timing Analyzer and specify a global f_{MAX} requirement, the Intel Quartus Prime software interprets all clocks as related. Consequently, the Intel Quartus Prime software might try to retime register-to-register paths associated with different clocks.

To avoid this circumstance, provide individual f_{MAX} requirements to each clock when using Classic Timing Analysis. When you constrain each clock individually, the Intel Quartus Prime software assumes no relationship between different clock domains and considers each clock domain to be asynchronous to other clock domains; hence no register-to-register paths crossing clock domains are retimed.

When you use the Timing Analyzer, register-to-register paths across clock domains are never retimed, because the Timing Analyzer treats all clock domains as asynchronous to each other unless they are intentionally grouped.

- Contain registers that are constrained to a single LAB location
- Contain registers that are connected to SERDES
- Are considered virtual I/O pins
- Registers that have the **Netlist Optimizations** logic option set to **Never Allow**

The Intel Quartus Prime software assumes that a synchronization register chain consists of two registers. If your design has synchronization register chains with more than two registers, you must indicate the number of registers in your synchronization chains so that they are not affected by register retiming. To do this, perform the following steps:

1. Click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**.
2. Modify the **Synchronization Register Chain Length** setting to match the synchronization register length used in your design. If you set a value of 1 for the **Synchronization Register Chain Length**, it means that any registers connected to the first register in a register-to-register connection can be moved during retiming. A value of n > 1 means that any registers in a sequence of length 1, 2,… *n* are not moved during register retiming.

   If you want to consider logic cells that meet any of these conditions for physical synthesis, you can override these rules by setting the **Netlist Optimizations** logic option to **Always Allow** on a given set of registers.

Related Information

Analyzing and Optimizing the Design Floorplan on page 102

6.2. 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.

Applying all the physical synthesis options at the **Extra** effort level generally produces the best results for those options, but adds significantly to the compilation time. You can also use the **Physical synthesis effort level** options to decrease the compilation time. The WYSIWYG primitive resynthesis option does not add much compilation time relative to the overall design compilation time.

To find the best results, you can use the Intel Quartus Prime Design Space Explorer II (DSE) to apply various sets of netlist optimization options.

Related Information

Design Space Explorer II on page 12

6.2.1. 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 Intel 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 Intel 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/cascade 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).

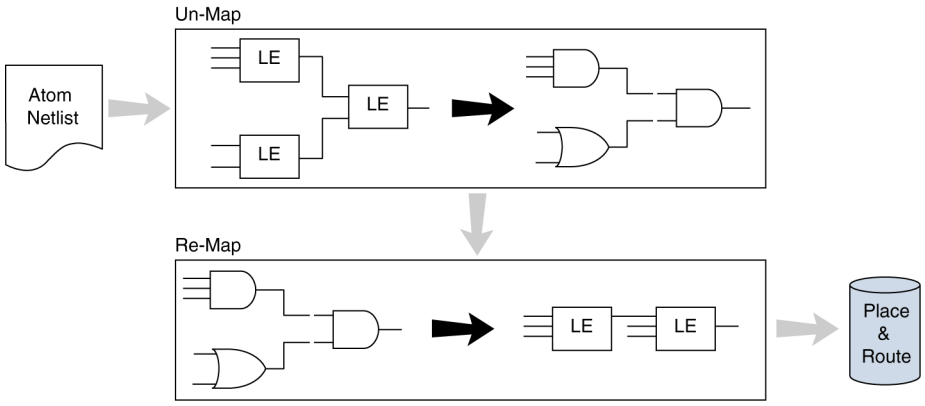
The **Perform WYSIWYG primitive resynthesis**option can change node names in the `.vqm` file or `.edf` file from your third-party synthesis tool, because the primitives in the atom netlist are broken apart and then re-mapped by the Intel Quartus Prime software. The re-mapping process removes duplicate 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 Intel Quartus Prime software reads.

If you use the Intel 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 51. Intel Quartus Prime Flow for WYSIWYG Primitive Resynthesis



6.2.2. Saving a Node-Level Netlist

For non-Intel Arria 10 designs, you can preserve a node-level netlist in Verilog Quartus Mapping File (`.vqm`) format. You might need to preserve nodes if you use the Logic Lock (Standard) flow to back-annotate placement, import one design into another, or both. For all device families that support incremental compilation, you can use this feature to preserve compilation results.

Note: This feature does not support Intel Arria 10 devices.

Use the **Export version-compatible database** option to save synthesis results as an atom-based netlist in `.vqm` file format. By default, the Intel Quartus Prime software places the `.vqm` in the **atom_netlists** directory under the current project directory.

If you use the physical synthesis optimizations and want to lock down the location of all LEs and other device resources in the design with the **Back-Annotate Assignments** command, a `.vqm` file netlist is required. The `.vqm` file preserves the changes that you made to your original netlist. Because the physical synthesis optimizations depend on the placement of the nodes in the design, back-annotating the placement changes the results from physical synthesis. Changing the results means that node names are different, and your back-annotated locations are no longer valid.

You should not use an Intel Quartus Prime-generated `.vqm` file or back-annotated location assignments with physical synthesis optimizations unless you have finalized the design. Making any changes to the design invalidates your physical synthesis results and back-annotated location assignments. If you require changes later, use the new source HDL code as your input files, and remove the back-annotated assignments corresponding to the Intel Quartus Prime-generated `.vqm` file.

To back-annotate logic locations for a design that was compiled with physical synthesis optimizations, first create a `.vqm` file. When recompiling the design with the hard logic location assignments, use the new `.vqm` file as the input source file and turn off the physical synthesis optimizations for the new compilation.

If you are importing a `.vqm` file and back-annotated locations into another project that has any **Netlist Optimizations** turned on, you must apply the **Never Allow** constraint to make sure node names don't change; otherwise, the back-annotated location or Logic Lock (Standard) assignments are invalid.

To preserve the nodes from Intel Quartus Prime physical synthesis optimization options for devices that do not support incremental compilation, perform the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.
2. In the **Category** list, select **Compilation Process Settings**. The **Compilation Process Settings** page appears.
3. Turn on**Export version-compatible database**. This setting is not available for some devices.
4. Click **OK**.

6.3. Viewing Synthesis and Netlist Optimization Reports

Physical synthesis optimizations performed during synthesis write results to the synthesis report. To access this report, perform the following steps:

1. On the Processing menu, click **Compilation Report**.
2. In the **Compilation Report** list, open the **Analysis & Synthesis**. folder to view synthesis results.
3. In the **Compilation Report** list, open the **Fitter** folder to view the **Netlist Optimizations** table.

6.4. 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 Intel 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

- Command Line Scripting
- Tcl Scripting

- API Functions for Tcl
- Intel Quartus Prime Standard Edition Settings File Reference Manual
  For information about all settings and constraints in the Intel Quartus Prime software.

6.4.1. 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 23. Synthesis Netlist Optimizations and Associated Settings

Setting Name	Intel 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	BALANCEDHIGH PERFORMANCE EFFOR AGGRESSIVE PERFORMANCE	Global, Instance
Power-Up Don't Care	ALLOW_POWER_UP_DONT_CARE	ON, OFF	Global
Save a node-level netlist into a persistent source file	LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT	ON, OFF	Global
	LOGICLOCK_INCREMENTAL_COMPILE_FILE	*<file name>*	
Allow Netlist Optimizations	ADV_NETLIST_OPT_ALLOWED	"ALWAYS ALLOW", DEFAULT, "NEVER ALLOW"	Instance

6.4.2. 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 24. Physical Synthesis Optimizations and Associated Settings

Setting Name	Intel Quartus Prime Settings File Variable Name	Values	Type
Perform Physical Synthesis for Combinational Logic for Performance (no Arria 10 support)	PHYSICAL_SYNTHESIS_COMBO_LOGIC	ON, OFF	Global
Perform Physical Synthesis for Combinational Logic for Fitting (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_COMBO_LOGIC_FOR_AREA	ON, OFF	Global
Advanced Physical Synthesis	ADVANCED_PHYSICAL_SYNTHESIS	ON, OFF	Global

continued...

Setting Name	Intel Quartus Prime Settings File Variable Name	Values	Type
Automatic Asynchronous Signal Pipelining	PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING	ON, OFF	Global
Perform Register Duplication for Performance (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION	ON, OFF	Global
Perform Register Retiming for Performance (no Intel Arria 10 support)	PHYSICAL_SYNTHESIS_REGISTER_RETIMING	ON, OFF	Global
Power-Up Don't Care	ALLOW_POWER_UP_DONT_CARE	ON, OFF	Global, Instance
Power-Up Level	POWER_UP_LEVEL	HIGH, LOW	Instance
Allow Netlist Optimizations	ADV_NETLIST_OPT_ALLOWED	"ALWAYS ALLOW", DEFAULT, "NEVER ALLOW"	Instance
Save a node-level netlist into a persistent source file (no Intel Arria 10 support)	LOGICLOCK_INCREMENTAL_COMPILE_ASSIGNMENT	ON, OFF	Global
	LOGICLOCK_INCREMENTAL_COMPILE_FILE	*<file name>*	

6.4.3. Back-Annotating Assignments

You can use the `logiclock_back_annotate` Tcl command to back-annotate resources in your design. This command can back-annotate resources in Logic Lock (Standard) regions, and resources in designs without Logic Lock (Standard) regions.

The following Tcl command back-annotates all registers in your design:

```
logiclock_back_annotate -resource_filter "REGISTER"
```

The `logiclock_back_annotate` command is in the `backannotate` package.

6.5. Netlist Optimizations and Physical Synthesis Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2018.05.07	18.0.0	Removed topic: *Isolating a Partition Netlist*.
2017.11.06	17.1.0	• Added topic: *Isolating a Partition Netlist*.
2016.10.31	16.1.0	• Updated physical synthesis options and procedure.
2016.05.02	16.0.0	• Stated limitations about deprecated physical synthesis options.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.

continued...

Document Version	Intel Quartus Prime Version	Changes
2014.12.15	14.1.0	• Updated location of Fitter Settings, Analysis & Synthesis Settings, and Physical Synthesis Optimizations Settings to Compiler Settings. • Updated DSE II content.
June 2014	14.0.0	Updated format.
November 2013	13.1.0	Removed HardCopy device information.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Template update.
July 2010	10.0.0	• Added links to Intel Quartus Prime Help in several sections. • Removed Referenced Documents section. • Reformatted Document Revision History
November 2009	9.1.0	• Added information to "Physical Synthesis for Registers—Register Retiming" • Added information to "Applying Netlist Optimization Options" • Made minor editorial updates
March 2009	9.0.0	• 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	8.1.0	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0.0	• Updated "Physical Synthesis Optimizations for Performance on page 11-9 • Added Physical Synthesis Options for Fitting on page 11-16

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

7. Engineering Change Orders with the Chip Planner

Programmable logic can accommodate changes to a system specification late in the design cycle. In a typical engineering project development cycle, the specification of the programmable logic portion is likely to change after engineering begins or while integrating all system elements. Last-minute design changes, commonly referred to as engineering change orders (ECOs), are small targeted changes to the functionality of a design after the design has been fully compiled.

The Chip Planner supports ECOs by allowing quick and efficient changes to your logic late in the design cycle for supported devices. The Chip Planner provides a visual display of your post-place-and-route design mapped to the device architecture of your chosen FPGA and allows you to create, move, and delete logic cells and I/O atoms.

Note: The Intel Quartus Prime Standard Edition ECO feature does not support Intel Arria 10 devices.

In addition to making ECOs, the Chip Planner allows you to perform detailed analysis on routing congestion, relative resource usage, logic placement, Logic Lock (Standard) regions, fan-ins and fan-outs, paths between registers, and delay estimates for paths.

ECOs directly apply to atoms in the supported target device. As such, performing an ECO relies on your understanding of the device architecture of the target device.

Related Information

- Analyzing and Optimizing the Design Floorplan on page 102
  For more information about using the Chip Planner for design analysis
- Literature
  For more information about the architecture of your device

7.1. Engineering Change Orders

In the context of an FPGA design, you can apply an ECO directly to a physical resource on the device to modify its behavior. ECOs are typically made during the verification stage of a design cycle. When a small change is required on a design (such as modifying a PLL for a different clock frequency or routing a signal out to a pin for analysis) recompilation of the entire design can be time consuming, especially for larger designs.

Because several iterations of small design changes can occur during the verification cycle, recompilation times can quickly add up. Furthermore, a full recompilation due to a small design change can result in the loss of previous design optimizations. Making ECOs, instead of performing a full recompilation on your design, limits the change only to the affected portions of logic.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

7.1.1. Performance Preservation

You can preserve the results of previous design optimizations when you make changes to an existing design with one of the following methods:

- Incremental compilation
- Rapid recompile
- ECOs

Choose the method to modify your design based on the scope of the change. The methods above are arranged from the larger scale change to the smallest targeted change to a compiled design.

The incremental compilation feature allows you to preserve compilation results at an RTL component or module level. After the initial compilation of your design, you can assign modules in your design hierarchy to partitions. Upon subsequent compilations, incremental compilation recompiles changed partitions based on the chosen preservation levels.

The rapid recompilation feature leverages results from the latest post-fit netlist to determine the changes required to honor modifications you have made to the source code. If you run a rapid recompilation, the Compiler refits only changed portion of the netlist.

ECOs provide a finer granularity of control compared to the incremental compilation and the rapid recompilation feature. All modifications are performed directly on the architectural elements of the device. You should use ECOs for targeted changes to the post-fit netlist.

Note: In the Intel Quartus Prime software versions 10.0 and later, the software does not preserve ECO modifications to the netlist when you recompile a design with the incremental compilation feature turned on. You can reapply ECO changes made during a previous compilation with the Change Manager.

Related Information

Intel Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design

7.1.2. Compilation Time

In the traditional programmable logic design flow, a small change in the design requires a complete recompilation of the design. A complete recompilation of the design consists of synthesis and place-and-route. Making small changes to the design to reach the final implementation on a board can be a long process. Because the Chip Planner works only on the post-place-and-route database, you can implement your design changes in minutes without performing a full compilation.

7.1.3. Verification

After you make a design change, you can verify the impact on your design. To verify that your changes do not violate timing requirements, perform static timing analysis with the Intel Quartus Prime Timing Analyzer after you check and save your netlist changes in the Chip Planner.

Additionally, you can perform a gate-level or timing simulation of the ECO-modified design with the post-place-and-route netlist generated by the Intel Quartus Prime software.

Related Information

Intel Quartus Prime Timing Analyzer User Guide

7.1.4. Change Modification Record

All ECOs made with the Chip Planner are logged in the Change Manager to track all changes. With the Change Manager, you can easily revert to the original post-fit netlist or you can pick and choose which ECOs to apply.

Additionally, the Intel Quartus Prime software provides support for multiple compilation revisions of the same project. You can use ECOs made with the Chip Planner in conjunction with revision support to compare several different ECO changes and revert back to previous project revisions when required.

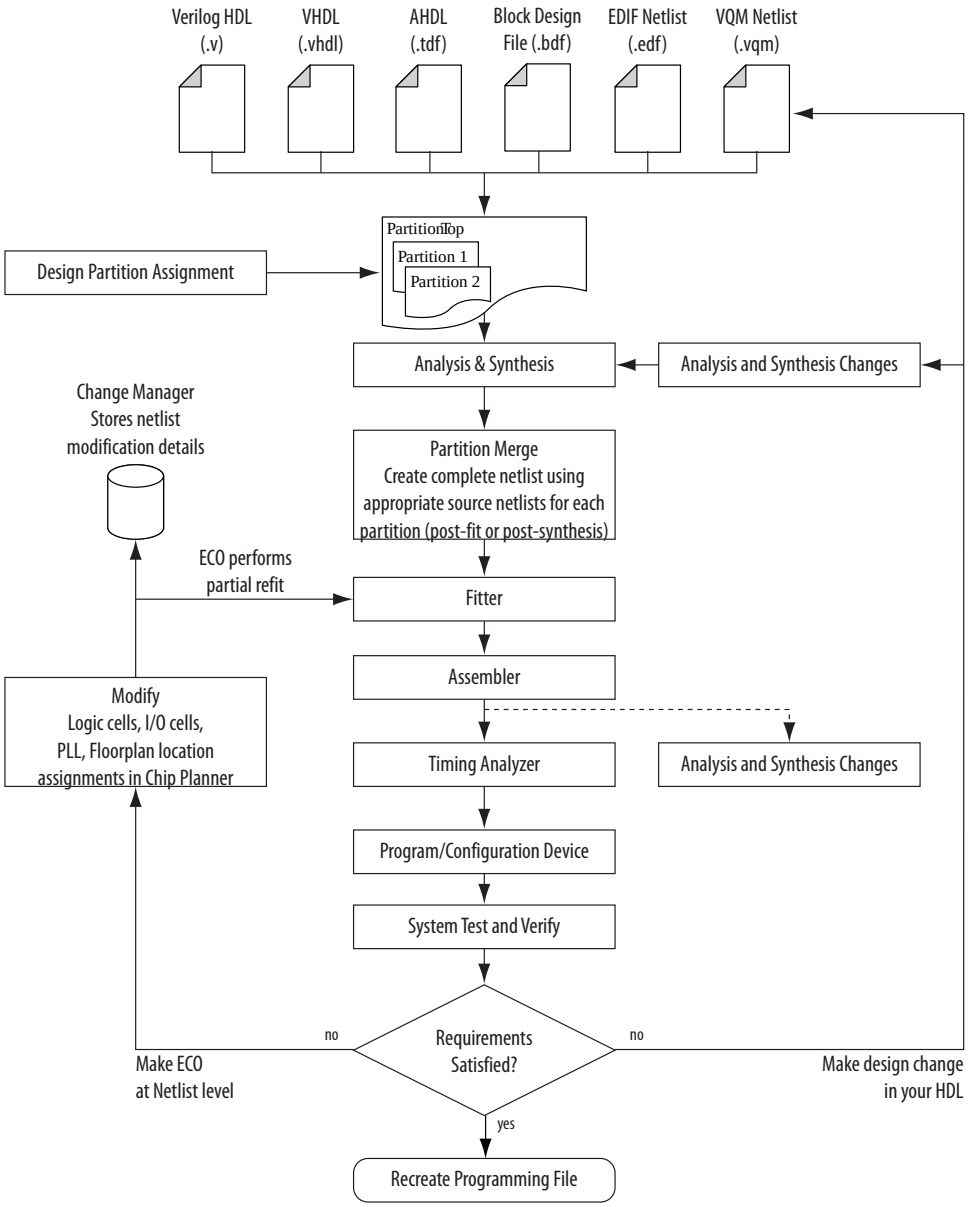
7.2. ECO Design Flow

For iterative verification cycles, implementing small design changes at the netlist level can be faster than making an RTL code change. As such, making ECO changes are especially helpful when you debug the design on silicon and require a fast turnaround time to generate a programming file for debugging the design.

Note: The Intel Quartus Prime Standard Edition ECO feature does not support Intel Arria 10 devices.

The figure shows the design flow for making ECOs.

Figure 52. Design Flow to Support ECOs



A typical ECO application occurs when you uncover a problem on the board and isolate the problem to the appropriate nodes or I/O cells on the device. You must be able to correct the functionality quickly and generate a new programming file. By making small changes with the Chip Planner, you can modify the post-place-and-route netlist directly without having to perform synthesis and logic mapping, thus decreasing the turnaround time for programming file generation during the verification cycle. If the

change corrects the problem, no modification of the HDL source code is necessary. You can use the Chip Planner to perform the following ECO-related changes to your design:

- Document the changes made with the Change Manager
- Easily recreate the steps taken to produce design changes
- Generate EDA simulation netlists for design verification

Note: For more complex changes that require HDL source code modifications, the incremental compilation feature can help reduce recompilation time.

7.3. The Chip Planner Overview

The Chip Planner provides a visual display of device resources. It shows the arrangement and usage of the resource atoms in the device architecture that you are targeting. Resource atoms are the building blocks for your device, such as ALMs, LEs, PLLs, DSP blocks, memory blocks, or I/O elements.

The Chip Planner also provides an integrated platform for design analysis and for making ECOs to your design after place-and-route. The toolset consists of the Chip Planner (providing a device floorplan view of your mapped design) and two integrated subtools—the Resource Property Editor and the Change Manager.

For analysis, the Chip Planner can show logic placement, Logic Lock (Standard) regions, relative resource usage, detailed routing information, routing congestion, fan-ins and fan-outs, paths between registers, and delay estimates for paths. Additionally, the Chip Planner allows you to create location constraints or resource assignment changes, such as moving or deleting logic cells or I/O atoms with the device floorplan. For ECO changes, the Chip Planner enables you to create, move, or delete logic cells in the post-place-and-route netlist for fast programming file generation. Additionally, you can open the Resource Property Editor from the Chip Planner to edit the properties of resource atoms or to edit the connections between resource atoms. All changes to resource atoms and connections are logged automatically with the Change Manager.

7.3.1. Opening the Chip Planner

To open the Chip Planner, on the Tools menu, click **Chip Planner**. Alternatively, click the **Chip Planner** icon on the Intel Quartus Prime software toolbar.

Optionally, you can open the Chip Planner by cross-probing from the shortcut menu in the following tools:

- Design Partition Planner
- Compilation Report
- Logic Lock (Standard) Regions window
- Technology Map Viewer
- Project Navigator window
- RTL source code
- Node Finder

- Simulation Report
- RTL Viewer
- Report Timing panel of the Timing Analyzer

7.3.2. The Chip Planner Tasks and Layers

The Chip Planner allows you to set up tasks to quickly implement ECO changes or manipulate assignments for the floorplan of the device. Each task consists of an editing mode and a set of customized layer settings.

Related Information

- Performing ECOs in the Resource Property Editor on page 147
- Analyzing and Optimizing the Design Floorplan on page 102

7.4. Performing ECOs with the Chip Planner (Floorplan View)

You can manipulate resource atoms in the Chip Planner when you select the ECO editing mode.

The following ECO changes can be made with the Chip Planner Floorplan view:

- Create atoms
- Delete atoms
- Move existing atoms

Note: To configure the properties of atoms, such as managing the connections between different LEs/ALMs, use the Resource Property Editor.

To select the ECO editing mode in the Chip Planner, in the **Editing Mode** list at the top of the Chip Planner, select the ECO editing mode.

Related Information

Performing ECOs in the Resource Property Editor on page 147

7.4.1. Creating, Deleting, and Moving Atoms

You can use the Chip Planner to create, delete, and move atoms in the post-compilation design.

7.4.2. Check and Save Netlist Changes

After making all the ECOs, you can run the Fitter to incorporate the changes by clicking the **Check and Save Netlist Changes** icon in the Chip Planner toolbar. The Fitter compiles the ECO changes, performs design rule checks on the design, and generates a programming file.

7.5. Performing ECOs in the Resource Property Editor

You can view and edit the following resources with the Resource Property Editor.

7.5.1. Logic Elements

An Altera® LE contains a four-input LUT, which is a function generator that can implement any function of four variables. In addition, each LE contains a register fed by the output of the LUT or by an independent function generated in another LE.

You can use the Resource Property Editor to view and edit any LE in the FPGA. To open the Resource Property Editor for an LE, on the Project menu, point to **Locate,** and then click **Locate in Resource Property Editor** in one of the following views:

- RTL Viewer
- Technology Map Viewer
- Node Finder
- Chip Planner

  For more information about LE architecture for a particular device family, refer to the device family handbook or data sheet.

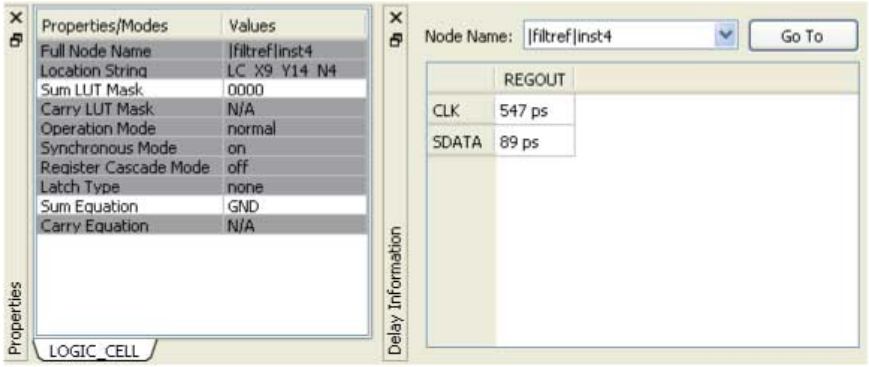
You can use the Resource Property Editor to change the following LE properties:

- Data input to the LUT
- LUT mask or LUT

7.5.1.1. Logic Element Properties

To view logic element properties, on the View menu, click **View Properties**.

Figure 53. LE Properties in the Resource Property Editor



7.5.1.2. Modes of Operation

LUTs in an LE can operate in either normal or arithmetic mode.

When an LE is configured in normal mode, the LUT in the LE can implement a function of four inputs.

When the LE is configured in arithmetic mode, the LUT in the LE is divided into two 3-input LUTs. The first LUT generates the signal that drives the output of the LUT, while the second LUT generates the carry-out signal. The carry-out signal can drive only a carry-in signal of another LE.

For more information about LE modes of operation, refer to volume 1 of the appropriate device handbook.

7.5.1.3. Sum and Carry Equations

You can change the logic function implemented by the LUT by changing the sum and carry equations. When the LE is configured in normal mode, you can change only the sum equation. When the LE is configured in arithmetic mode, you can change both the sum and the carry equations.

The LUT mask is the hexadecimal representation of the LUT equation output. When you change the LUT equation, the Intel Quartus Prime software automatically changes the LUT mask. Conversely, when you change the LUT mask, the Intel Quartus Prime software automatically computes the LUT equation.

7.5.1.4. sload and sclr Signals

Each LE register contains a synchronous load (`sload`) signal and a synchronous clear (`sclr`) signal. You can invert either the `sload` or `sclr` signal feeding into the LE.

If the design uses the `sload` signal in an LE, the signal and its inversion state must be the same for all other LEs in the same LAB. For example, if two LEs in a LAB have the `sload` signal connected, both LEs must have the `sload` signal set to the same value. This is also true for the `sclr` signal.

7.5.1.5. Register Cascade Mode

When register cascade mode is enabled, the cascade-in port feeds the input to the register. The register cascade mode is used most often when the design implements shift registers.

You can change the register cascade mode by connecting (or disconnecting) the cascade in the port. However, if you create this port, you must ensure that the source port LE is directly above the destination LE.

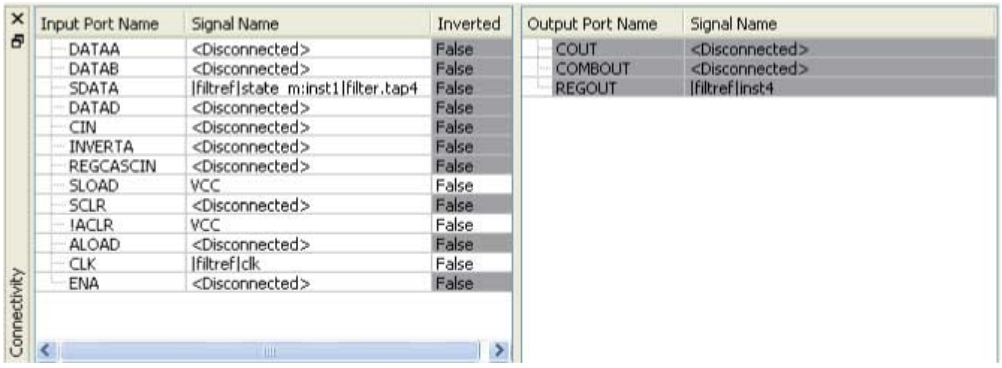
7.5.1.6. Cell Delay Table

The cell delay table describes the propagation delay from all inputs to all outputs for the selected LE.

7.5.1.7. Logic Element Connections

To view the connections that feed in and out of an LE, on the View menu, click **View Port Connections**.

Figure 54. View LE Connections in the Connectivity Window



7.5.1.8. Deleting a Logic Element

To delete an LE, follow these steps:

1. Right-click the desired LE in the Chip Planner, point to **Locate**, and click **Locate in Resource Property Editor**.
2. You must remove all fan-out connections from an LE prior to deletion. To delete fan-out connections, right-click each connected output signal, point to **Remove**, and click **Fanouts**. Select all of the fan-out signals in the **Remove Fan-outs** dialog box and click **OK**.
3. To delete an atom after all fan-out connections are removed, right-click the atom in the Chip Planner and click **Delete Atom**.

7.5.2. Adaptive Logic Modules

Each ALM contains LUT-based resources that can be divided between two adaptive LUTs (ALUTs).

With up to eight inputs to the two ALUTs, each ALM can implement various combinations of two functions. This adaptability allows the ALM to be completely backward-compatible with four-input LUT architectures. One ALM can implement any function with up to six inputs and certain seven-input functions. In addition to the ALUT-based resources, each ALM contains two programmable registers, two dedicated full adders, a carry chain, a shared arithmetic chain, and a register chain. The ALM can efficiently implement various arithmetic functions and shift registers with these dedicated resources.

You can implement the following types of functions in a single ALM:

- Two independent 4-input functions
- An independent 5-input function and an independent 3-input function
- A 5-input function and a 4-input function, if they share one input
- Two 5-input functions, if they share two inputs
- An independent 6-input function
- Two 6-input functions, if they share four inputs and share the same functions
- Certain 7-input functions

You can use the Resource Property Editor to change the following ALM properties:

- Data input to the LUT
- LUT mask or LUT equation

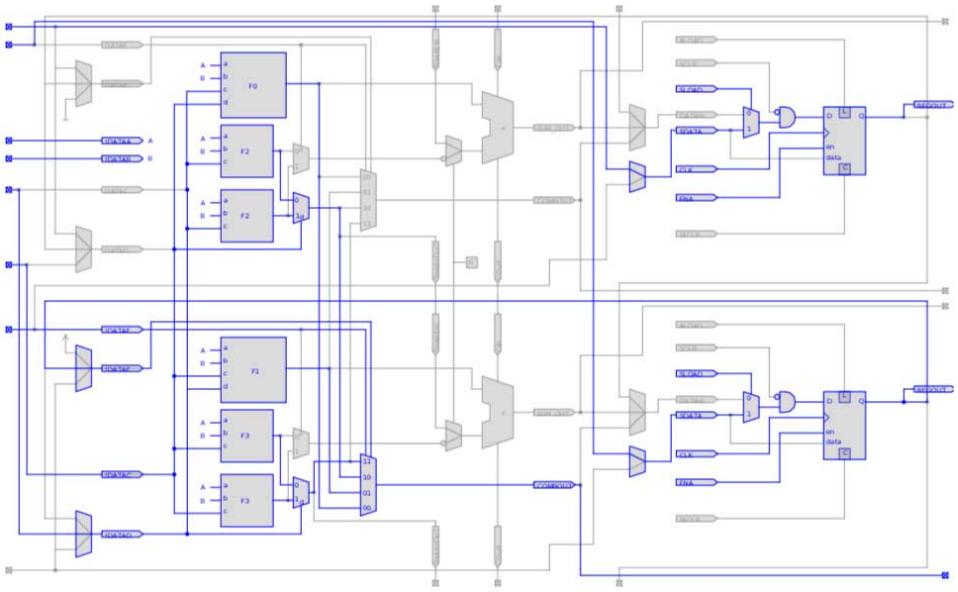
7.5.2.1. Adaptive Logic Module Schematic

You can view and edit any ALM atom with the Resource Property Editor by right-clicking the ALM in the RTL Viewer, the Node Finder, or the Chip Planner, and clicking **Locate in Resource Property Editor**.

For a detailed description of the ALM, refer to the device handbooks of devices based on an ALM architecture.

By default, the Intel Quartus Prime software displays the used resources in blue and the unused in gray. For the figure, the used resources are in blue and the unused resources are in gray.

Figure 55. Adaptive Logic Module



7.5.2.2. Adaptive Logic Module Properties

The properties that you can display for the ALM include an equations table that shows the name and location of each of the two combinational nodes and two register nodes in the ALM, the individual LUT equations for each of the combinational nodes, and the `combout`, `sumout`, `carryout`, and `shareout` equations for each combinational node.

7.5.2.3. Adaptive Logic Module Connections

Click **View > View Connectivity** to view the input and output connections for the ALM.

7.5.3. FPGA I/O Elements

Altera FPGAs that have high-performance I/O elements, including up to six registers, are equipped with support for a number of I/O standards that allow you to run your design at peak speeds. Use the Resource Property Editor to view, change connectivity, and edit the properties of the I/O elements. Use the Chip Planner (Floorplan view) to change placement, delete, and create new I/O elements.

For a detailed description of the device I/O elements, refer to the applicable device handbook.

You can change the following I/O properties:

- Delay chain
- Bus hold
- Weak pull up
- Slow slew rate

- I/O standard
- Current strength
- Extend OE disable
- PCI I/O
- Register reset mode
- Register synchronous reset mode
- Register power up
- Register mode

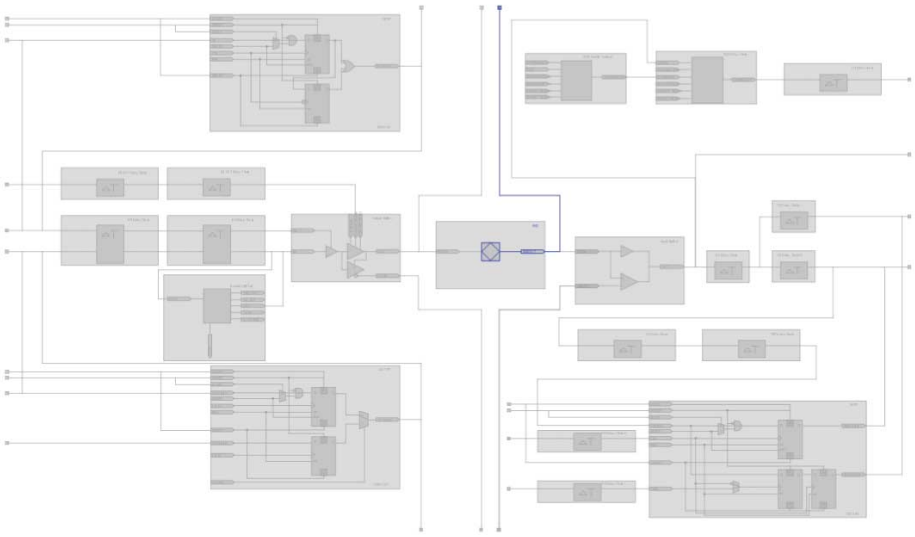
7.5.3.1. Stratix V I/O Elements

The I/O elements in Stratix® V devices contain a bidirectional I/O buffer and I/O registers to support a complete embedded bidirectional single data rate (SDR) or double data rate (DDR) transfer.

I/O registers are composed of the input path for handling data from the pin to the core, the output path for handling data from the core to the pin, and the output enable path for handling the output enable signal to the output buffer. These registers allow faster source-synchronous register-to-register transfers and resynchronization. The input path consists of the DDR input registers, alignment and synchronization registers, and half data rate blocks; you can bypass each block in the input path. The input path uses the deskew delay to adjust the input register clock delay across process, voltage, and temperature (PVT) variations.

By default, the Intel Quartus Prime software displays the used resources in blue and the unused resources in gray.

Figure 56. Stratix V Device I/O Element Structure



Related Information

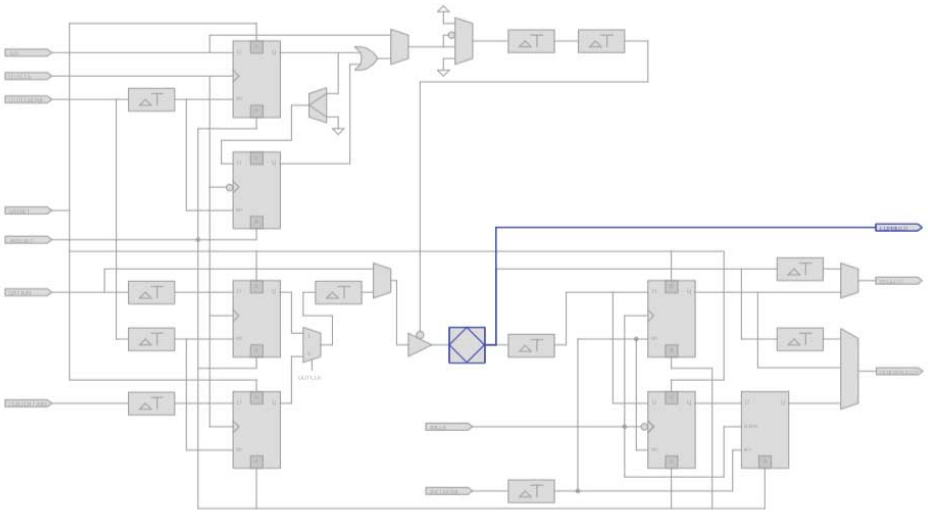
Stratix V Device Handbook

7.5.3.2. Stratix IV I/O Elements

The I/O elements in Stratix IV devices contain a bidirectional I/O buffer and I/O registers to support a complete embedded bidirectional SDR or DDR transfer.

The I/O registers are composed of the input path for handling data from the pin to the core, the output path for handling data from the core to the pin, and the output enable path for handling the output enable signal for the output buffer. Each path consists of a set of delay elements that allow you to fine-tune the timing characteristics of each path for skew management. By default, the Intel Quartus Prime software displays the used resources in blue and the unused resources in gray.

Figure 57. Stratix IV I/O Element and Structure



Related Information

Literature

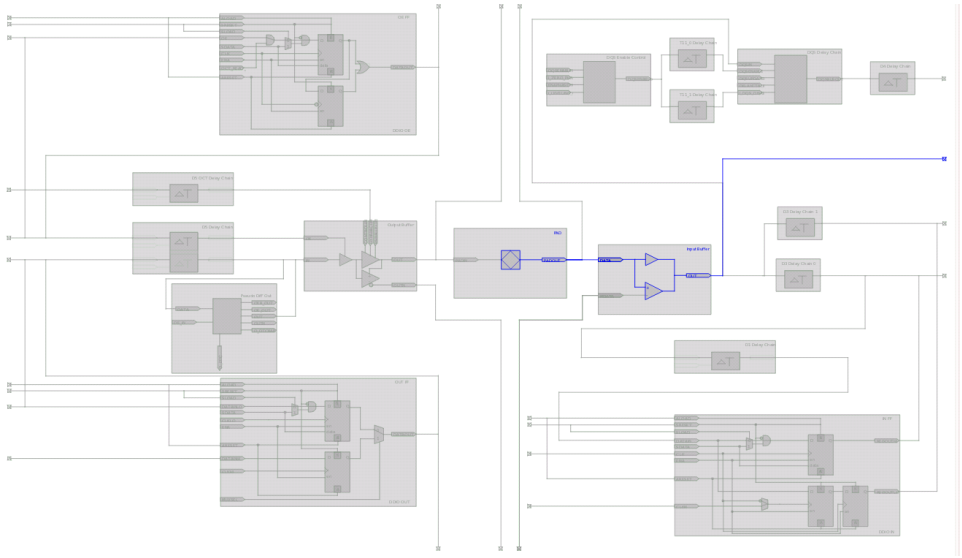
For more information about I/O elements in Stratix IV devices

7.5.3.3. Arria V I/O Elements

The I/O elements in Arria® V devices contain a bidirectional I/O buffer and I/O registers to support a complete embedded bidirectional SDR or DDR transfer.

The I/O registers are composed of the input path for handling data from the pin to the core, the output path for handling data from the core to the pin, and the output enable path for handling the output enable signal for the output buffer. Each path consists of a set of delay elements that allow you to fine-tune the timing characteristics of each path for skew management. By default, the Intel Quartus Prime software displays the used resources in blue and the unused resources in gray.

Figure 58. Arria V Device I/O Element and Structure

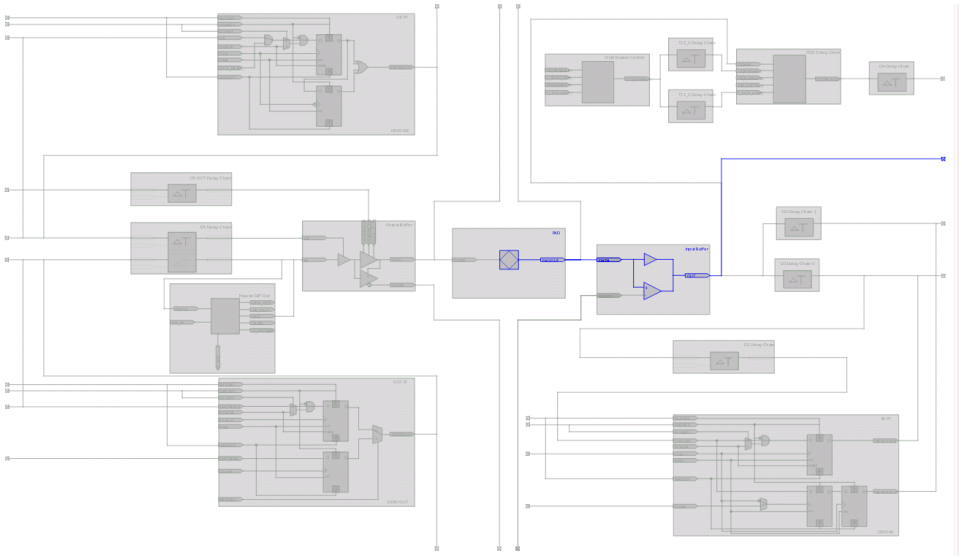


7.5.3.4. Cyclone V I/O Elements

The I/O elements in Cyclone V devices contain a bidirectional I/O buffer and registers for complete embedded bidirectional single data rate transfer. The I/O element contains three input register, two output registers, and two output-enable registers. The two output registers and two output-enable registers are utilized for double-data rate (DDR) applications.

You can use the input registers for fast setup times and the output registers for fast clock-to-output times. Additionally, you can use the output-enable (OE) registers for fast clock-to-output enable timing. You can use I/O elements for input, output, or bidirectional data paths. By default, the Intel Quartus Prime software displays the used resources in blue and the unused resources in gray.

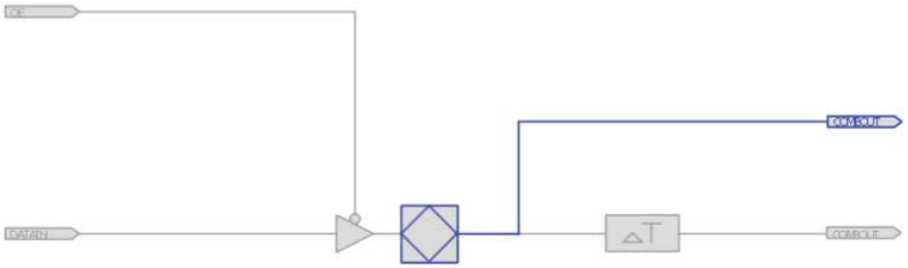
Figure 59. Cyclone V Device I/O Elements and Structure



7.5.3.5. MAX V I/O Elements

The I/O elements in MAX® V devices contain a bidirectional I/O buffer. You can drive registers from adjacent LABs to or from the bidirectional I/O buffer of the I/O element. By default, the Intel Quartus Prime software displays the used resources in blue and the unused resources in gray.

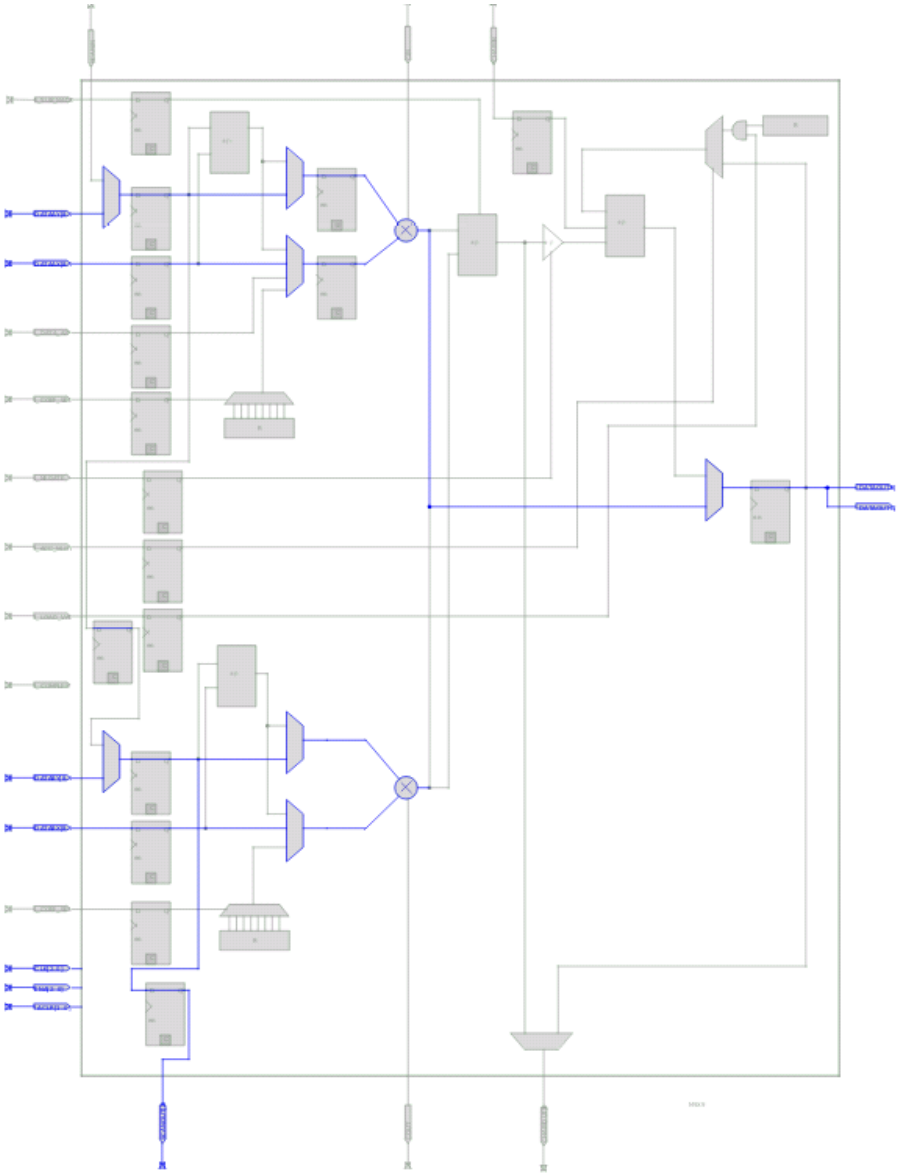
Figure 60. MAX V Device I/O Elements and Structure



7.5.4. FPGA RAM Blocks

With the Resource Property Editor, you can view the architecture of different RAM blocks in the device, modify the input and output registers to and from the RAM blocks, and modify the connectivity of the input and output ports. By default, the Intel Quartus Prime software displays the used resources in blue and the unused resources in gray.

Figure 61. M9K RAM View in a Stratix V Device

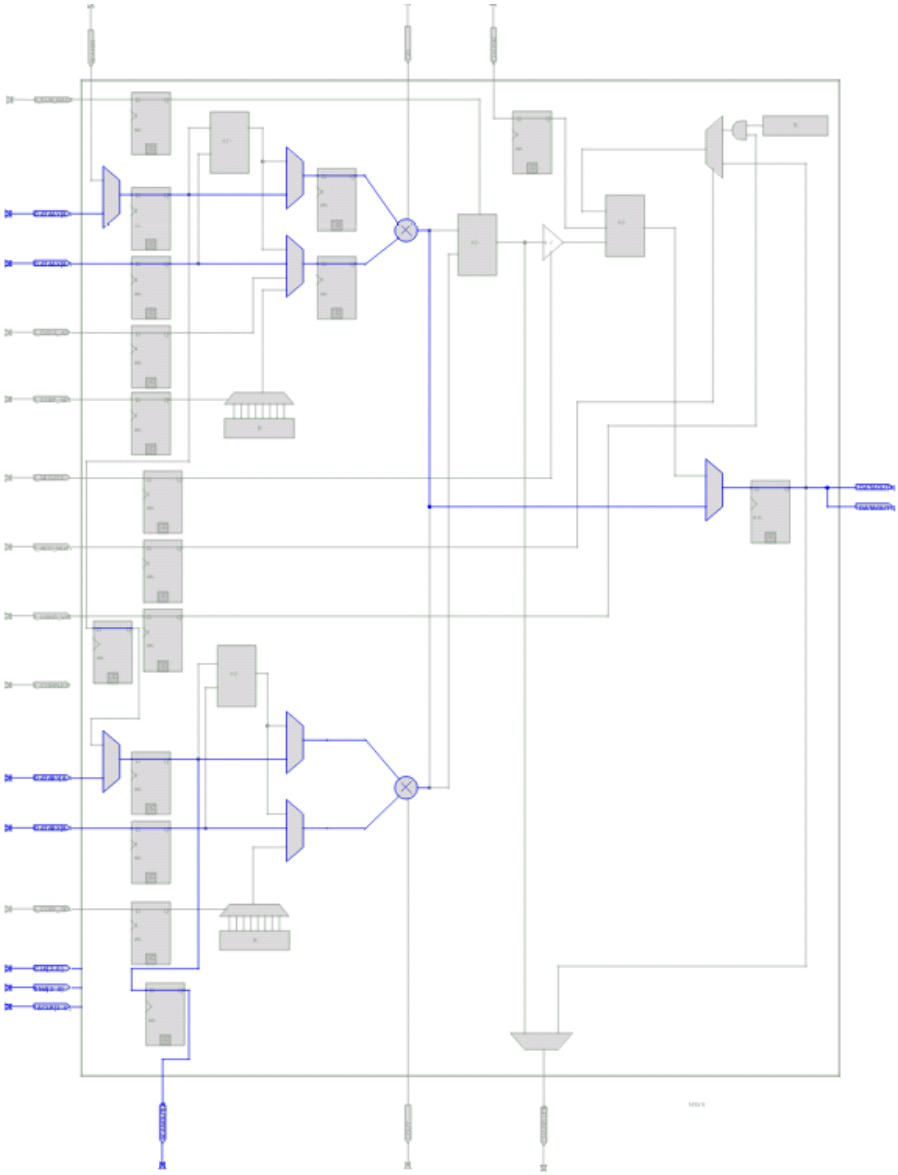


7.5.5. FPGA DSP Blocks

Dedicated hardware DSP circuit blocks in Altera devices provide performance benefits for the critical DSP functions in your design.

The Resource Property Editor allows you to view the architecture of DSP blocks in the Resource Property Editor for the Cyclone and Stratix series of devices. The Resource Property Editor also allows you to modify the signal connections to and from the DSP blocks and modify the input and output registers to and from the DSP blocks. By default, theIntel Quartus Prime software displays the used resources in blue and the unused resources in gray.

Figure 62. DSP Block View in a Stratix V Device



7.6. Change Manager

The Change Manager maintains a record of every change you perform with the Chip Planner, the Resource Property Editor, the Signal Probe feature, or a Tcl script. Each row of data in the Change Manager represents one ECO.

The Change Manager allows you to apply changes, roll back changes, delete changes, and export change records to a Text File (**.txt**), a Comma-Separated Value File (**.csv**), or a Tcl Script File (**.tcl**). The Change Manager tracks dependencies between changes, so that when you apply, roll back, or delete a change, any prerequisite or dependent changes are also applied, rolled back, or deleted.

7.6.1. Complex Changes in the Change Manager

Certain changes in the Change Manager (including creating or deleting atoms and changing connectivity) can appear to be self-contained, but are actually composed of multiple actions. The Change Manager marks such complex changes with a plus icon in the **Index** column.

You can click the plus icon to expand the change record and show all the component actions preformed as part of that complex change.

Related Information

Example of Managing Changes With the Change Manager

7.6.2. Managing Signal Probe Signals

The Signal Probe pins that you create from the **Signal Probe Pins** dialog box are recorded in the Change Manager. After you have made a Signal Probe assignment, you can use the Change Manager to quickly disable Signal Probe assignments by selecting **Revert to Last Saved Netlist** on the shortcut menu in the Change Manager.

Related Information

Quick Design Debugging Using Signal Probe

7.6.3. Exporting Changes

You can export changes to a **.txt**, a **`.csv`**, or a **`.tcl`**. Tcl scripts allow you to reapply changes that were deleted during compilation.

Related Information

Intel Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design

7.7. 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. The Tcl commands for controlling the Chip Planner are located in the `chip_planner` package of the `quartus_cdb` executable.

Related Information

- About Intel Quartus Prime Scripting
- Tcl Scripting
- Intel Quartus Prime Settings File Manual
- Command Line Scripting

7.8. Common ECO Applications

You can use an ECO to make a post-compilation change to your design.

To help build your system quickly, you can use Chip Planner functions to perform the following activities:

- Adjust the drive strength of an I/O with the Chip Planner
- Modify the PLL properties with the Resource Property Editor, see "Modify the PLL Properties With the Chip Planner"
- Modify the connectivity between new resource atoms with the Chip Planner and Resource Property Editor

Related Information

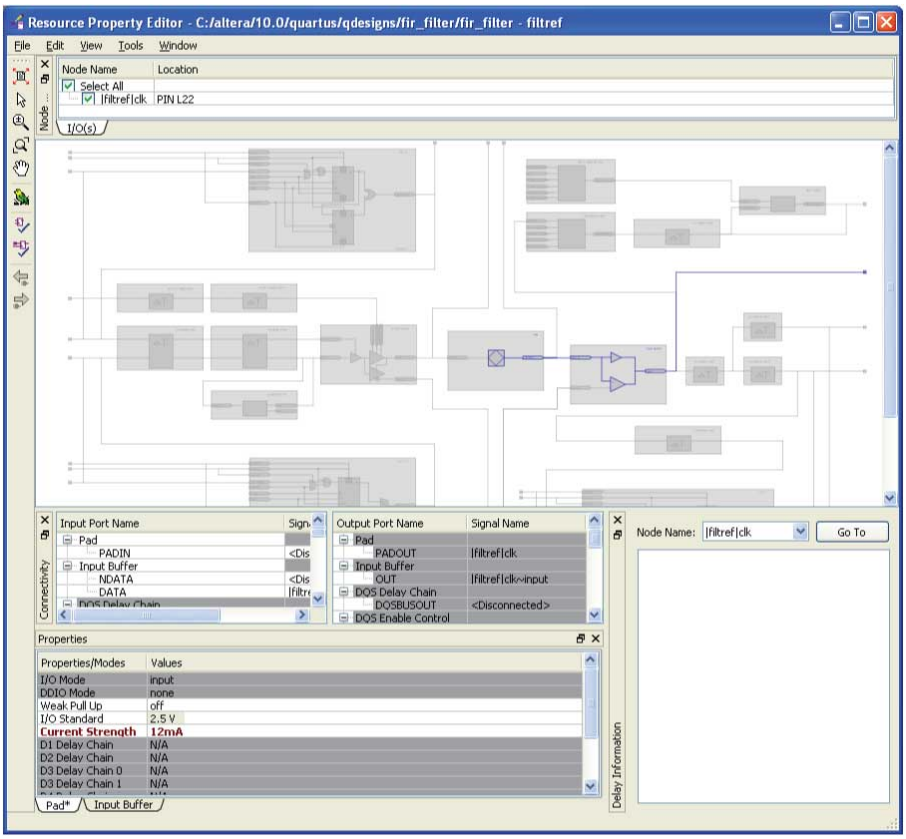
Modify the PLL Properties With the Chip Planner on page 160

7.8.1. Adjust the Drive Strength of an I/O with the Chip Planner

To adjust the drive strength of an I/O, follow these steps to incorporate the ECO changes into the netlist of the design.

1. In the **Editing Mode** list at the top of the Chip Planner, select the ECO editing mode.
2. Locate the I/O in the **Resource Property Editor**.
3. In the **Resource Property Editor**, point to the **Current Strength** option in the **Properties** pane and double-click the value to enable the drop-down list.
4. Change the value for the **Current Strength** option.
5. Right-click the ECO change in the Change Manager and click **Check & Save All Netlist Changes** to apply the ECO change.

Figure 63. I/O in the Resource Property Editor



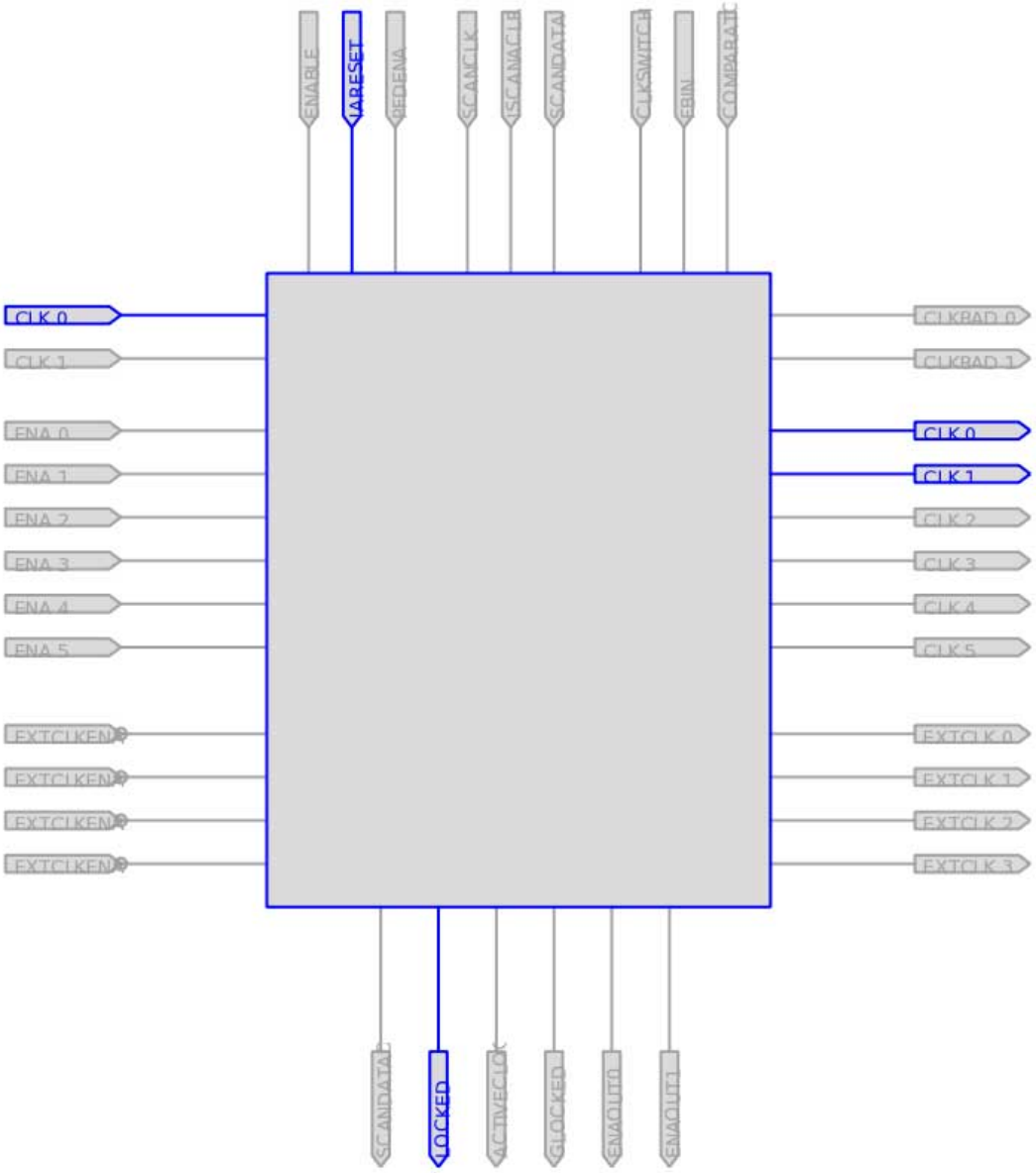
Note: You can change the pin locations of input or output ports with the ECO flow. You can drag and move the signal from an existing pin location to a new location while in the Post Compilation Editing (ECO) task in the Chip Planner. You can then click **Check & Save All Netlist Changes** to compile the ECO.

7.8.2. Modify the PLL Properties With the Chip Planner

You use PLLs to modify and generate clock signals to meet design requirements. Additionally, you can use PLLs to distribute clock signals to different devices in a design, reducing clock skew between devices, improving I/O timing, and generating internal clock signals.

The Resource Property Editor allows you to view and modify PLL properties to meet your design requirements.

Figure 64. PLL View in the Resource Property Editor of a Stratix Device



7.8.3. PLL Properties

The Resource Property Editor allows you to modify PLL options, such as phase shift, output clock frequency, and duty cycle.

You can also change the following PLL properties with the Resource Property Editor:

- Input frequency
- M V_{CO} tap
- M initial
- M value

- N value
- M counter delay
- N counter delay
- M2 value
- N2 value
- SS counter
- Charge pump current
- Loop filter resistance
- Loop filter capacitance
- Counter delay
- Counter high
- Counter low
- Counter mode
- Counter initial
- V_{CO} tap

You can also view post-compilation PLL properties in the Compilation Report. To do so, in the Compilation Report, select **Fitter** and then select **Resource Section**.

7.8.3.1. Adjusting the Duty Cycle

Use the equation to adjust the duty cycle of individual output clocks.

$$\text{High \%} = \frac{\text{Counter High}}{(\text{Counter High} + \text{Counter Low})}$$

7.8.3.2. Adjusting the Phase Shift

Use the equation to adjust the phase shift of an output clock of a PLL.

$$\text{Phase Shift} = (\text{Period } V_{co} \times 0.125 \times \text{Tap } V_{co}) + (\text{Initial } V_{co} \times \text{Period } V_{co})$$

For normal mode, Tap V_{CO}, Initial V_{CO}, and Period V_{CO} are governed by the following settings:

Tap V_{CO}= Counter Delay- M Tap V_{CO}

Initial V_{CO}= Counter Delay- M Initial

Period V_{CO}= In Clock Period x N÷M

For external feedback mode, Tap V_{CO}, Initial V_{CO}, and Period V_{CO} are governed by the following settings:

Tap V_{CO}= Counter Delay- M Tap V_{CO}

Initial V_{CO}= Counter Delay- M Initial

Period V_{CO}= $\underline{\text{In Clock Period x N}}$

(M+ Counter High+Counter Low)

Related Information

Stratix Device Handbook

7.8.3.3. Adjusting the Output Clock Frequency

Use the equation to adjust the PLL output clock in normal mode.

$$\text{Output Clock Frequency} = \text{Input Frequency} \bullet \frac{\text{M Value}}{\text{N Value} + \text{Counter High} + \text{Counter Low}}$$

Use the equation to adjust the PLL output clock in external feedback mode.

$$\text{OUTCLK} = \frac{\text{M Value} + \text{External Feedback Counter High} + \text{External Feedback Counter Low}}{\text{N Value} + \text{Counter High} + \text{Counter Low}}$$

7.8.3.4. Adjusting the Spread Spectrum

Use the equation to adjust the spread spectrum for your PLL.

$$\%\ \text{Spread} = \frac{M_2 N_1}{M_1 N_2}$$

7.8.4. Modify the Connectivity between Resource Atoms

The Chip Planner and Resource Property Editor allow you to create new resource atoms and manipulate the existing connection between resource atoms in the post-fit netlist. These features are useful for small changes when you are debugging a design, such as manually inserting pipeline registers into a combinational path that fails timing, or routing a signal to a spare I/O pin for analysis.

Use the following procedure to create a new register in a Cyclone V device and route register output to a spare I/O pin. This example illustrates how to create a new resource atom and modify the connections between resource atoms.

To create new resource atoms and manipulate the existing connection between resource atoms in the post-fit netlist, follow these steps:

1. Create a new register in the Chip Planner.
2. Locate the atom in the Resource Property Editor.
3. To assign a clock signal to the register, right-click the clock input port for the register, point to **Edit connection,** and click **Other**. Use the Node Finder to assign a clock signal from your design.
4. To tie the SLOAD input port to V_{CC}, right-click the clock input port for the register, point to **Edit connection**, and click **VCC**.
5. Assign a data signal from your design to the SDATA port.
6. In the Connectivity window, under the output port names, copy the port name of the register.
7. In the Chip Planner, locate a free I/O resource and create an output buffer.
8. Locate the new I/O atom in the Resource Property Editor.

9. On the input port to the output buffer, right-click, point to **Edit connection**, and click **Other**.
10. In the **Edit Connection** dialog box, type the output port name of the register you have created.
11. Run the ECO Fitter to apply the changes by clicking **Check and Save Netlist Changes**.

    Note: A successful ECO connection is subject to the available routing resources. You can view the relative routing utilization by selecting **Routing Utilization** as the Background Color Map in the **Layers Settings** dialog box of the Chip Planner. Also, you can view individual routing channel utilization from local, row, and column interconnects with the tooltips created when you position your mouse pointer over the appropriate resource. Refer to the device data sheet for more information about the architecture of the routing interconnects of your device.

7.9. Post ECO Steps

After you make an ECO change with the Chip Planner, you must perform static timing analysis of your design with the Timing Analyzer to ensure that your changes did not adversely affect the timing performance of your design.

For example, when you turn on one of the delay chain settings for a specific pin, you change the I/O timing. Therefore, to ensure that the design still meets all timing requirements, you should perform static timing analysis.

Related Information

Intel Quartus Prime Timing Analyzer User Guide
For more information about performing a static timing analysis of your design

7.10. Engineering Change Orders with the Chip Planner Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2018.05.07	18.0.0	Added statement indicating no Chip Planner ECO support for Intel Arria 10 devices.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
June 2014	14.0.0	• Updated formatting. • Removed references to Stratix, Stratix II, Stratix III, Arria GX, Arria II GX, Cyclone, Cyclone II, Cyclone III, and MAX II devices. • Added MAX V, Cyclone V, Arria V I/O elements
June 2012	12.0.0	Removed survey link.
November 2011	10.1.1	Template update.
December 2010	10.1.0	• Updated chapter to new template • Removed "The Chip Planner FloorPlan Views" section • Combined "Creating Atoms", "Deleting Atoms", and "Moving Atoms" sections, and linked to Help. • Added Stratix V I/O elements in "FPGA I/O Elements".

continued...

Document Version	Intel Quartus Prime Version	Changes
July 2010	10.0.0	• Added information to page 17–1. • Added information to "Engineering Change Orders" on page 17–2. • Changed heading from "Performance" to "Performance Preservation" on page 7–2. • Updated information in "Performance Preservation" on page 17–2. • Changed heading from "Documentation" to "Change Modification Record" on page 17–3. • Changed heading from "Resource Property Editor" to "Performing ECOs in the Resource Property Editor" on page 17–15. • Removed "Using Incremental Compilation in the ECO Flow" section. Preservation support for ECOs with the incremental compilation flow has been removed in the Quartus II software version 10.0. • Removed "Referenced Documents" section.
November 2009	9.1.0	• Updated device support list • Made minor editorial updates
March 2009	9.0.0	• Updated Figure 17–17. • Made minor editorial updates. • Chapter 15 was previously Chapter 13 in the 8.1.0 release.
November 2008	8.1.0	• Corrected preservation attributes for ECOs in the section "Using Incremental Compilation in the ECO Flow" on page15–32. • Minor editorial updates. • Changed to 8½" x 11" page size.
May 2008	8.0.0	• Updated device support list • Modified description for ECO support for block RAMs and DSP blocks • Corrected Stratix PLL ECO example • Added an application example to show modifying the connectivity between resource atoms

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Programmer

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Programming Intel FPGA Devices..... 3
1.1. Programming Flow.....3
1.1.1. Stand-Alone Intel Quartus Prime Programmer.....3
1.1.2. Optional Programming or Configuration Files..... 4
1.1.3. Secondary Programming Files.....5
1.2. Intel Quartus Prime Programmer Window.....5
1.2.1. Editing the Details of an Unknown Device.....6
1.2.2. Setting Up the Hardware..... 6
1.2.3. Setting the JTAG Hardware..... 6
1.2.4. Using the JTAG Chain Debugger Tool..... 7
1.3. Programming and Configuration Modes.....7
1.4. Design Security Keys..... 7
1.5. Verifying if Programming Files Correspond to a Compilation of the Same Source Files..... 7
1.5.1. Obtaining Project Hash for Arria V, Stratix V, Cyclone V and Intel MAX 10 Devices..... 8
1.5.2. Obtaining Project Hash for Intel Arria 10 Devices..... 8
1.6. Convert Programming Files Dialog Box..... 9
1.6.1. Debugging Your Configuration.....10
1.6.2. Converting Programming Files for Partial Reconfiguration..... 12
1.7. Flash Loaders.....14
1.8. JTAG Debug Mode for Partial Reconfiguration..... 14
1.8.1. Configuring Partial Reconfiguration Bitstream in JTAG Debug Mode..... 15
1.9. Scripting Support.....21
1.9.1. The jtagconfig Debugging Tool..... 21
1.9.2. Generating a Partial-Mask SRAM Object File using a Mask Settings File and a SRAM Object File.....22
1.9.3. Generating Raw Binary File for Partial Reconfiguration using a .pmsf..... 22
1.10. Programming Intel FPGA Devices Revision History..... 22

A. Intel Quartus Prime Standard Edition User Guides.....24

1. Programming Intel FPGA Devices

The Intel® Quartus® Prime Programmer allows you to program and configure Intel FPGA CPLD, FPGA, and configuration devices. After compiling your design, use the Intel Quartus Prime Programmer to program or configure your device, to test the functionality of the design on a circuit board.

1.1. Programming Flow

In the FPGA flow, device programming requires a fully compiled design that includes the programming or configuration files that the Assembler generates.

To program a device:

1. Convert the programming or configuration file to target the configuration device and, optionally, create secondary programming files.

Table 1. Programming and Configuration File Format

File Format	FPGA	CPLD	Configuration Device	Serial Configuration Device
SRAM Object File (`.sof`)	Yes	—	—	—
Programmer Object File (`.pof`)	—	Yes	Yes	Yes
JEDEC JESD71 STAPL Format File (`.jam`)	Yes	Yes	Yes	—
Jam Byte Code File (`.jbc`)	Yes	Yes	Yes	—

2. In the Intel Quartus Prime Programmer, program and configure the FPGA, CPLD, or configuration device with the appropriate programming or configuration files.

The FPGA now contains the design that you specified in the Intel Quartus Prime project.

1.1.1. Stand-Alone Intel Quartus Prime Programmer

Intel FPGA offers the free stand-alone Programmer, which has the same full functionality as the Intel Quartus Prime Programmer in the Intel Quartus Prime software. The stand-alone Programmer is useful when programming your devices with another workstation, so you do not need two full licenses. You can download the stand-alone Programmer from the Download Center on the Altera website.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Stand-Alone Programmer Memory Limitations

The stand-alone Programmer may use significant memory during the following operations:

- During auto-detect operations
- When the programming file is added to the flash
- During manual attachment of the flash into the Programmer window

The 32-bit stand-alone Programmer can only use a limited amount of memory when launched in 32-bit Windows. Note the following specific limitations of 32-bit stand-alone Programmer:

Table 2. Stand-Alone Programmer Memory Limitations

Application	Maximum Flash Device Size	Flash Device Operation Using PFL
32-bit Stand-Alone Programmer	Up to 512 Mb	Single Flash Device
64-bit Stand-Alone Programmer	Up to 2 Gb	Multiple Flash Device

The stand-alone Programmer supports combination and/or conversion of Intel Quartus Prime programming files using the **Convert Programming Files** dialog box. You can convert programming files, such as Mask Settings File (`.msf`), Partial-Mask SRAM Object File (`.pmsf`), SRAM Object Files (`.sof`), or Programmer Object Files (`.pof`) into other file formats that support device configuration schemes for Intel FPGA devices.

Note the following device-specific file conversion limitations with use of the 32-bit stand-alone Programmer:

Table 3. Stand-Alone Programmer File Conversion Limitations

Programming File Conversion	Device Support
32-bit Programming File Conversion	All Supported Intel FPGA Devices Except Intel Arria® 10
64-bit Programming File Conversion	All Supported Intel FPGA Devices

Related Information

Download Center

1.1.2. Optional Programming or Configuration Files

The Intel Quartus Prime software can generate optional programming or configuration files in various formats that you can use with programming tools other than the Intel Quartus Prime Programmer. When you compile a design in the Intel Quartus Prime software, the Assembler automatically generates either a `.sof` or `.pof`. The Assembler also allows you to convert FPGA configuration files to programming files for configuration devices.

Related Information

AN 425: Using Command-Line Jam STAPL Solution for Device Programming

1.1.3. Secondary Programming Files

The Intel Quartus Prime software generates programming files in various formats for use with different programming tools.

Table 4. File Types Generated by the Intel Quartus Prime Software and Supported by the Intel Quartus Prime Programmer

File Type	Generated by the Intel Quartus Prime Software	Supported by the Intel Quartus Prime Programmer
`.sof`	Yes	Yes
`.pof`	Yes	Yes
`.jam`	Yes	Yes
`.jbc`	Yes	Yes
JTAG Indirect Configuration File (`.jic`)	Yes	Yes
Serial Vector Format File (`.svf`)	Yes	—
Hexadecimal (Intel-Format) Output File (`.hexout`)	Yes	—
Raw Binary File (`.rbf`)	Yes	Yes (1)
Raw Binary File for Partial Reconfiguration (`.rbf`)	Yes	Yes (2)
Tabular Text File (`.ttf`)	Yes	—
Raw Programming Data File (`.rpd`)	Yes	—

1.2. Intel Quartus Prime Programmer Window

The Intel Quartus Prime **Programmer** window allows you to:

- Add your programming and configuration files.
- Specify programming options and hardware.
- Start the programming or configuration of the device.

To open the **Programmer** window, click **Tools ➤ Programmer**. As you proceed through the programming flow, the Intel Quartus Prime **Message** window reports the status of each operation.

Related Information

Programmer Page (Options Dialog Box)
In Intel Quartus Prime Help

(1) Raw Binary File (`.rbf`) is supported by the Intel Quartus Prime Programmer in Passive Serial (PS) configuration mode.

(2) Raw Binary File for Partial Reconfiguration (`.rbf`) is supported by the Intel Quartus Prime Programmer in JTAG debug mode.

1.2.1. Editing the Details of an Unknown Device

When the Intel 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.

1.2.2. Setting Up the Hardware

Before you can program or configure the device, you must have the correct hardware setup. The Intel Quartus Prime Programmer provides the flexibility to choose a download cable or programming hardware.

1.2.3. Setting the JTAG Hardware

The JTAG server allows the Intel Quartus Prime Programmer to access the JTAG hardware. You can also access the JTAG download cable or programming hardware connected to a remote computer through the JTAG server of that computer. With the JTAG server, you can 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.

1.2.3.1. Running JTAG Daemon with Linux

The JTAGD daemon allows a remote machine to program or debug a board that is connected to a Linux host over the network. The JTAGD daemon also allows multiple programs to use JTAG resources at the same time. The JTAGD daemon is the Linux version of a JTAG server.

Run the JTAGD daemon to avoid:

- 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, follow these steps:

1. Create an `/etc/jtagd` directory.
2. Set the permissions of this directory and the files in the directory to allow you to have the read/write access.
3. Run `jtagd` (with no arguments) from your `quartus/bin` directory.

The JTAGD daemon is now running and does not terminate when you log off.

1.2.4. Using the 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. In addition, the tool 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. You access the tool by clicking **Tools ➤ JTAG Chain Debugger** on the Intel Quartus Prime software.

1.3. Programming and Configuration Modes

The following table lists the programming and configuration modes supported by Intel FPGA devices.

Table 5. Programming and Configuration Modes

Configuration Mode Supported by the Intel Quartus Prime Programmer	FPGA	CPLD	Configuration Device	Serial Configuration Device
JTAG	Yes	Yes	Yes	—
Passive Serial (PS)	Yes	—	—	—
Active Serial (AS) Programming	—	—	—	Yes
Configuration via Protocol (CvP)	Yes	—	—	—
In-Socket Programming	—	Yes (except for MAX® II CPLDs)	Yes	Yes

Related Information

Configuration via Protocol (CvP) Implementation in V-series Intel FPGAs Devices User Guide
Describes the CvP configuration mode.

1.4. Design Security Keys

The Intel Quartus Prime Programmer supports the generation of encryption key programming files and encrypted configuration files for Intel FPGAs that support the design security feature. You can also use the Intel Quartus Prime Programmer to program the encryption key into the FPGA.

Related Information

AN 556: Using the Design Security Features in Intel FPGAs

1.5. Verifying if Programming Files Correspond to a Compilation of the Same Source Files

The project hash feature allows you to verify if two programming files correspond to a compilation of the same set of source files. During compilation, the Intel Quartus Prime software generates a unique project hash and embeds this value in programming files (`.sof`). The project hash is available for Arria V, Stratix® V, Cyclone® V, Intel MAX 10, and Intel Arria 10 device families.

The project hash doesn't change for different builds of the Intel Quartus Prime software, or when you install a software patch. However, if you upgrade any IP with a different build or patch, the project hash changes.

1.5.1. Obtaining Project Hash for Arria V, Stratix V, Cyclone V and Intel MAX 10 Devices

To obtain the project hash value of a `.sof` programming file for a design targeted to Arria V, Stratix V, Cyclone V, and Intel MAX 10 devices, use the following command, which dumps out metadata information that includes the project hash.

```
quartus_cpf --info <sof-file-name>
```

Example 1. Output of Project Hash Extraction

In this example, the programming file name is `cb_intosc.sof`.

```
File: cb_intosc.sof
        File CRC: 0x0000
        Creator: Quartus Prime Compiler
        Version 17.0.0 Internal Build 565 02/09/2017 SJ Standard Edition
        Comment: UNIX
        Device: 5SGSMD5K2F40
        Data checksum: 0x02534E5A
        JTAG usercode: 0x02534E5A
        Project Hash:  0x556e737065636966696564
```

1.5.2. Obtaining Project Hash for Intel Arria 10 Devices

To obtain the project hash value of a `.sof` programming file for a design targeted to Intel Arria 10 devices, create a file named `project_hash.tcl`. In your file, copy and paste the following code:

```
###############################################
## Begin project_hash.tcl
#
##
## @copyright Intel 2017
##
proc main_run {} {
global quartus
set qargs $quartus(args)
set nargs [llength $qargs]
load_package asm2
load_devices
set handle -1
set sof_file [lindex $qargs 0]
if [file exists $sof_file] {
set handle [open_sof $sof_file]
}
print "/metadata/0/project_hash"
if { $handle != -1 } {
close_handle $handle
}
}
###############################################
main_run
## End of project_hash.tcl
###############################################
```

Save the `project_hash.tcl` file in the same directory that contains your programming file, and type in the command line:

```
quartus_asm -t project_hash.tcl <sof-file>
```

The script prints the project hash value to the command line output.

Example 2. Output of Project Hash Extraction:

In this example, the programming file is `one_and.sof`.

```
Info: *******************************************************************
Info: Running Quartus Prime Assembler
Info: Version 17.0.0 Build 594 04/18/2017 SJ Standard 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: Sat Apr 22 00:44:19 2017
Info: Command: quartus_asm -t project_hash.tcl one_and.sof
Info: Quartus(args): one_and.sof
Info: Using INI file /data/msandova/qmap/quartus.ini
0x16cc7e6773644d398b740451aa0b26e3
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: 1111 megabytes
Info: Processing ended: Sat Apr 22 00:44:27 2017
Info: Elapsed time: 00:00:08
```

1.6. Convert Programming Files Dialog Box

The **Convert Programming Files** dialog box in the Programmer allows you to convert programming files from one file format to another. To access the **Convert Programming Files** dialog box, click **File ➤ Convert Programming Files...** on the Intel Quartus Prime software.

For example, to store the FPGA data in configuration devices, you can convert the `.sof` data to another format, such as `.pof`, `.hexout`, `.rbf`, `.rpd`, or `.jic`, and then program the configuration device.

You can also configure multiple devices with an external host, such as a microprocessor or CPLD. For example, you can combine multiple `.sof` files into one `.pof` file. To save time in subsequent conversions, click **Save Conversion Setup** to save your conversion specifications in a Conversion Setup File (`.cof`). Click **Open Conversion Setup Data** to load your `.cof` setup in the **Convert Programming Files** dialog box.

Example 3. Conversion Setup File Contents

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>
<cof>
    <output_filename>output_file.pof</output_filename>
```

```
    <n_pages>1</n_pages>
    <width>1</width>
    <mode>14</mode>
    <sof_data>
        <user_name>Page_0</user_name>
        <page_flags>1</page_flags>
        <bit0>
            <sof_filename>/users/jbrossar/template/output_files/
template_test.sof</sof_filename>
        </bit0>
    </sof_data>
    <version>7</version>
    <create_cvp_file>0</create_cvp_file>
    <create_hps_iocsr>0</create_hps_iocsr>
    <auto_create_rpd>0</auto_create_rpd>
    <options>
        <map_file>1</map_file>
    </options>
    <MAX10_device_options>
        <por>0</por>
        <io_pullup>1</io_pullup>
        <auto_reconfigure>1</auto_reconfigure>
        <isp_source>0</isp_source>
        <verify_protect>0</verify_protect>
        <epof>0</epof>
        <ufm_source>0</ufm_source>
    </MAX10_device_options>
    <advanced_options>
        <ignore_epcs_id_check>0</ignore_epcs_id_check>
        <ignore_condone_check>2</ignore_condone_check>
        <plc_adjustment>0</plc_adjustment>
        <post_chain_bitstream_pad_bytes>-1</post_chain_bitstream_pad_bytes>
        <post_device_bitstream_pad_bytes>-1</post_device_bitstream_pad_bytes>
        <bitslice_pre_padding>1</bitslice_pre_padding>
    </advanced_options>
</cof>
```

Related Information

Convert Programming Files Dialog Box
In Intel Quartus Prime Help

1.6.1. Debugging Your Configuration

Use the **Advanced** option in the **Convert Programming Files** dialog box to debug your configuration. You must choose the advanced settings that apply to your Intel FPGA device. You can direct the Intel Quartus Prime software to enable or disable an advanced option by turning the option on or off in the **Advanced Options** dialog box. When you change settings in the **Advanced Options** dialog box, the change affects `.pof`, `.jic`, `.rpd`, and `.rbf` files.

The following table lists the **Advanced Options** settings in more detail:

Table 6. Advanced Options Settings

Option Setting	Description
Disable EPCS ID check	FPGA skips the EPCS silicon ID verification. Default setting is unavailable (EPCS ID check is enabled). Applies to the single- and multi-device AS configuration modes on all FPGA devices.
Disable AS mode CONF_DONE error check	FPGA skips the CONF_DONE error check.

continued...

Option Setting	Description
	Default setting is unavailable (AS mode CONF_DONE error check is enabled). Applies to single- and multi-device (AS) configuration modes on all FPGA devices. The CONF_DONE error check is disabled by default for Stratix V, Arria V, and Cyclone V devices for AS-PS multi device configuration mode.
Program Length Count adjustment	Specifies the offset you can apply to the computed PLC of the entire bitstream. Default setting is 0. The value must be an integer. Applies to single- and multi-device (AS) configuration modes on all FPGA devices.
Post-chain bitstream pad bytes	Specifies the number of pad bytes appended to the end of an entire bitstream. Default value is set to 0 if the bitstream of the last device is uncompressed. Set to 2 if the bitstream of the last device is compressed.
Post-device bitstream pad bytes	Specifies the number of pad bytes appended to the end of the bitstream of a device. Default value is 0. No negative integer. Applies to all single-device configuration modes on all FPGA devices.
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. Default value is 1. Valid setting is 0 or 1. 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.

The following table lists the symptoms you may encounter if a configuration fails, and describes the advanced options you must use to debug your configuration.

Failure Symptoms	Disable EPCS 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 (3)	Yes (4)	—
Decompression feature is enabled.	—	Yes	Yes	Yes (3)	Yes (4)	—
Encryption feature is enabled.	—	Yes	Yes	Yes (3)	Yes (4)	—

continued...

(3) Use only for multi-device chain

(4) Use only for single-device chain

Failure Symptoms	Disable EPCS ID Check	Disable AS Mode CONF_DONE Error Check	PLC Settings	Post-Chain Bitstream Pad Bytes	Post-Device Bitstream Pad Bytes	Bitslice Padding Value
CONF_DONE stays low after a configuration cycle.	—	Yes	Yes (5)	Yes (3)	Yes (4)	—
CONF_DONE goes high momentarily after a configuration cycle.	—	Yes	Yes (6)	—	—	—
FPGA does not enter user mode even though CONF_DONE goes high.	—	—	—	Yes (3)	Yes (4)	—
Configuration failure occurs at the beginning of a configuration cycle.	Yes	—	—	—	—	—
Newly introduced EPCS, such as EPCS128.	Yes	—	—	—	—	—
Failure in .pof generation for EPC device using Intel Quartus Prime Convert Programming File Utility when the decompression feature is enabled.	—	—	—	—	—	Yes

1.6.2. Converting Programming Files for Partial Reconfiguration

The **Convert Programming File** dialog box supports the following programming file generation and option for Partial Reconfiguration:

- Partial-Masked SRAM Object File (.pmsf) output file generation, with .msf and .sof as input files.
- .rbf for Partial Reconfiguration output file generation, with a .pmsf as the input file.

  Note: The .rbf for Partial Reconfiguration file is only for Partial Reconfiguration.
- Providing the **Enable decompression during Partial Reconfiguration** option to enable the option bit for bitstream decompression during Partial Reconfiguration, when converting a full design .sof to any supported file type.

Related Information

- Design Planning for Partial Reconfiguration
  In *Intel Quartus Prime Standard Edition User Guide: Partial Reconfiguration*

(5) Start with positive offset to the PLC settings

(6) Start with negative offset to the PLC settings

- Design Planning for Partial Reconfiguration

1.6.2.1. Generating .pmsf using a .msf and a .sof

To generate the `.pmsf` in the **Convert Programming Files** dialog box:

1. In the **Convert Programming Files** dialog box, under the **Programming file type** field, select **Partial-Masked SRAM Object File (.pmsf)**.
2. In **File name**, specify the necessary output file name.
3. In the **Input files to convert** field, add necessary input files to convert. You can add only a `.msf` and `.sof`.
4. Click **Generate**.

1.6.2.2. Generating a .rbf for Partial Reconfiguration from a .pmsf file

After generating the `.pmsf` file, you convert the `.pmsf` file into a `.rbf` file with the **Convert Programming Files** dialog box.

To generate the `.rbf` for Partial Reconfiguration:

1. In the **Convert Programming Files** dialog box, in the **Programming file type** field, select **Raw Binary File for Partial Reconfiguration (.rbf)**.
2. In the **File name** field, specify the output file name.
3. In the **Input files to convert** field, add input files to convert.

   You can add only one `.pmsf` file.
4. Select the `.pmsf`, and click **Properties**.
   The **PMSF File Properties** dialog box appears.
5. Make your selection either by turning on or turning off the following options:
   - **Compression option**—This option enables compression on Partial Reconfiguration bitstream. If you turn on this option, then you must turn on the **Enable decompression during Partial Reconfiguration** option.
   - **Enable SCRUB mode option**—The default of this option is based on AND/OR mode. This option is valid only when Partial Reconfiguration masks in your design are not overlapped vertically. Otherwise, you cannot generate the `.rbf` for Partial Reconfiguration.
   - **Write memory contents option**—This option is a workaround for initialized RAM/ROM in a Partial Reconfiguration region.

   For more information about these options refer to *Design Planning for Partial Reconfiguration* in *Intel Quartus Prime Standard Edition User Guide: Partial Reconfiguration*.
6. Click **OK**.
7. Click **Generate**.

Related Information

Design Planning for Partial Reconfiguration
In *Intel Quartus Prime Standard Edition User Guide: Partial Reconfiguration*

1.6.2.3. Enable Decompression During Partial Reconfiguration Option

You can turn on the **Enable decompression during Partial Reconfiguration** option in the **SOF File Properties: Bitstream Encryption** dialog box, which you can access from the **Convert Programming File** dialog box. This option is available when converting a `.sof` to any supported programming file types listed in *Secondary Programming Files*.

This option is hidden for other targeted devices that do not support Partial Reconfiguration. To view this option in the **SOF File Properties: Bitstream Encryption** dialog box, the `.sof` must be targeted on an Intel FPGA device that supports Partial Reconfiguration.

If you turn on the **Compression** option when generating the `.rbf` for Partial Reconfiguration, then you must turn on the **Enable decompression during Partial Reconfiguration** option.

Related Information

Secondary Programming Files on page 5

1.7. Flash Loaders

Parallel and serial configuration devices do not support the JTAG interface. However, you can use a flash loader to program configuration devices in-system via the JTAG interface. You can use an FPGA as a bridge between the JTAG interface and the configuration device. The Intel Quartus Prime software supports parallel and serial flash loaders.

1.8. JTAG Debug Mode for Partial Reconfiguration

The JTAG debug mode allows you to configure partial reconfiguration bitstream through the JTAG interface. Use this feature to debug PR bitstream and eventually helping you in your PR design prototyping. This feature is available for internal and external host. Using the JTAG debug mode forces the Data Source Controller to be in x16 mode.

During JTAG debug operation, the JTAG command sent from the Intel Quartus Prime Programmer ignores and overrides most of the Partial Reconfiguration IP core interface signals (`clk`, `pr_start`, `double_pr`, `data[]`, `data_valid`, and `data_read`).

Note: The TCK is the main clock source for PR IP core during this operation.

You can view the status of Partial Reconfiguration operation in the messages box and the Progress bar in the Intel Quartus Prime Programmer. The PR_DONE, PR_ERROR, and CRC_ERROR signals will be monitored during PR operation and reported in the Messages box at the end of the operation.

The Intel Quartus Prime Programmer can detect the number of PR_DONE instruction(s) in plain or compressed PR bitstream and, therefore, can handle single or double PR cycle accordingly. However, only single PR cycle is supported for encrypted Partial Reconfiguration bitstream in JTAG debug mode (provided that the specified device is configured with the encrypted base bitstream which contains the PR IP core in the design).

Note: Configuring an incompatible PR bitstream to the specified device may corrupt your design, including the routing path and the PR IP core placed in the static region. When this issue occurs, the PR IP core stays in an undefined state, and the Intel Quartus Prime Programmer is unable to reset the IP core. As a result, the Intel Quartus Prime Programmer generates the following error when you try to configure a new PR bitstream:

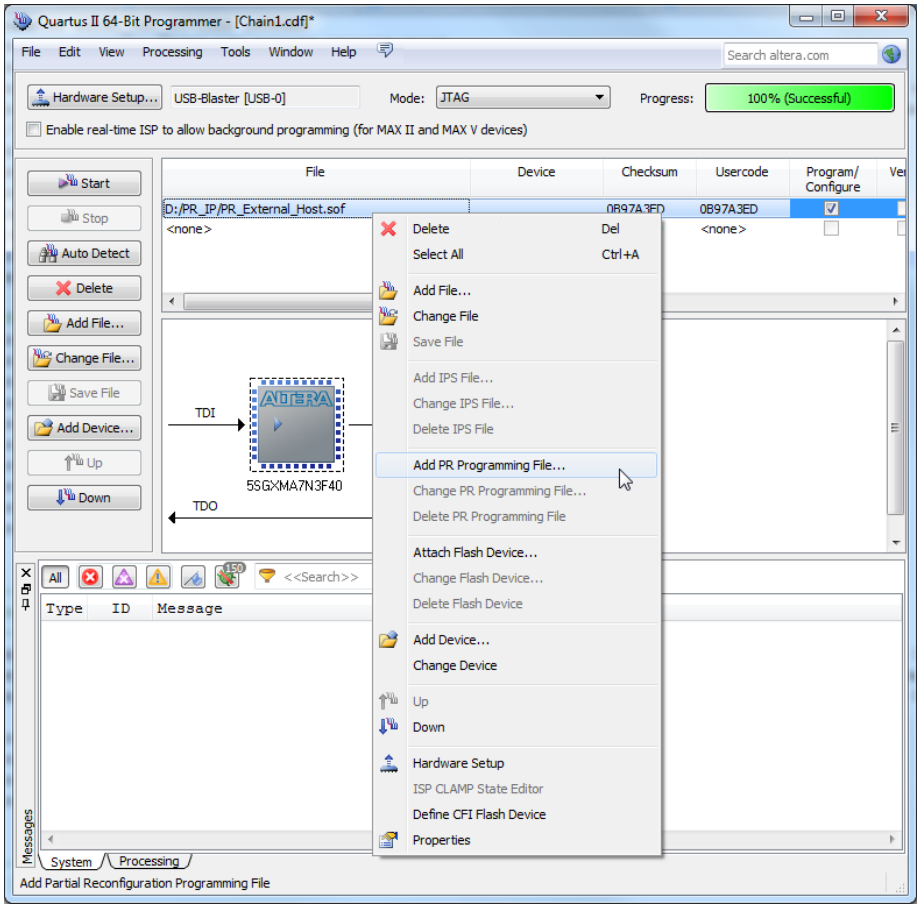
```
Error (12897): Partial Reconfiguration status: Can't reset the PR megafunction.
This issue occurred because the design was corrupted by an incompatible PR
bitstream in the previous PR operation. You must reconfigure the device with a
good design.
```

1.8.1. Configuring Partial Reconfiguration Bitstream in JTAG Debug Mode

To configure the Partial Reconfiguration bitstream in JTAG debug mode, follow these steps:

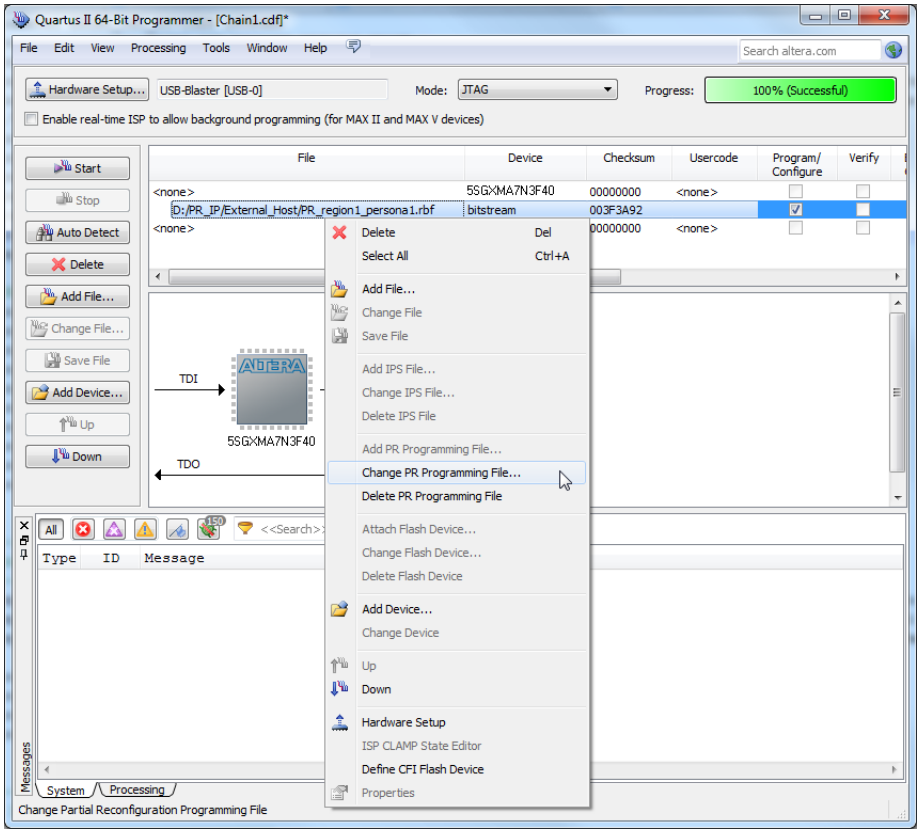
1. In the Intel Quartus Prime Programmer GUI, right click a highlighted base bitstream (in `.sof`) and then click **Add PR Programming File** to add the PR bitstream (`.rbf`).

Figure 1. Adding PR Programming File



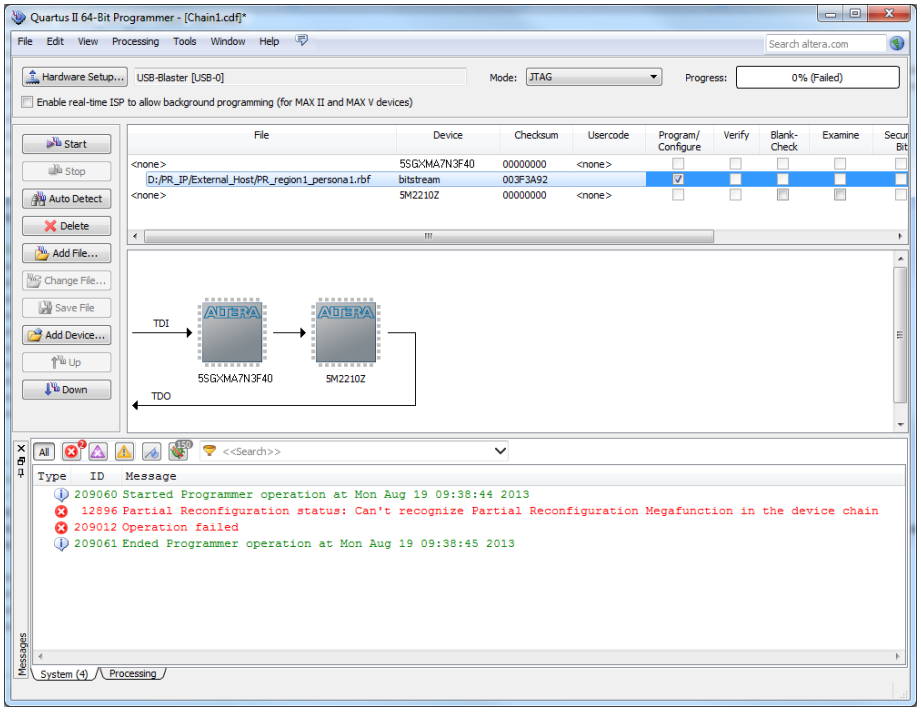
2. After adding the PR bitstream, you can change or delete the Partial Reconfiguration programming file by clicking **Change PR Programming File** or **Delete PR Programming File**.

Figure 2. Change PR Programming File or Delete PR Programming File



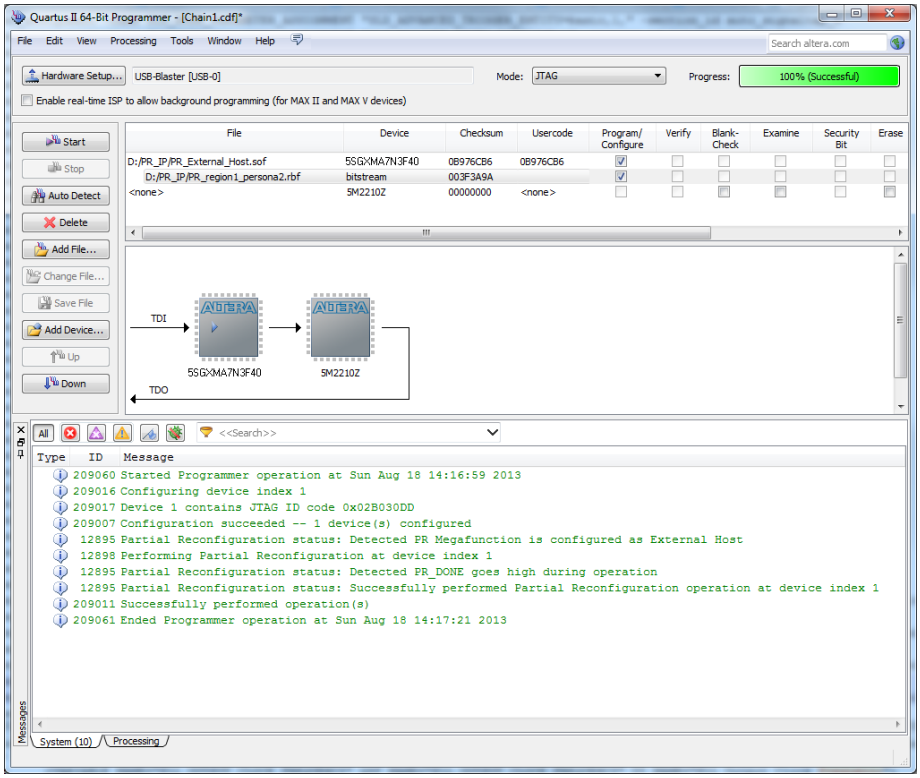
3. Click **Start** to configure the PR bitstream. The Intel Quartus Prime Programmer generates an error message if the specified device does not contain the PR IP core in the design (you must instantiate the Partial Reconfiguration IP core in your design to use the JTAG debug mode).

Figure 3. Starting PR Bitstream Configuration



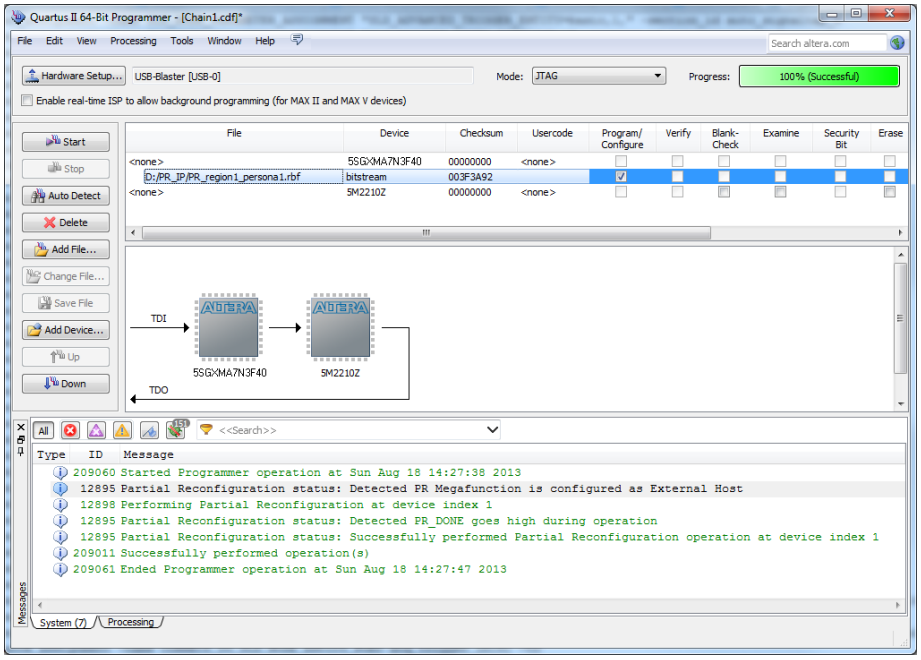
4. Configure the valid `.rbf` in JTAG debug mode with the Intel Quartus Prime Programmer.

Figure 4. Configuring Valid .rbf



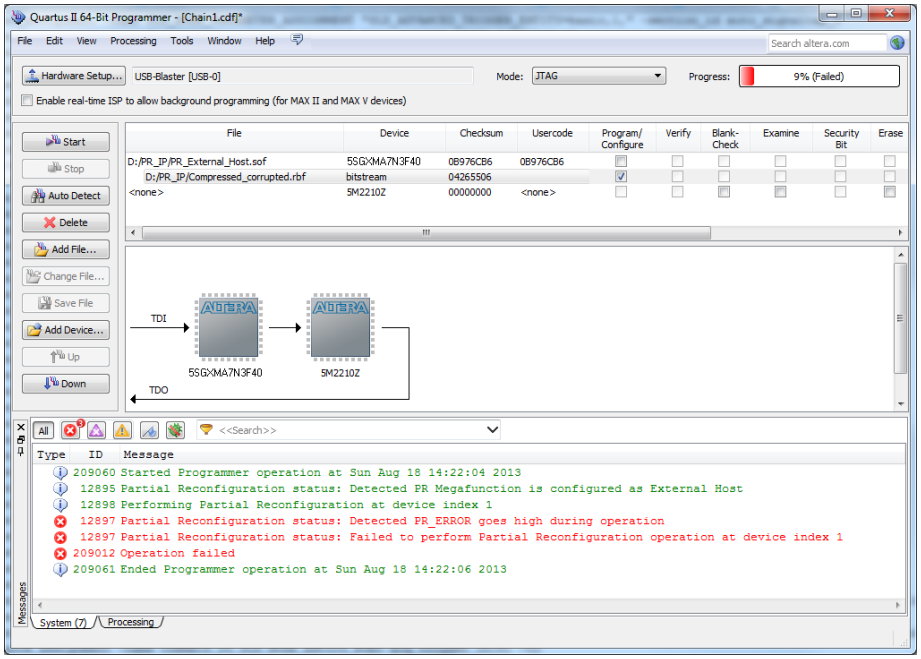
5. The JTAG debug mode is also supported if the PR IP core is pre-programmed on the specified device.

Figure 5. Partial Reconfiguration IP Core Successfully Pre-programmed



6. The Intel Quartus Prime Programmer reports error when you try to configure the corrupted `.rbf` in JTAG debug mode.

Figure 6. Configuring Corrupted .rbf



1.9. Scripting Support

In addition to the Intel Quartus Prime Programmer GUI, you can use the Intel Quartus Prime command-line executable `quartus_pgm.exe` (or `quartus_pgm` in Linux) to access programmer functionality from the command line and from scripts. The programmer accepts `.pof`, `.sof`, and `.jic` programming or configuration files and `.cdf` files.

The following example shows a command that programs a device:

```
quartus_pgm –c byteblasterII –m jtag –o bpv;design.pof ←
```

Where:

- `-c byteblasterII` specifies the Intel FPGA Intel FPGA Parallel Port Cable download cable
- `-m jtag` specifies the JTAG programming mode
- `-o bpv` represents the blank-check, program, and verify operations
- `design.pof` represents the `.pof` used for the programming

The Programmer automatically executes the erase operation before programming the device.

For Linux terminal, use:

```
quartus_pgm –c byteblasterII –m jtag –o bpv\;design.pof
```

Related Information

Intel Quartus Prime Scripting
In Intel Quartus Prime Help

1.9.1. 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 Intel 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 *Intel Quartus Prime Standard Edition User Guide: Scripting*

1.9.2. Generating a Partial-Mask SRAM Object File using a Mask Settings File and a SRAM Object File

You can generate a `.pmsf` with the `quartus_cpf` command by typing the following command:

```
quartus_cpf -p <pr_revision.msf> <pr_revision.sof> <new_filename.pmsf>
```

1.9.3. Generating Raw Binary File for Partial Reconfiguration using a .pmsf

You can generate a `.rbf` for Partial Reconfiguration with the `quartus_cpf` command by typing the following command:

```
quartus_cpf –o foo.txt –c <pr_revision.pmsf> <pr_revision.rbf>
```

Note: You must run this command in the same directory where the files are located.

1.10. Programming Intel FPGA Devices Revision History

Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • Renamed topic: *Project Hash* to *Verifying if Programming Files Correspond to a Compilation of the Same Source Files*.
2017.05.08	17.0.0	• Added Project Hash feature.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Quartus Prime*.
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.

continued...

Document Version	Intel Quartus Prime Version	Changes
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.
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.

Related Information

Altera Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the Altera documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Partial Reconfiguration

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Design Planning for Partial Reconfiguration....3
1.1. Terminology....3
1.1.1. Determining Resources for Partial Reconfiguration....5
1.2. An Example of a Partial Reconfiguration Design....6
1.3. Partial Reconfiguration Modes....6
1.3.1. SCRUB Mode....7
1.3.2. AND/OR Mode....8
1.3.3. Programming File Sizes for a Partial Reconfiguration Project....9
1.4. Partial Reconfiguration Design Flow....10
1.4.1. Design Partitions for Partial Reconfiguration....12
1.4.2. Incremental Compilation Partitions for Partial Reconfiguration....12
1.4.3. Partial Reconfiguration Controller Instantiation in the Design....13
1.4.4. Wrapper Logic for PR Regions....16
1.5. Freeze Logic for PR Regions....18
1.5.1. Clocks and Other Global Signals for a PR Design....20
1.5.2. Floorplan Assignments for PR Designs....21
1.6. Implementation Details for Partial Reconfiguration....22
1.6.1. Interface with the PR Control Block through a PR Host....22
1.6.2. Partial Reconfiguration Pins....23
1.6.3. PR Control Signals Interface....24
1.6.4. Reconfiguring a PR Region....25
1.6.5. Partial Reconfiguration Cycle Waveform....27
1.7. Example of a Partial Reconfiguration Design with an External Host....29
1.7.1. Example of Using an External Host with Multiple Devices....29
1.8. Example Partial Reconfiguration with an Internal Host....30
1.9. Partial Reconfiguration Project Management....31
1.9.1. Create Reconfigurable Revisions....31
1.9.2. Compiling Reconfigurable Revisions....32
1.9.3. Timing Closure for a Partial Reconfiguration Project....32
1.9.4. PR Bitstream Compression and Encryption (Intel Arria® 10 Designs)....32
1.10. Programming Files for a Partial Reconfiguration Project....33
1.10.1. Generating Required Programming Files....36
1.10.2. Generate PR Programming Files with the Convert Programming Files Dialog Box....36
1.11. On-Chip Debug for PR Designs....39
1.12. Partial Reconfiguration Known Limitations....40
1.12.1. Memory Blocks Initialization Requirement for PR Designs....40
1.12.2. M20K RAM Blocks in PR Designs....40
1.12.3. MLAB Blocks in PR designs....42
1.12.4. Implementing Memories with Initialized Content....43
1.12.5. Initializing M20K Blocks with a Double PR Cycle....45
1.13. Document Revision History....45

A. Intel Quartus Prime Standard Edition User Guides....46

1. Design Planning for Partial Reconfiguration

The Partial Reconfiguration (PR) feature in the Intel® Quartus® Prime software allows you to reconfigure a portion of the FPGA dynamically, while the remainder of the device continues to operate.

This chapter assumes a basic knowledge of Altera's FPGA design flow, incremental compilation, and LogicLock™ region features available in the Intel Quartus Prime software. It also assumes knowledge of the internal FPGA resources such as logic array blocks (LABs), memory logic array blocks (MLABs), memory types (RAM and ROM), DSP blocks, clock networks.

The Intel Quartus Prime software supports the PR feature for the Intel Stratix® V device family and Cyclone® V devices whose part number ends in "SC", for example, 5CGXFC9E6F35I8NSC.

Related Information

- Terminology on page 3
- An Example of a Partial Reconfiguration Design on page 6
- Partial Reconfiguration Design Flow on page 10
- Implementation Details for Partial Reconfiguration on page 22
- Example of a Partial Reconfiguration Design with an External Host on page 29
- Example Partial Reconfiguration with an Internal Host on page 30
- Partial Reconfiguration Project Management on page 31
- Programming Files for a Partial Reconfiguration Project on page 33
- Partial Reconfiguration Known Limitations on page 40
- mySupport

1.1. Terminology

The following terms are commonly used in this chapter.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- **project:** A Intel Quartus Prime project contains the design files, settings, and constraints files required for the compilation of your design.
- **revision:** In the Intel Quartus Prime software, a revision is a set of assignments and settings for one version of your design. A Intel Quartus Prime project can have several revisions, and each revision has its own set of assignments and settings. A revision helps you to organize several versions of your design into a single project.
- **incremental compilation:** This is a feature of the Intel Quartus Prime software that allows you to preserve results of previous compilations of unchanged parts of the design, while changing the implementation of the parts of your design that you have modified since your previous compilation of the project. The key benefits include timing preservation and compile time reduction by only compiling the logic that has changed.
- **partition:** You can partition your design along logical hierarchical boundaries. Each design partition is independently synthesized and then merged into a complete netlist for further stages of compilation. With the Intel Quartus Prime incremental compilation flow, you can preserve results of unchanged partitions at specific preservation levels. For example, you can set the preservation levels at post-synthesis or post-fit, for iterative compilations in which some part of the design is changed. A partition is only a logical partition of the design, and does not necessarily refer to a physical location on the device. However, you may associate a partition with a specific area of the FPGA by using a floorplan assignment.

  For more information on design partitions, refer to the *Best Practices for Incremental Compilation Partitions and Floorplan Assignments* chapter in the *Intel Quartus Prime Handbook*.
- **LogicLock region:** A LogicLock region constrains the placement of logic in your design. You can associate a design partition with a LogicLock region to constrain the placement of the logic in the partition to a specific physical area of the FPGA.

  For more information about LogicLock regions, refer to the *Analyzing and Optimizing the Design Floorplan* chapter in the *Intel Quartus Prime Handbook* Volume 2.
- **PR project:** Any Intel Quartus Prime design project that uses the PR feature.
- **PR region:** A design partition with an associated contiguous LogicLock region in a PR project. A PR project can have one or more PR regions that can be partially reconfigured independently. A PR region may also be referred to as a PR partition.
- **static region:** The region outside of all the PR regions in a PR project that cannot be reprogrammed with partial reconfiguration (unless you reprogram the entire FPGA). This region is called the static region, or fixed region.
- **persona:** A PR region has multiple implementations. Each implementation is called a persona. PR regions can have multiple personas. In contrast, static regions have a single implementation or persona.
- **PR control block:** Dedicated block in the FPGA that processes the PR requests, handshake protocols, and verifies the CRC.
- **PR IP Core:** Altera soft IP that can be used to configure the PR control block in the FPGA to mange the PR bitstream source.

Related Information

Analyzing and Optimizing the Design Floorplan

1.1.1. Determining Resources for Partial Reconfiguration

You can use partial reconfiguration to configure only the resources such as LABs, embedded memory blocks, and DSP blocks in the FPGA core fabric that are controlled by configuration RAM (CRAM).

The functions in the periphery, such as GPIOs or I/O Registers, are controlled by I/O configuration bits and therefore cannot be partially reconfigured. Clock multiplexers for GCLK and QCLK are also not partially reconfigurable because they are controlled by I/O periphery bits.

Figure 1. Partially Reconfigurable Resources

These are the types of resource blocks in a Stratix V device.

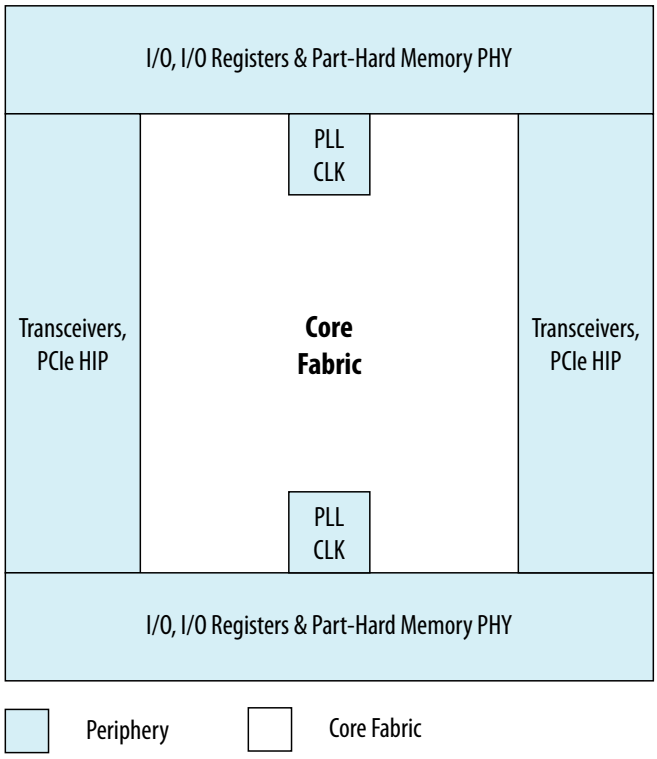


Table 1. Reconfiguration Modes of the FPGA Resource Block

The following table describes the reconfiguration type supported by each FPGA resource block, which are shown in the figure.

Hardware Resource Block	Reconfiguration Mode
Logic Block	Partial Reconfiguration
Digital Signal Processing	Partial Reconfiguration
Memory Block	Partial Reconfiguration
Transceivers	Dynamic Reconfiguration ALTGX_Reconfig
PLL	Dynamic Reconfiguration ALTGX_Reconfig

continued...

Hardware Resource Block	Reconfiguration Mode
Core Routing	Partial Reconfiguration
Clock Networks	Clock network sources cannot be changed, but a PLL driving a clock network can be dynamically reconfigured
I/O Blocks and Other Periphery	Not supported

The transceivers and PLLs in Altera FPGAs can be reconfigured using dynamic reconfiguration. For more information on dynamic reconfiguration, refer to the *Dynamic Reconfiguration in Stratix V Devices* chapter in the *Stratix V Handbook*.

Related Information

Dynamic Reconfiguration in Stratix V Devices

1.2. An Example of a Partial Reconfiguration Design

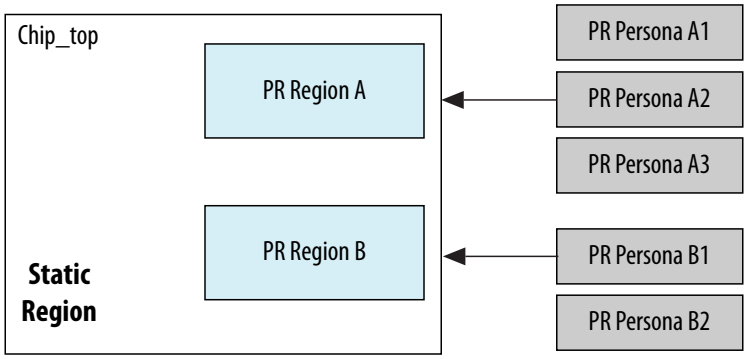
A PR design is divided into two parts. The static region where the design logic does not change, and one or more PR regions.

Each PR region can have different design personas, that change with partial reconfiguration.

PR Region A has three personas associated with it; A1, A2, and A3. PR Region B has two personas; B1 and B2. Each persona for the two PR regions can implement different application specific logic, and using partial reconfiguration, the persona for each PR region can be modified without interrupting the operation of the device in the static or other PR region.

Figure 2. Partial Reconfiguration Project Structure

The following figure shows the top-level of a PR design, which includes a static region and two PR regions.



1.3. Partial Reconfiguration Modes

When you implement a design on an Altera FPGA device, your design implementation is controlled by bits stored in CRAM inside the FPGA.

You can use partial reconfiguration in the SCRUB mode or the AND/OR mode. The mode you select affects your PR flow in ways detailed later in this chapter.

The CRAM bits control individual LABs, MLABs, M20K memory blocks, DSP blocks, and routing multiplexers in a design. The CRAM bits are organized into a frame structure representing vertical areas that correspond to specific locations on the FPGA. If you change a design and reconfigure the FPGA in a non-PR flow, the process reloads all the CRAM bits to a new functionality.

Configuration bitstreams used in a non-PR flow are different than those used in a PR flow. In addition to standard data and CRC check bits, configuration bitstreams for partial reconfiguration also include instructions that direct the PR control block to process the data for partial reconfiguration.

The configuration bitstream written into the CRAM is organized into configuration frames. If a LAB column passes through multiple PR regions, those regions share some programming frames.

1.3.1. SCRUB Mode

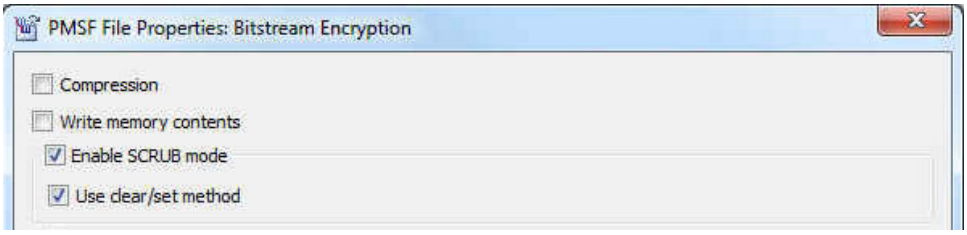
When using SCRUB mode in partial reconfiguration, the unchanging CRAM bits from the static region are "scrubbed" back to their original values.

The static regions controlled by the CRAM bits from the same programming frame as the PR region continue to operate. All the CRAM bits corresponding to a PR region are overwritten with new data, regardless of what was previously contained in the region.

The SCRUB mode of partial reconfiguration involves re-writing all the bits in an entire LAB column of the CRAM, including bits controlling any part of the static region above and below the PR region boundary being reconfigured. You can choose to scrub the values of the CRAM bits to 0, and then rewrite them by turning on the **Use clear/set method** along with **Enable SCRUB mode**. The **Use clear/set method** is the more reliable option, but can increase the size of your bitstream. You can also choose to simply **Enable SCRUB mode**.

Note: You must turn on **Enable SCRUB mode** to use **Use clear/set method**.

Figure 3. Enable SCRUB mode and Use clear/set method

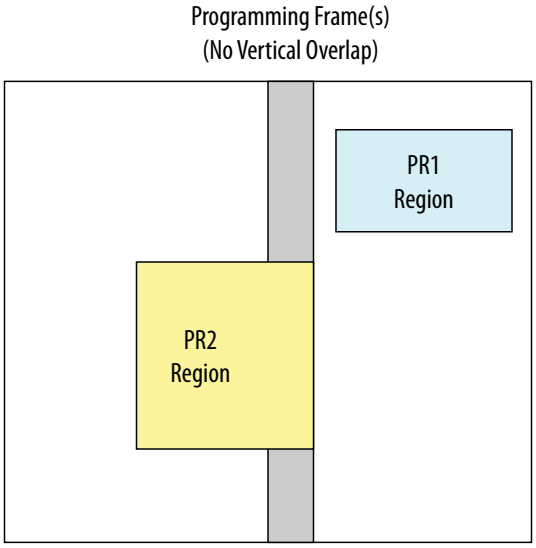


If there are more than one PR regions along a LAB column, and you are trying to reconfigure one of the PR regions, it is not not possible to correctly determine the bits associated with the PR region that is not changing. For this reason, you can not use the SCRUB mode when you have two PR regions that have a vertically overlapping column in the device. This restriction does not apply to the static bits because they never change and you can rewrite them with the same value of the configuration bit.

If you turn on **Enable SCRUB** mode and do not turn on **Use clear/set method**, then the scrub is done in a single pass, writing new values of the CRAM without clearing all the bits first. The advantage of using the SCRUB mode is that the programming file size is much smaller than the AND/OR mode.

Figure 4. SCRUB Mode

This is the floorplan of a FPGA using SCRUB mode, with two PR regions, whose columns do not overlap.



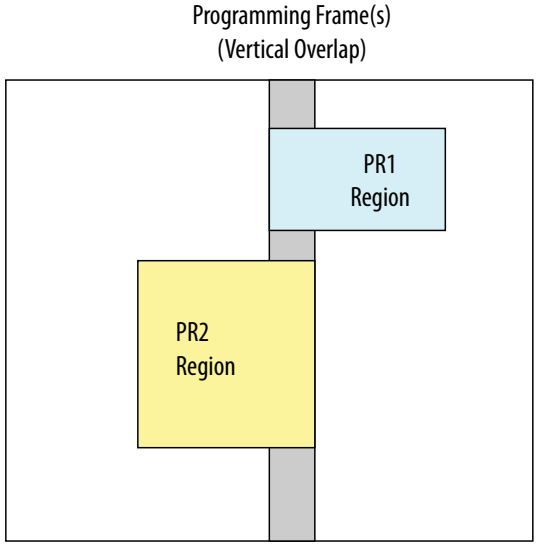
1.3.2. AND/OR Mode

The AND/OR mode refers to how the bits are rewritten. Partial reconfiguration with AND/OR uses a two-pass method.

Simplistically, this can be compared to bits being ANDed with a MASK, and ORed with new values, allowing multiple PR regions to vertically overlap a single column. In the first pass, all the bits in the CRAM frame for a column passing through a PR region are ANDed with '0's while those outside the PR region are ANDed with '1's. After the first pass, all the CRAM bits corresponding to the PR region are reset without modifying the static region. In the second pass for each CRAM frame, new data is ORed with the current value of 0 inside the PR region, and in the static region, the bits are ORed with '0's so they remain unchanged. The programming file size of a PR region using the AND/OR mode could be twice the programming file size of the same PR region using SCRUB mode.

Figure 5. AND/OR Mode

This is the floorplan of a FPGA using AND/OR mode, with two PR regions, with columns that overlap.



Note: If you have overlapping PR regions in your design, you must use AND/OR mode to program all PR regions, including PR regions with no overlap. The Intel Quartus Prime software will not permit the use of SCRUB mode when there are overlapping regions. If none of your regions overlap, you can use AND/OR, SCRUB, or a mixture of both.

1.3.3. Programming File Sizes for a Partial Reconfiguration Project

The programming file size for a partial reconfiguration bitstream is proportional to the area of the PR region.

A partial reconfiguration programming bitstream for AND/OR mode makes two passes on the PR region; the first pass clears all relevant bits, and the second pass sets the necessary bits. Due to this two-pass sequence, the size of a partial bitstream can be larger than a full FPGA programming bitstream depending on the size of the PR region.

When using the AND/OR mode for partial reconfiguration, the formula which describes the approximate file size within ten percent is:

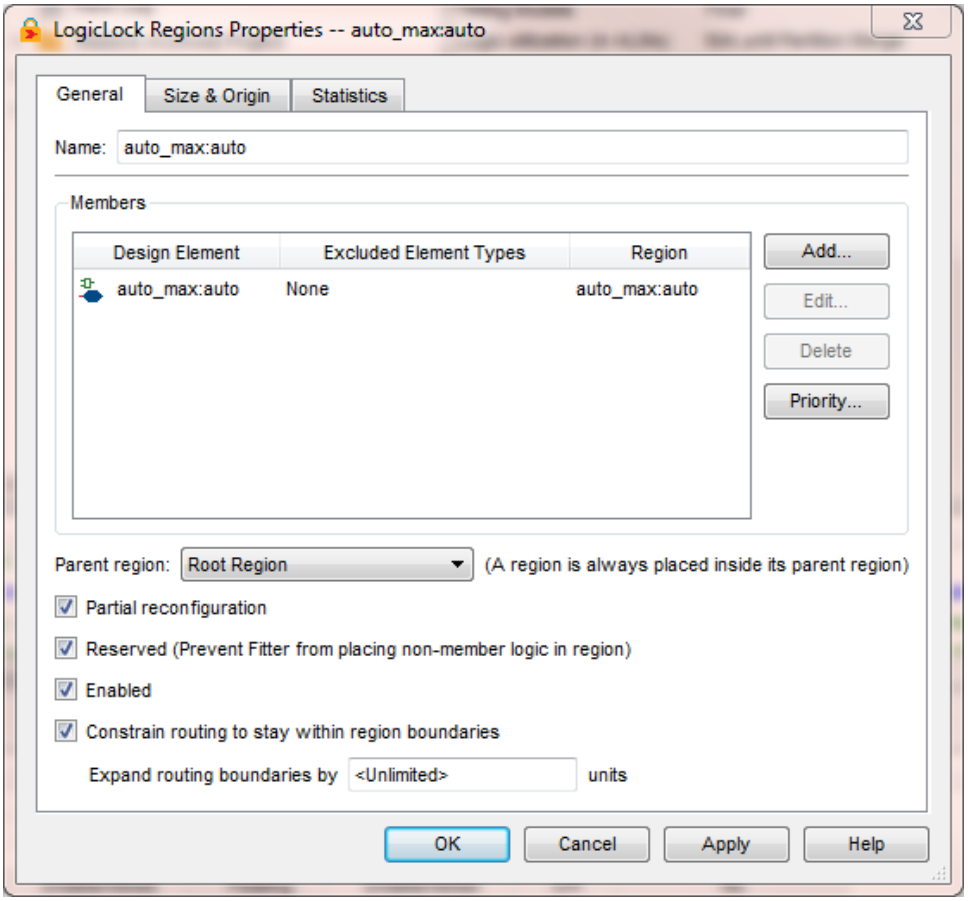
```
PR bitstream size = ((Size of region in the horizontal
direction) /(full horizontal dimension of the part)) * 2 * (size
of full bitstream)
```

The way the Fitter reserves routing for partial reconfiguration increases the effective size for small PR regions from a bitstream perspective. PR bitstream sizes in designs with a single small PR region will not match the file size computed by this equation.

Note: The PR bitstream size is approximately half of the size computed above when using single-pass SCRUB mode. When you use the SCRUB mode with **Use clear/set method** turned on, the bitstream size is comparable to the size calculated for the AND/OR mode.

You can limit expansion of the routing regions in the **LogicLock Regions Properties** dialog box. Alt+L opens the **LogicLock Regions Window**, then right-click on a LogicLock region and click **LogicLock Region Properties**.

Figure 6. LogicLock Regions Properties dialog box



Turn on **Partial reconfiguration**, **Reserved**, **Enabled**, and **Constrain routing to stay within region boundaries**.

You can also control expansion of the routing regions by adding the following two assignments to your Intel Quartus Prime Settings file (`.qsf`):

```
set_global_assignment -name LL_ROUTING_REGION Expanded -
section_id <region name> set_global_assignment -name
LL_ROUTING_REGION_EXPANSION_SIZE 0 -section_id <region name>
```

Adding these to your `.qsf` disables expansion and minimizes the bitstream size.

1.4. Partial Reconfiguration Design Flow

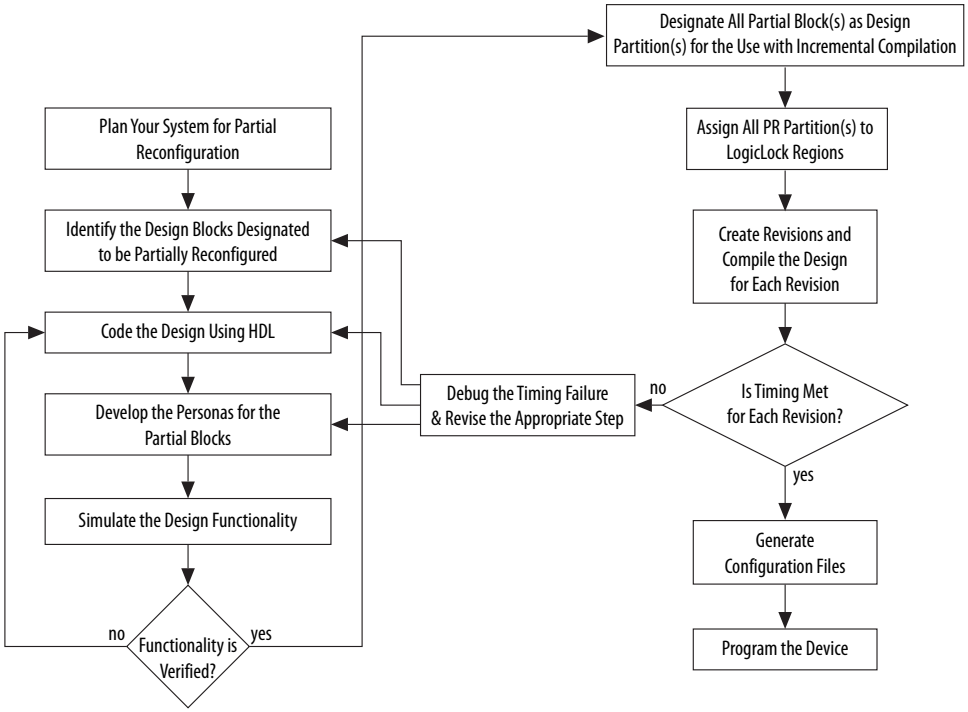
Partial reconfiguration is based on the revision feature in the Intel Quartus Prime software. Your initial design is the base revision, where you define the boundaries of the static region and reconfigurable regions on the FPGA. From the base revision, you create multiple revisions, which contain the static region and describe the differences in the reconfigurable regions.

Two types of revisions are specific to partial reconfiguration: reconfigurable and aggregate. Both import the persona for the static region from the base revision. A reconfigurable revision generates personas for PR regions. An aggregate revision is used to combine personas from multiple reconfigurable revisions to create a complete design suitable for timing analysis.

The design flow for partial reconfiguration also utilizes the Intel Quartus Prime incremental compilation flow. To take advantage of incremental compilation for partial reconfiguration, you must organize your design into logical and physical partitions for synthesis and fitting. The partitions for which partial reconfiguration is enabled (PR partitions) must also have associated LogicLock assignments.

Revisions make use of personas, which are subsidiary archives describing the characteristics of both static and reconfigurable regions, that contain unique logic which implements a specific set of functions to reconfigure a PR region of the FPGA. Partial reconfiguration uses personas to pass this logic from one revision to another.

Figure 7. Partial Reconfiguration Design Flow



The PR design flow requires more initial planning than a standard design flow. Planning requires setting up the design logic for partitioning, and determining placement assignments to create a floorplan. Well-planned partitions can help improve design area utilization and performance, and make timing closure easier. You should also decide whether your system requires partial reconfiguration to originate from the FPGA pins or internally, and which mode you are using; the AND/OR mode or the SCRUB mode, because this influences some of the planning steps described in this section.

You must structure your source code or design hierarchy to ensure that logic is grouped correctly for optimization. Implementing the correct logic grouping early in the design cycle is more efficient than restructuring the code later. The PR flow

requires you to be more rigorous about following good design practices. The guidelines for creating partitions for incremental compilation also include creating partitions for partial reconfiguration.

Use the following best practice guidelines for designing in the PR flow, which are described in detail in this section:

- Determining resources for partial reconfiguration
- Partitioning the design for partial reconfiguration
- Creating incremental compilation partitions for partial reconfiguration
- Instantiating the PR IP core in the design
- Creating wrapper logic for PR regions
- Creating freeze logic for PR regions
- Planning clocks and other global signals for the PR design
- Creating floorplan assignments for the PR design

1.4.1. Design Partitions for Partial Reconfiguration

You must create design partitions for each PR region that you want to partially reconfigure. Optionally, you can also create partitions for the static parts of the design for timing preservation and/or for reducing compilation time.

There is no limit on the number of independent partitions or PR regions you can create in your design. You can designate any partition as a PR partition by enabling that feature in the LogicLock Regions window in the Intel Quartus Prime software.

Partial reconfiguration regions do not support the following IP blocks that require a connection to the JTAG controller:

- In-System Memory Content EditorI
- In-System Signals & Probes
- Virtual JTAG
- Nios II with debug module
- Signal Tap tap or trigger sources

Note: PR partitions can contain only FPGA core resources, they cannot contain I/O or periphery elements.

1.4.2. Incremental Compilation Partitions for Partial Reconfiguration

Use the following best practices guidelines when creating partitions for PR regions in your design:

- Register all partition boundaries; register all inputs and outputs of each partition when possible. This practice prevents any delay penalties on signals that cross partition boundaries and keeps each register-to-register timing path within one partition for optimization.
- Minimize the number of paths that cross partition boundaries.
- Minimize the timing-critical paths passing in or out of PR regions. If there are timing-critical paths that cross PR region boundaries, rework the PR regions to avoid these paths.
- The Intel Quartus Prime software can optimize some types of paths between design partitions for non-PR designs. However, for PR designs, such inter-partition paths are strictly not optimized.

1.4.3. Partial Reconfiguration Controller Instantiation in the Design

Normally you would use the Altera PR IP core to configure the PR process. When you instantiate the PR IP within your PR design, the Stratix V PR control block and the Stratix V CRC block are automatically instantiated in your design. However, you can also write your own custom logic to do the function of the PR IP. In case you are creating your own control logic, or if you are using the PR IP in the external host mode (where in the logic that controls PR process is outside the FPGA undergoing PR operation) , you must instantiate the Stratix V PR control block and the Stratix V CRC block in your design in order to use the PR feature in external host mode. Please refer to the Partial Reconfiguration with an External Host topic for more details.

If you perform PR in internal host mode, you do not have to instantiate the PR control block and the CRC block, since they are are instantiated for you by the PR IP core. Instantiation of the partial reconfiguration controller is required only if your design includes partial reconfiguration in external host mode. Please refer to the *Partial Reconfiguration with an External Host* topic for more details.

When you are manually instantiating the Stratix V Control Block and CRC block, you may want to add the PR control and CRC blocks at the top level of the design.

For example, in a design named Core_Top, all the logic is contained under the Core_Top module hierarchy. Create a wrapper (Chip_Top) at the top-level of the hierarchy that instantiates this Core_Top module, the Stratix V PR control block, and the Stratix V CRC check modules.

If you are performing partial reconfiguration from pins, then the required pins should be on the I/O list for the top-level (Chip_Top) of the project, as shown in the code in the following examples. If you are performing partial reconfiguration from within the core, you may choose another configuration scheme, such as Active Serial, to transmit the reconfiguration data into the core, and then assemble it to 16-bit wide data inside the FPGA within your logic. In such cases, the PR pins are not part of the FPGA I/O.

1.4.3.1. Component Declaration of the PR Control Block and CRC Block in VHDL

To instantiate the PR control block and the CRC block in your design manually, use this code sample containing the component declaration in VHDL. The PR function is performed from within the core (code located in Core_Top) and you must add additional ports to Core_Top to connect to both components. This example is in VHDL but you can create a similar instantiation in Verilog as well.

```
-- The Stratix V control block interface

component stratixv_prblock is
  port(
      clk: in STD_LOGIC := '0';
      corectl: in STD_LOGIC := '0';
      data: in STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
      done: out STD_LOGIC;
      error: out STD_LOGIC;
      externalrequest: out STD_LOGIC;
      prrequest: in STD_LOGIC := '0';
      ready: out STD_LOGIC
      );
end component;

-- The Stratix V CRC block for diagnosing CRC errors

component stratixv_crcblock is
  port(
      shiftnld: in STD_LOGIC ;
      clk: in STD_LOGIC ;
      crcerror: out STD_LOGIC
      );
end component;
```

The following rules apply when connecting the PR control block to the rest of your design:

- The `corectl` signal must be set to '1' (when using partial reconfiguration from core) or to '0' (when using partial reconfiguration from pins).
- The `corectl` signal has to match the **Enable PR pins** option setting in the Device and Pin Options dialog box on the Setting page; if you have turned on **Enable PR pins**, then the `corectl` signal on the PR control block instantiation must be toggled to '0'.
- When performing partial reconfiguration from pins the Intel Quartus Prime software automatically assigns the PR unassigned pins. If you so choose, you can make pin assignments to all the dedicated PR pins in **Pin Planner** or **Assignment Editor**.
- When performing partial reconfiguration from core, you can connect the `prblock` signals to either core logic or I/O pins, excluding the dedicated programming pin such as DCLK.

1.4.3.2. Instantiating the PR Control Block and CRC Block in VHDL

This code example instantiates a PR control block in VHDL, inside your top-level project, Chip_Top:

```
entity Chip_Top is port (
                         --User I/O signals (excluding PR related signals)
                         ..
                         ..
                         );
```

```
end Chip_Top;

-- Following shows the architecture behavior of Chip_Top

m_pr : stratixv_prblock
  port map(
           clk       => dclk,
           corectl   =>'0', --1 - when using PR from inside
                      --0 - for PR from pins; You must also enable
                      -- the appropriate option in Intel Quartus Prime settings
           prrequest => pr_request,
           data      => pr_data,
           error     => pr_error,
           ready     => pr_ready,
           done      => pr_done
          );
m_crc : stratixv_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
          );
```

For more information on port connectivity for reading the Error Message Register (EMR), refer to the following application note.

Related Information

AN539: Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices

1.4.3.3. Instantiating the PR Control Block and CRC Block in Verilog HDL

The following example instantiates a PR control block in Verilog HDL, inside your top-level project, Chip_Top:

```
module Chip_Top (
                //User I/O signals (excluding PR related signals)
                ..
                ..
                //PR interface & configuration signals
                  pr_request,
                  pr_ready,
                  pr_done,
                  crc_error,
                  dclk,
                  pr_data,
                  init_done
                 );

//user I/O signal declaration
..
..
//PR interface and configuration signals declaration
  input   pr_request;
  output  pr_ready;
  output  pr_done;
  output  crc_error;
  input  dclk;
  input  [15:0] pr_data;
  output init_done
```

```
stratixv_prblock stratixv_prblock_inst
  (
    .clk      (dclk),
    .corectl  (1'b0),
    .prrequest(pr_request),
    .data     (pr_data),
    .error    (pr_error),
    .ready    (pr_ready),
    .done     (pr_done)
  );

stratixv_crcblock stratixv_crcblock_inst
  (
    .clk      (clk),
    .shiftnld (1'b1),
    .crcerror (crc_error)
  );
endmodule
```

For more information on port connectivity for reading the Error Message Register (EMR), refer to the following application note.

Related Information

AN539: Test Methodology of Error Detection and Recovery using CRC in Altera FPGA Devices

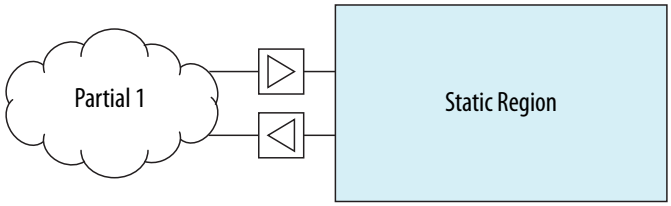
1.4.4. Wrapper Logic for PR Regions

Each persona of a PR region must implement the same input and output boundary ports. These ports act as the boundary between static and reconfigurable logic.

Implementing the same boundary ports ensures that all ports of a PR region remain stationary regardless of the underlying persona, so that the routing from the static logic does not change with different PR persona implementations.

Figure 8. Wire-LUTs at PR Region Boundary

The Intel Quartus Prime software automatically instantiates a wire-LUT for each port of the PR region to lock down the same location for all instances of the PR persona.



If one persona of your PR region has a different number of ports than others, then you must create a wrapper so that the static region always communicates with this wrapper. In this wrapper, you can create dummy ports to ensure that all of the PR personas of a PR region have the same connection to the static region.

The sample code below each create two personas; `persona_1` and `persona_2` are different functions of one PR region. Note that one persona has a few dummy ports. The first example creates partial reconfiguration wrapper logic in Verilog HDL:

```
// Partial Reconfiguration Wrapper in Verilog HDL
module persona    //this module is persona_1
  (
   input reset,
   input [2:0] a,
```

```
   input [2:0] b,
   input [2:0] c,
   output [3:0] p,
   output [7:0] q
  );
reg [3:0] p, q;
always@(a or b)
  begin
    p = a + b ;
  end

always@(a or b or c or p)
  begin
    q = (p*a - b*c )
  end
endmodule

module persona    //this module is persona_2
(
 input reset,
 input [2:0] a,
 input [2:0] b,
 input [2:0] c,   //never used in this persona
 output [3:0] p,
 output [7:0] q   //never assigned in this persona
);
reg [3:0] p, q;
always@(a or b)
  begin
    p = a * b;    // note q is not assigned value in this persona
  end
endmodule
```

The following example creates partial reconfiguration wrapper logic in VHDL.

```
-- Partial Reconfiguration Wrapper in VHDL
-- this module is persona_1
entity persona is
  port(
       a:in STD_LOGIC_VECTOR (2 downto 0);
      b:in STD_LOGIC_VECTOR (2 downto 0);
      c:in STD_LOGIC_VECTOR (2 downto 0);
      p: out STD_LOGIC_VECTOR (3 downto 0);
      q: out STD_LOGIC_VECTOR (7 downto 0)
       );
end persona;

architecture synth of persona is
    begin
      process(a,b)
        begin
        p <= a + b;
       end process;

     process (a, b, c, p)
        begin
          q <= (p*a - b*c);
        end process;
end synth;

-- this module is persona_2
entity persona is
  port(
       a:in STD_LOGIC_VECTOR (2 downto 0);
       b:in STD_LOGIC_VECTOR (2 downto 0);
       c:in STD_LOGIC_VECTOR (2 downto 0);   --never used in this persona
       p:out STD_LOGIC_VECTOR (3 downto 0);
       q:out STD_LOGIC_VECTOR (7 downto 0) --never used in this persona
      );
end persona_2;
```

```
architecture synth of persona_2 is
  begin
    process(a, b)
      begin
        p <= a *b; --note q is not assigned a value in this persona
     end process;
end synth;
```

1.5. Freeze Logic for PR Regions

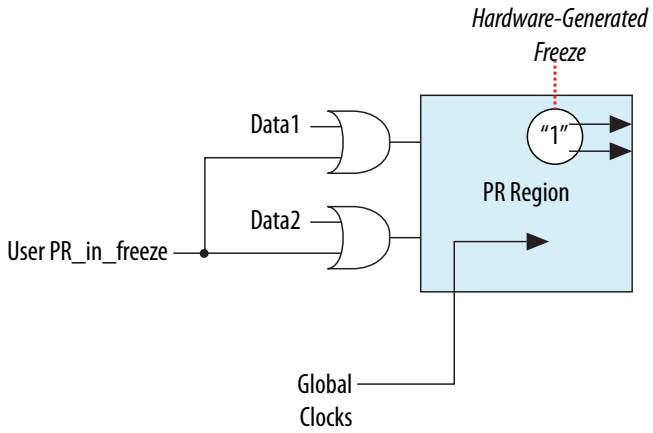
When you use partial reconfiguration, you must freeze all non-global inputs of a PR region except global clocks. Locally routed signals are not considered global signals, and must also be frozen during partial reconfiguration. Freezing refers to driving a '1' on those PR region inputs. When you start a partial reconfiguration process, the chip is in user mode, with the device still running.

When you instantiate the Altera PR IP core in your design, the IP incudes a freeze port which you can use to freeze the non-global inputs of the PR region. In case your design has multiple PR regions, you must create decoding logic to freeze only the inputs of the PR region being partially reconfigured.

If you are not using the Altera PR IP, you must include logic to freeze the inputs of the PR regions in the design as required for proper operation.

Freezing all non-global inputs for the PR region ensures there is no contention between current values that may result in unexpected behavior of the design after partial reconfiguration is complete. Global signals going into the PR region should not be frozen to high. The Intel Quartus Prime software freezes the outputs from the PR region; therefore the logic outside of the PR region is not affected.

Figure 9. Freezing at PR Region Boundary



During partial reconfiguration, the static region logic should not depend on the outputs from PR regions to be at a specific logic level for the continued operation of the static region.

The easiest way to control the inputs to PR regions is by creating a wrapper around the PR region in RTL. In addition to freezing all inputs high, you can also drive the outputs from the PR block to a specific value, if required by your design. For example,

if the output drives a signal that is active high, then your wrapper could freeze the output to GND. The idea is to make sure the static region will not stall or go to indeterminate state, when the PR region is getting a new persona through PR.

The following example implements a freeze wrapper in Verilog HDL, on a module named `pr_module`.

```
module freeze_wrapper
(
 input reset,      // global reset signal
 input freeze,     // PR process active, generated by user logic
 input clk1,       // global clock signal
 input clk2,       // non-global clock signal
 input [3:0] control_mode,
 input [3:0] framer_ctl,
 output [15:0] data_out
);
wire [3:0]control_mode_wr, framer_ctl_wr;
wire clk2_to_wr;
//instantiate pr_module
pr_module pr_module
(
 .reset (reset),                   //input
 .clk1 (clk1),                     //input, global clock
 .clk2 (clk2_to_wr),               // input, non-global clock
 .control_mode (control_mode_wr),  //input
 .framer_ctl (framer_ctl_wr),      //input
 .pr_module_out (data_out)         // collection of outputs from pr_module
);

// Freeze all inputs

assign control_mode_wr = freeze ? 4'hF: control_mode;
assign framer_ctl_wr = freeze ? 4'hF: framer_ctl;
assign clk2_to_wr = freeze ? 1'b1 : clk2;

endmodule
```

The following example implements a freeze wrapper in VHDL, on a module named `pr_module`.

```
entity freeze_wrapper is
port(
     reset:in STD_LOGIC;      -- global reset signal
     freeze:in STD_LOGIC;
     clk1: in STD_LOGIC;      -- global signal
     clk2: in STD_LOGIC;      -- non-global signal
     control_mode: in STD_LOGIC_VECTOR (3 downto 0);
     framer_ctl: in STD_LOGIC_VECTOR (3 downto 0);
     data_out: out STD_LOGIC_VECTOR (15 downto 0)
    );
end freeze_wrapper;

architecture behv of freeze_wrapper is

  component pr_module
  port(
       reset:in STD_LOGIC;
       clk1:in STD_LOGIC;
       clk2:in STD_LOGIC;
       control_mode:in STD_LOGIC_VECTOR (3 downto 0);
       framer_ctl:in STD_LOGIC_VECTOR (3 downto 0);
       pr_module_out:out STD_LOGIC_VECTOR (15 downto 0)
      );
  end component

  signal control_mode_wr: in STD_LOGIC_VECTOR (3 downto 0);
  signal framer_ctl_wr : in STD_LOGIC_VECTOR (3 downto 0);
```

```
  signal clk2_to_wr : STD_LOGIC;
  signal data_out_temp : STD_LOGIC_VECTOR (15 downto 0);
  signal logic_high : STD_LOGIC_VECTOR (3 downto 0):="1111";

  begin
    data_out(15 downto 0) <= data_out_temp(15 downto 0);

  m_pr_module: pr_module
     port map (
               reset => reset,
               clk1 => clk1,
               clk2 => clk2_to_wr,
               control_mode =>control_mode_wr,
               framer_ctl => framer_ctl_wr,
               pr_module_out => data_out_temp);
               -- freeze all inputs

  control_mode_wr <= logic_high when (freeze ='1') else control_mode;
  framer_ctl_wr <= logic_high when (freeze ='1') else framer_ctl;
  clk2_to_wr <= logic_high(0) when (freeze ='1') else clk2;

end architecture;
```

1.5.1. Clocks and Other Global Signals for a PR Design

For non-PR designs, the Intel Quartus Prime software automatically promotes high fan-out signals onto dedicated clocks or other forms of global signals during the pre-fitter stage of design compilation using a process called global promotion. For PR designs, however, automatic global promotion is disabled by default for PR regions, and you must assign the global clock resources necessary for PR partitions. Clock resources can be assigned by making Global Signal assignments in the Intel Quartus Prime Assignment Editor, or by adding Clock Control Block (altclkctrl) IP core blocks in the design that drive the desired global signals.

There are 16 global clock networks in a Stratix V device. However, only six unique clocks can drive a row clock region limiting you to a maximum of six global signals in each PR region. The Intel Quartus Prime software must ensure that any global clock can feed every location in the PR region.

The limit of six global signals to a PR region includes the GCLK, QCLK and PCLKs used inside of the PR region. Make QSF assignments for global signals in your project's Intel Quartus Prime Settings File (`.qsf`), based on the clocking requirements for your design. In designs with multiple clocks that are external to the PR region, it may be beneficial to align the PR region boundaries to be within the global clock boundary (such as QCLK or PCLK).

If your PR region requires more than six global signals, modify the region architecture to reduce the number of global signals within this to six or fewer. For example, you can split a PR region into multiple regions, each of which uses only a subset of the clock domains, so that each region does not use more than six.

Every instance of a PR region that uses the global signals (for example, PCLK, QCLK, GCLK, ACLR) must use a global signal for that input.

Global signals can only be used to route certain secondary signals into a PR region and the restrictions for each block are listed in the following table. Data signals and other secondary signals not listed in the table, such as synchronous clears and clock enables are not supported.

Table 2. Supported Signal Types for Driving Clock Networks in a PR Region

Block Types	Supported Signals for Global/Periphery/Quadrant Clock Networks
LAB	`Clock, ACLR`
RAM	`Clock, ACLR, Write Enable(WE), Read Enable(RE)`
DSP	`Clock, ACLR`

Note: PR regions are allowed to contain output ports that are used outside of the PR region as global signals.

- If a global signal feeds both static and reconfigurable logic, the restrictions in the table also apply to destinations in the static region. For example, the same global signal cannot be used as an SCLR in the static region and an ACLR in the PR region.
- A global signal used for a PR region should only feed core blocks inside and outside the PR region. In particular you should not use a clock source for a PR region and additionally connect the signal to an I/O register on the top or bottom of the device. Doing so may cause the Assembler to give an error because it is unable to create valid programming mask files.

1.5.2. Floorplan Assignments for PR Designs

You must create a LogicLock region so the interface of the PR region with the static region is the same for any persona you implement. If different personas of a PR region have different area requirements, you must make a LogicLock region assignment that contains enough resources to fit the largest persona for the region. The static regions in your project do not necessarily require a floorplan, but depending on any other design requirement, you may choose to create a floorplan for a specific static region. If you create multiple PR regions, and are using SCRUB mode, make sure you have one column or row of static region between each PR region.

There is no minimum or maximum size for the LogicLock region assigned for a PR region. Because wire-LUTs are added on the periphery of a PR region by the Intel Quartus Prime software, the LogicLock region for a PR region must be slightly larger than an equivalent non-PR region. Make sure the PR regions include only the resources that can be partially reconfigured; LogicLock regions for PR can only contain only LABs, DSPs, and RAM blocks. When creating multiple PR regions, make sure there is at least one static region column between each PR region. When multiple PR regions are present in a design, the shape and alignment of the region determines whether you use the SCRUB or AND/OR PR mode.

You can use the default **Auto size** and **Floating location** LogicLock region properties to estimate the preliminary size and location for the PR region.

You can also define regions in the floorplan that match the general location and size of the logic in each partition. You may choose to create a LogicLock region assignment that is non-rectangular, depending on the design requirements, but disjoint LogicLock regions are not allowed for PR regions in your first compilation of the project.

After compilation, use the Fitter-determined size and origin location as a starting point for your design floorplan. Check the quality of results obtained for your floorplan location assignments and make changes to the regions as needed.

Alternatively, you can perform Analysis and Synthesis, and then set the regions to the required size based on resource estimates. In this case, use your knowledge of the connections between partitions to place the regions in the floorplan.

1.6. Implementation Details for Partial Reconfiguration

This section describes implementation details that help you create your PR design.

1.6.1. Interface with the PR Control Block through a PR Host

During partial reconfiguration, a PR bitstream stored outside the FPGA being partially reconfigured must be sent to the PR Control Block in the FPGA. This enables the control block to update the CRAM bits necessary to configure the PR region in the FPGA.

Two scenarios are possible, depending on whether the control logic to transfer the bitstream is located within the FPGA or outside the FPGA being reconfigured.

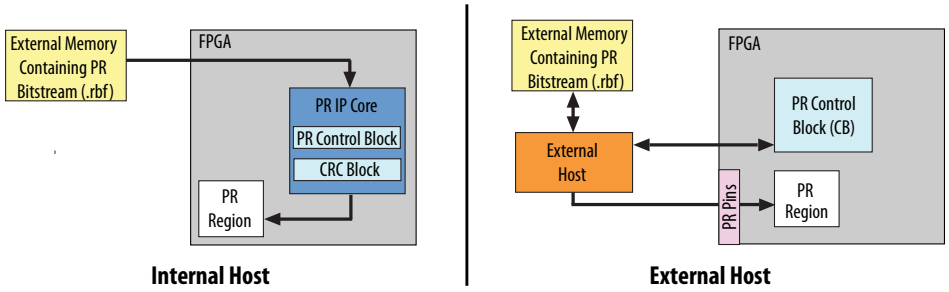
- If the PR IP core is instantiated inside the FPGA being reconfigured, it is termed PR with an internal host; the Altera PR IP core helps you perform the transfer of the PR bitstream.
- When the PR IP is instantiated outside the FPGA being reconfigured, it is termed as PR with an external host.

There is a well-defined interface and a specific protocol to transfer the PR bitstream from the external bitstream source to the PR control block. When you use the Altera PR IP core, the protocol requirements are automatically met by the IP.

It is also possible to write your own control logic, or use a Nios® processor to do this PR bitstream transfer. Note that when create your own control logic for the PR Host, you must make sure to meet the interface requirements described later in this chapter.

Figure 10. Managing Partial Reconfiguration with an Internal or External Host

The figure shows how these blocks should be connected to the PR control block (CB). In your system, you will have either the External Host or the Internal Host, but not both. The external host can be implemented by instantiating the PR IP core outside the FPGA being reconfigured, may be in another Altera FPGA, or processor/PC (PR over PCIe) , or can be implemented by user logic.



The PR mode is independent of the full chip programming mode. You can use any of the supported full chip configuration modes for configuring the full FPGA for your PR design.

If you are creating your own custom logic for implementing a PR internal host, you can use any interface to load the PR bitstream data to the FPGA; for example, from a serial or a parallel flash device; and then format the PR bitstream data to match the FPPx16 interface on the PR Control Block.

When using an external host, you must implement the control logic for managing system aspects of partial reconfiguration on an external device. To use the external host for your design, turn on the **Enable PR Pins** option in the **Device and Pin Options** dialog box in the Intel Quartus Prime software when you compile your design. If this setting is turned off, then you must use an internal host. Also, you must tie the `corectl` port on the PR control block instance in the top-level of the design to the appropriate level for the selected mode.

Related Information

Partial Reconfiguration Pins on page 23
Partial Reconfiguration Dedicated Pins Table

1.6.2. Partial Reconfiguration Pins

Partial reconfiguration can be performed through external pins or from inside the core of the FPGA.

When using PR from pins, some of the I/O pins are dedicated for implementing partial reconfiguration functionality. If you perform partial reconfiguration from pins, then you must use the passive parallel with 16 data bits (FPPx16) configuration mode. All dual-purpose pins should also be specified to **Use as regular I/O**.

To enable partial reconfiguration from pins in the Intel Quartus Prime software, perform the following steps:

1. From the Assignments menu, click **Device**, then click **Device and Pin Options**.
2. In the **Device and Pin Options** dialog box, select **Partial Reconfiguration** in the **Category** list and turn on **Enable PR pins** from the **Options** list.
3. Click **Configuration** in the **Category** list and select **Passive Parallel x16** from the **Configuration scheme** list.
4. Click **Dual-Purpose Pins** in the **Category** list and verify that all pins are set to **Use as regular I/O** rather than **Use as input tri-stated**.
5. Click **OK**, or continue to modify other settings in the **Device and Pin Options** dialog box.
6. Click **OK**.

Note: You can enable open drain on PR pins from the **Device and Pin Options** dialog box in the **Partial Reconfiguration** dialog box.

Table 3. Partial Reconfiguration Dedicated Pins Description

Pin Name	Pin Type	Pin Description
PR_REQUEST	Input	Dedicated input when **Enable PR pins** is turned on; otherwise, available as user I/O. Logic high on pin indicates the PR host is requesting partial reconfiguration.
PR_READY	Output	Dedicated output when **Enable PR pins** is turned on; otherwise, available as user I/O. Logic high on this pin indicates the Stratix V control block is ready to begin partial reconfiguration.

continued...

Pin Name	Pin Type	Pin Description
PR_DONE	Output	Dedicated output when **Enable PR pins** is turned on; otherwise, available as user I/O. Logic high on this pin indicates that partial reconfiguration is complete.
PR_ERROR	Output	Dedicated output when **Enable PR pins** is turned on; otherwise, available as user I/O. Logic high on this pin indicates the device has encountered an error during partial reconfiguration.
DATA[15:0]	Input	Dedicated input when **Enable PR pins** is turned on; otherwise available as user I/O. These pins provide connectivity for PR_DATA to transfer the PR bitstream to the PR Controller.
DCLK	Bidirectional	Dedicated input when **Enable PR pins** is turned on; PR_DATA is sent synchronous to this clock.

For more information on different configuration modes for Stratix V devices, and specifically about FPPx16 mode, refer to the *Configuration, Design Security, and Remote System Upgrades in Stratix V Devices* chapter of the *Stratix V Handbook*.

Related Information

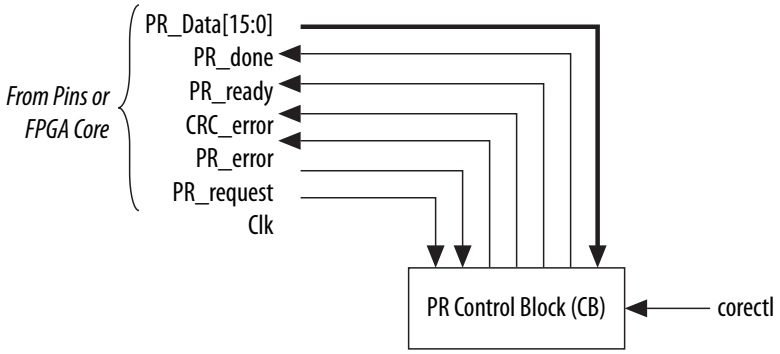
Configuration, Design Security, and Remote System Upgrades in Stratix V Devices

1.6.3. PR Control Signals Interface

You can use the Intel Quartus Prime **Assembler** and the **Convert Programming File** utilities to generate the different bitstreams necessary for full chip configuration and for partial reconfiguration. The programming bit-stream for partial reconfiguration contains the instructions (opcodes) as well as the configuration bits, necessary for reconfiguring each of the partial regions. When using an external host, the interface ports on the control block are mapped to FPGA pins. When using an internal host, these signals are within the core of the FPGA. When using the PR IP core as an internal host, connect the signals on the PR IP core appropirately as described in the Partial Reconfiguration IP Core User Guide and follow the instructions to start the PR process on the FPGA. If you are not using the PR IP core, make sure you understand these PR interface signals.

Figure 11. Partial Reconfiguration Interface Signals

These handshaking control signals are used for partial reconfiguration.



- PR_DATA: The configuration bitstream is sent on PR_ DATA[15:0], synchronous to the Clk.
- PR_DONE: Sent from CB to control logic indicating the PR process is complete.
- PR_READY: Sent from CB to control logic indicating the CB is ready to accept PR data from the control logic.
- CRC_Error: The CRC_Error generated from the device's CRC block, is used to determine whether to partially reconfigure a region again, when encountering a CRC_Error.
- PR_ERROR: Sent from CB to control logic indicating an error during partial reconfiguration.
- PR_REQUEST: Sent from your control logic to CB indicating readiness to begin the PR process.
- corectl: Determines whether partial reconfiguration is performed internally or through pins.

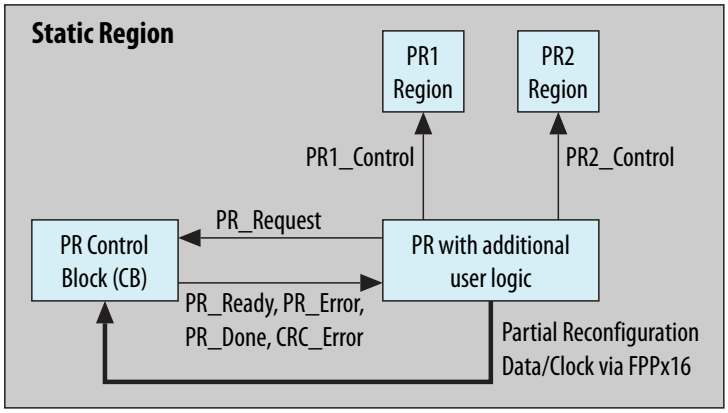
1.6.4. Reconfiguring a PR Region

The figure below shows an internal host for PR, where the PR IP core is implemented inside the FPGA. However, these principles are also applicable for partial reconfiguration with an external host.

The PR control block (CB) represents the Stratix V PR controller inside the FPGA. PR1 and PR2 are two PR regions in a user design. In addition to the four control signals (PR_REQUEST, PR_READY, PR_DONE, PR _ERROR) and the data/clock signals interfacing with the PR control block, your PR Control IP should also send a control signal (PR_CONTROL) to each PR region. This signal implements the freezing and unfreezing of the PR Interface signals. This is necessary to avoid contention on the FPGA routing fabric. In a case such as this, you need to add some decoding logic in the design, in addition to instantiating the PR IP core.

Figure 12. Example of a PR System with Two PR Regions

Implementation of PR Control logic in the FPGA.



After the FPGA device has been configured with a full chip configuration at least once, the INIT_DONE signal is released, and the signal is asserted high due to the external resistor on this pin. The INIT_DONE signal must be assigned to a pin to monitor it externally. When a full chip configuration is complete, and the device is in user mode, the following steps describe the PR sequence:

1. Begin a partial reconfiguration process from your PR Control logic, which initiates the PR process for one or more of the PR regions (asserting PR1_Control or PR2_Control in the figure). The wrapper HDL described earlier freezes (pulls high) all non-global inputs of the PR region before the PR process.
2. If you are using the PR IP core, use the PR_START signal to start reconfiguring the PR region. When you are not using the PR IP core, your control logic should send the PR_REQUEST signal from your control logic to the PR Control Block (CB). If your design uses an external controller, monitor INIT_DONE to verify that the chip is in user mode before asserting the PR_START or PR_REQUEST signal. The CB initializes itself to accept the PR data and clock stream. After that, the CB asserts a PR_READY signal to indicate it can accept PR data. If you are using the PR IP, the timing relationsips between the control and data signals is managed by the IP core. Data and clock signals are sent to the PR control block to partially reconfigure the PR region interface.

   Note: If you write your own controller logic, specify that exactly four clock-cycles must occur before sending the PR data to make sure the PR process progresses correctly.

   - When there are multiple PR personas for the PR region, your control logic must determine the programming file data for partial reconfiguration and specify the correct file.
   - When there are multiple PR regions in the design, then your control logic determines which regions require reconfiguration based on system requirements.
   - At the end of the PR process, the PR control block asserts a PR_DONE signal and deasserts the PR_READY signal. The Altera PR IP core further processes these signals to assert a 3-bit status signal. If you are not using the Altera PR IP, your design must take approcpriate action as defined by the timing diagrams when PR_DONE is asserted.
   - If you want to suspend sending data, you can implement logic to pause the clock at any point.
3. When you are not using the PR IP core, your custom control logic must deassert the PR_REQUEST signal within eight clock cycles after the PR_DONE signal goes high. If your logic does not deassert the PR_REQUEST signal within eight clock cycles, a new PR cycle starts.
4. If your design includes additional PR regions, repeat steps 2 – 3 for each region. Otherwise, proceed to step 5.
5. When you are not using the PR IP core, your custom control logic must deassert the PR_CONTROL signal(s) to the PR region. The freeze wrapper releases all input signals of the PR region, thus the PR region is ready for normal user operation.
6. You must perform a reset cycle to the PR region to bring all logic in the region to a known state. After partial reconfiguration is complete for a PR region, the states in which the logic in the region come up is unknown.

The PR event is now complete, and you can resume operation of the FPGA with the newly configured PR region.

At any time after the start of a partial reconfiguration cycle, the PR host can suspend sending the PR_DATA, but the host must suspend sending the PR_CLK at the same time. If the PR_CLK is suspended after a PR process, there must be at least 20 clock cycles after the PR_DONE or PR_ERROR signal is asserted to prevent incorrect behavior.

For an overview of different reset schemes in Altera devices, refer to the *Recommended Design Practices* chapter in the *Intel Quartus Prime Handbook*.

Related Information

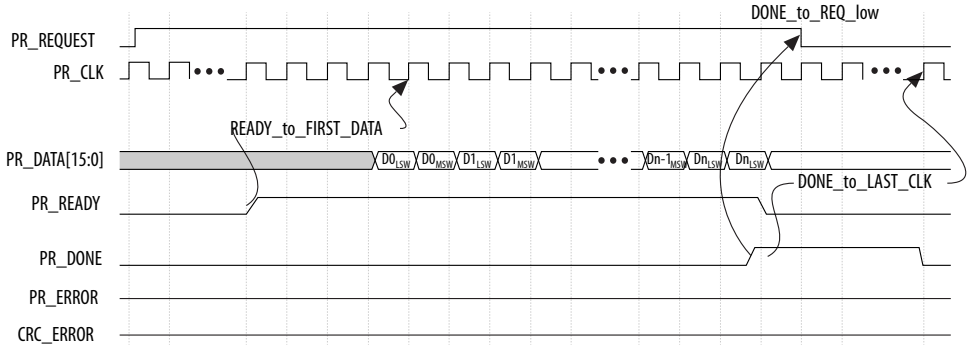
Partial Reconfiguration Cycle Waveform on page 27
For more information on clock requirements for partial reconfiguration.

1.6.5. Partial Reconfiguration Cycle Waveform

When you are using the Altera PR IP in the internal host mode, all the timing relations between various interface signals are met by default, and you can skip reading this section. If you are using PR with an external host or implementing your own custom PR internal host logic, pay attention to these timing relationships when designing your logic. The PR host initiates the PR request, transfers the data to the FPGA device when it is ready, and monitors the PR process for any errors or until it is done.

A PR cycle is initiated by the host (internal or external) by asserting the PR_REQUEST signal high. When the FPGA device is ready to begin partial reconfiguration, it responds by asserting the PR_READY signal high. The PR host responds by sending configuration data on DATA [15:0]. The data is sent synchronous to PR_CLK. When the FPGA device receives all PR data successfully, it asserts the PR_DONE high, and de-asserts PR_READY to indicate the completion of the PR cycle. The PR host must monitor the PR process until either the successful completion of PR (indicated by PR_DONE), or an error condition is asserted.

Figure 13. Partial Reconfiguration Timing Diagram



If there is an error encountered during partial reconfiguration, the FPGA device asserts the PR_ERROR signal high and de-asserts the PR_READY signal low.

Whenever either of these two signals are asserted, the host must de-assert PR_REQUEST within eight PR_CLK cycles. As a response to PR_ERROR error, the host can optionally request another partial reconfiguration or perform a full FPGA configuration.

To prevent incorrect behavior, the PR_CLK signal must be active a minimum of twenty clock cycles after PR_DONE or PR_ERROR signal is asserted high. Once PR_DONE is asserted, PR_REQUEST must be de-asserted within eight clock cycles. PR_DONE is de-asserted by the device within twenty PR_CLK cycles. The host can assert PR_REQUEST again after the 20 clocks after PR_DONE is de-asserted.

Table 4. Partial Reconfiguration Clock Requirements

Signal timing requirements for partial reconfiguration.

Timing Parameters	Value (clock cycles)
PR_READY to first data	4 (exact)
PR_ERROR to last clock	20 (minimum)
PR_DONE to last clock	20 (minimum)
DONE_to_REQ_low	8 (maximum)
Compressed PR_READY to first data	4 (exact)
Encrypted PR_READY to first data (when using double PR)	8 (exact)
Encrypted and Compressed PR_READY to first data (when using double PR)	12 (exact)

At any time during partial reconfiguration, to pause sending PR_DATA, the PR host can stop toggling PR_CLK. The clock can be stopped either high or low.

At any time during partial reconfiguration, the PR host can terminate the process by de-asserting the PR request. A partially completed PR process results in a PR error. You can have the PR host restart the PR process after a failed process by sending out a new PR request 20 cycles later.

If you terminate a PR process before completion, and follow it up with a full FPGA configuration by asserting nConfig, then you must toggle PR_CLK for an additional 20 clock cycles prior to asserting nConfig to flush the PR_CONTROL_BLOCK and avoid lock up.

During these steps, the PR control block might assert a PR_ERROR or a CRC_ERROR signal to indicate that there was an error during the partial reconfiguration process. Assertion of PR_ERROR indicates that the PR bitstream data was corrupt, and the assertion of CRC error indicates a CRAM CRC error either during or after completion of PR process. If the PR_ERROR or CRC_ERROR signals are asserted, you must plan whether to reconfigure the PR region or reconfigure the whole FPGA, or leave it unconfigured.

Important: The PR_CLK signal has different a nominal maximum frequency for each device. Most Stratix V devices have a nominal maximum frequency of at least 62.5 MHz.

1.7. Example of a Partial Reconfiguration Design with an External Host

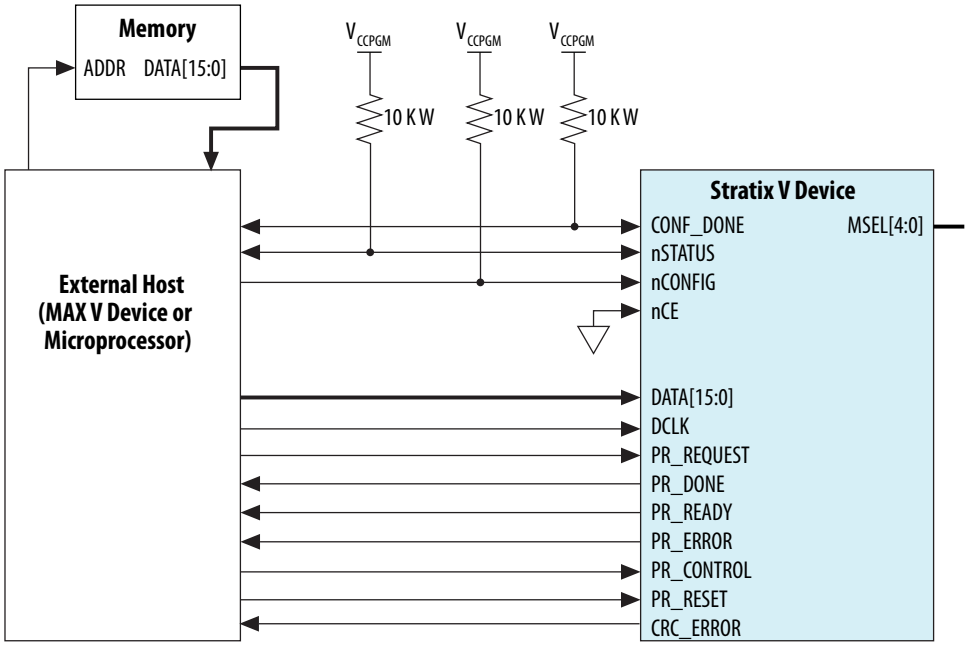
For partial reconfiguration using an external host, you must set the MSEL [4:0] pins for FPPx16 configuration scheme.

You can use Altera PR IP implemented by another supported Altera FPGA device, implement your own control logic in an FPGA or CPLD, or use a microcontroller to implement the configuration and PR controller. In this setup, shown in the following figure, the Stratix V device configures in FPPx16 mode during power-up. Alternatively, you can use a JTAG interface to configure the Stratix V device.

At any time during user-mode, the external host can initiate partial reconfiguration and monitor the status using the external PR dedicated pins: PR_REQUEST, PR_READY, PR_DONE, and PR_ERROR. In this mode, the external host must respond appropriately to the hand-shaking signals for a successful partial reconfiguration. This includes acquiring the data from the flash memory and loading it into the Stratix V device on DATA[15:0].

Figure 14. Connecting to an External Host

The connection setup for partial reconfiguration with an external host in the FPPx16 configuration scheme.



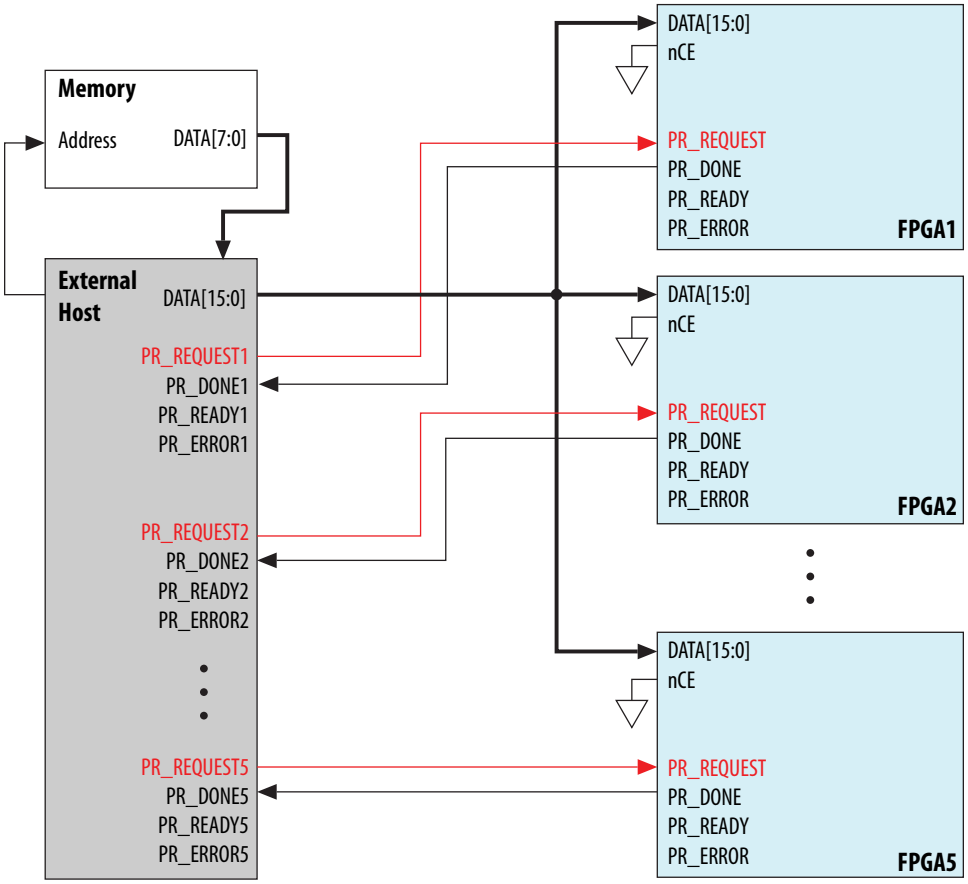
Note: If you don't care to write an external host controller, you can implement an external host with the Partial Reconfiguration IP core on a MAX 10 or other FPGA device.

1.7.1. Example of Using an External Host with Multiple Devices

You must design the external host to accommodate the arbitration scheme that is required for your system, as well as the partial reconfiguration interface requirement for each device.

Figure 15. Connecting Multiple FPGAs to an External Host

An example of an external host controlling multiple Stratix V devices on a board.



1.8. Example Partial Reconfiguration with an Internal Host

You can create PR internal host logic with the PR IP core. If your design uses an internal host, the PR IP core handles the required hand-shaking protocol with the PR control block.

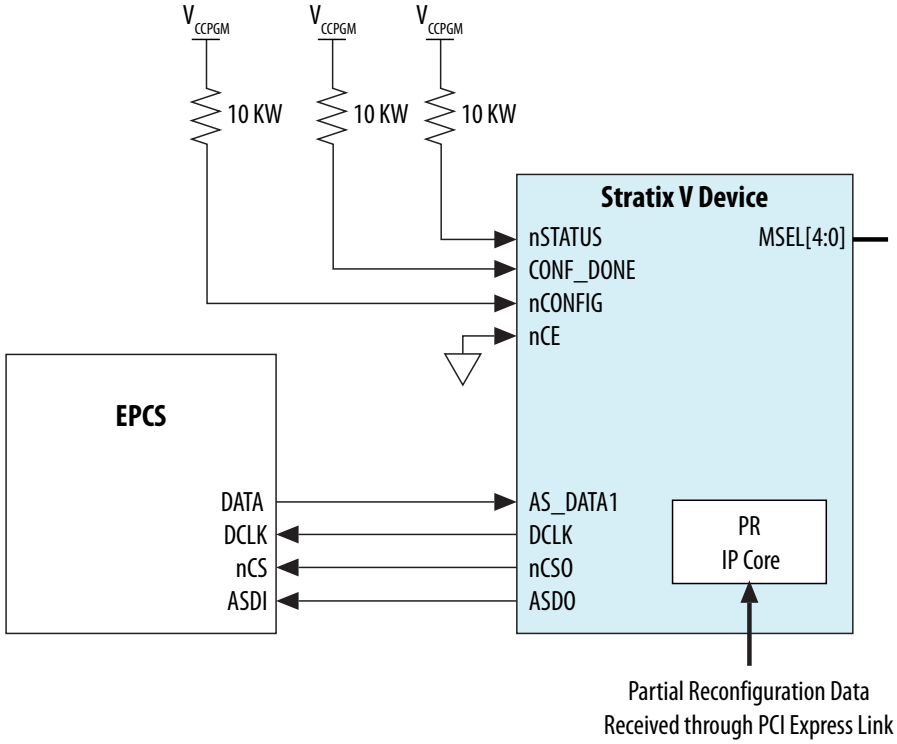
The PR programming bitstream(s) stored in an external flash device can be routed through the regular I/Os of the FPGA device, or received through the high speed transceiver channel (PCI Express, SRIO or Gigabit Ethernet), for processing by the internal host.

The PR dedicated pins (PR_REQUEST, PR_READY, PR_DONE, and PR_ERROR) can be used as regular I/Os when performing partial reconfiguration with an internal host. For the full FPGA configuration upon power-up, you can set the `MSEL[4:0]` pins to match the configuration scheme, for example, Active Serial, Passive Serial, FPPx8, FPPx16, or FPPx32. Alternatively, you can use the JTAG interface to configure the FPGA device. At any time during user-mode, you can initiate partial reconfiguration through the FPGA core fabric using the PR internal host.

In the following figure, the programming bitstream for partial reconfiguration is received through the PCI Express link, and your logic converts the data to the FPPx16 mode.

Figure 16. Connecting to an Internal Host

An example of the configuration setup when performing partial reconfiguration using the internal host.



1.9. Partial Reconfiguration Project Management

When compiling your PR project, you must create a base revision, and one or more reconfigurable revisions. The project revision you start out is termed the base revision.

1.9.1. Create Reconfigurable Revisions

To create a reconfigurable revision, use the **Revisions** tab of the **Project Navigator** window in the Intel Quartus Prime software. When you create a reconfigurable revision, the Intel Quartus Prime software adds the required assignments to associate the reconfigurable revision with the base revision of the PR project. You can add the necessary files to each revision with the **Add/Remove Files** option in the **Project** option under the **Project** menu in the Intel Quartus Prime software. With this step, you can associate the right implementation files for each revision of the PR project.

Important: You must use the **Revisions** tab of the **Project Navigator** window in the Intel Quartus Prime software when creating revisions for partial reconfiguration. Revisions created using **Project ➤ Revisions** cannot be reconfigured.

1.9.2. Compiling Reconfigurable Revisions

Altera recommends that you use the largest persona of the PR region for the base compilation so that the Intel Quartus Prime software can automatically budget sufficient routing.

Here are the typical steps involved in a PR design flow.

1. Compile the base revision with the largest persona for each PR region.
2. Create reconfigurable revisions for other personas of the PR regions by right-clicking in the **Revisions** tab in the Project Navigator.
3. Compile your reconfigurable revisions.
4. Analyze timing on each reconfigurable revision to make sure the design performs correctly to specifications.
5. Create aggregate revisions as needed.
6. Create programming files.

For more information on compiling a partial reconfiguration project, refer to *Performing Partial Reconfiguration* in Intel Quartus Prime Help.

1.9.3. Timing Closure for a Partial Reconfiguration Project

As with any other FPGA design project, simulate the functionality of various PR personas to make sure they perform to your system specifications. You must also make sure there are no timing violations in the implementation of any of the personas for every PR region in your design project.

In the Intel Quartus Prime software, this process is manual, and you must run multiple timing analyses, on the base, reconfigurable, and aggregate revisions. The different timing requirements for each PR persona can be met by using different SDC constraints for each of the personas.

The interface between the partial and static partitions remains identical for each reconfigurable and aggregate revision in the PR flow. If all the interface signals between the static and the PR regions are registered, and there are no timing violations within the static region as well as within the PR regions, the reconfigurable and aggregate revisions should not have any timing violations.

However, you should perform timing analysis on the reconfigurable and aggregate revisions, in case you have any unregistered signals on the interface between partial reconfiguration and static regions.

1.9.4. PR Bitstream Compression and Encryption (Intel Arria® 10 Designs)

You can compress and encrypt the base bitstream and the PR bitstream for your PR project using options available in the Intel 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 PR images only when the base image is encrypted.
- The Encryption Key Programming (`.ekp`) file generates when encrypting the base image and must be used for encrypting the PR bitstream.
- When you compress the bitstream, present each `PR_DATA[15:0]` word for exactly four clock cycles.

For partial reconfiguration with the PR Controller IP core, specify enhanced compression by turning on the **Enhanced compression** option when specifying the parameters in the IP Catalog or Platform Designer parameter editors.

Note: You cannot use encryption with enhanced compression simultaneously.

Table 5. Partial Reconfiguration Clock Requirements for Bitstream Compression

Timing Parameters	Value (clock cycles)
`PR_READY` to first data	4 (exact)
`PR_ERROR` to last clock	80 (minimum)
`PR_DONE` to last clock	80 (minimum)
`DONE_to_REQ_low`	8 (maximum)

Related Information

- Enable Partial Reconfiguration Bitstream Decompression when Configuring Base Design SOF file in JTAG mode on page 38
- Enable Bitstream Decryption Option on page 39
- Generate PR Programming Files with the Convert Programming Files Dialog Box on page 36

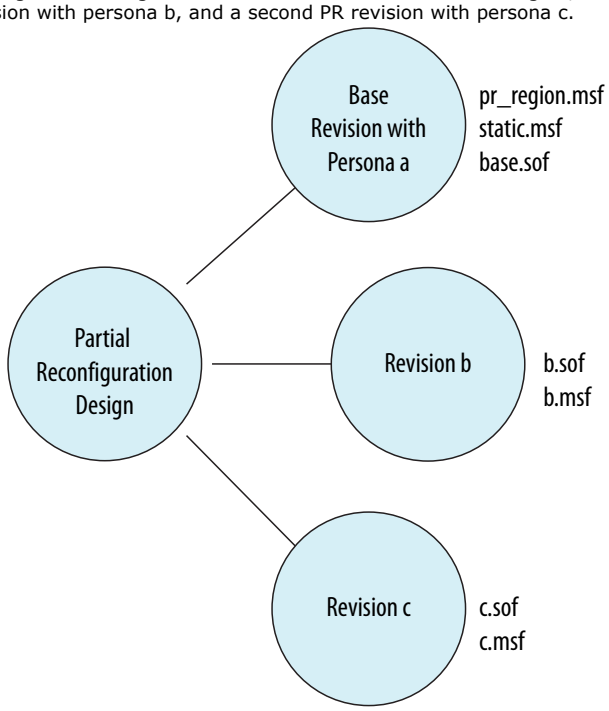
1.10. Programming Files for a Partial Reconfiguration Project

You must generate PR bitstream(s) based on the designs and send them to the control block for partial reconfiguration.

Compile the PR project, including the base revision and at least one reconfigurable revision before generating the PR bitstreams. The Intel Quartus Prime Programmer generates PR bitstreams. This generated bitstream can be sent to the PR ports on the control block for partial reconfiguration.

Figure 17. PR Project with Three Revisions

Consider a partial reconfiguration design that has three revisions and one PR region, a base revision with persona a, one PR revision with persona b, and a second PR revision with persona c.

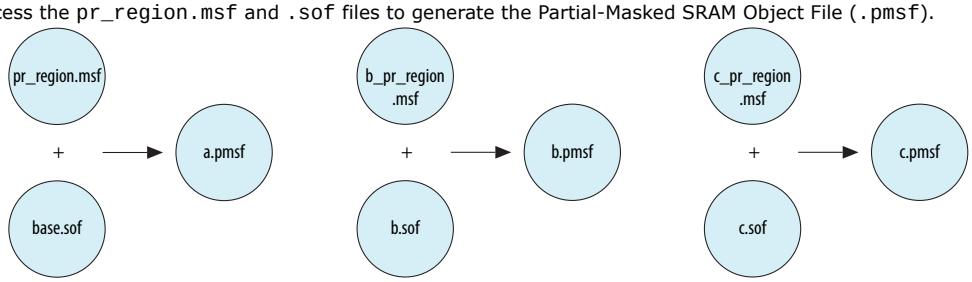


When these individual revisions are compiled in the Intel Quartus Prime software, the assembler produces Masked SRAM Object Files (`.msf`) and the SRAM Object Files (`.sof`) for each revision. The `.sof` files are created as before (for non-PR designs). Additionally, `.msf` files are created specifically for partial reconfiguration, one for each revision. The `pr_region.mfsf` file is the one of interest for generating the PR bitstream. It contains the mask bits for the PR region. Similarly, the `static.msf` file has the mask bits for the static region. The `.sof` files have the information on how to configure the static region as well as the corresponding PR region. The `pr_region.msf` file is used to mask out the static region so that the bitstream can be computed for the PR region. The default file name of the pr region `.msf` corresponds to the LogicLock region name, unless the name is not alphanumeric. In the case of a non-alphanumeric region name, the `.msf` file is named after the location of the lower left most coordinate of the region.

Note: Altera recommends naming all LogicLock regions to enhance documenting your design.

Figure 18. Generation of Partial-Masked SRAM Object Files (.pmsf)

You can convert files in the Convert Programming Files window or run the `quartus_cpf -p` command to process the `pr_region.msf` and `.sof` files to generate the Partial-Masked SRAM Object File (`.pmsf`).



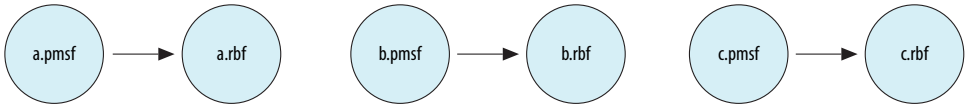
The `.msf` file helps determine the PR region from each of the `.sof` files during the PR bitstream computation.

Once all the `.pmsf` files are created, process the PR bitstreams by running the `quartus_cpf -o` command to produce the raw binary `.rbf` files for reconfiguration.

If one wishes to partially reconfigure the PR region with persona a, use the `a.rbf` bitstream file, and so on for the other personas.

Figure 19. Generating PR Bitstreams

This figure shows how three bitstreams can be created to partially reconfigure the region with persona a, persona b, or persona c as desired.



In the Intel Quartus Prime software, the Convert Programming Files window supports the generation of the required programming bitstreams. When using the `quartus_cpf` from the command line, the following options for generating the programming files are read from an option text file, for example, `option.txt`.

- If you want to use SCRUB mode, before generating the bitstreams create an option text file, with the following line:

  ```
  use_scrub=on
  ```

- If you have initialized M20K blocks in the PR region (ROM/Initialized RAM), then add the following line in the option text file, before generating the bitstreams:

  ```
  write_block_memory_contents=on
  ```

- If you want to compress the programming bitstream files, add the following line in the option text file. This option is available when converting base `.sof` to any supported programming file types, such as `.rbf`, `.pof` and JTAG Indirect Configuration File (`. jic`).

  ```
  bitstream_compression=on
  ```

Related Information

Generate PR Programming Files with the Convert Programming Files Dialog Box on page 36

1.10.1. Generating Required Programming Files

1. Generate `.sof` and `.msf` files (part of a full compilation of the base and PR revisions).
2. Generate a Partial-Masked SRAM Object File (`.pmsf`) using the following commands:

   ```
   quartus_cpf -p <pr_revision>.msf <pr_revision>.sof
   <new_filename>.pmsf
   ```

   for example:

   ```
   quartus_cpf -p x7y48.msf switchPRBS.sof x7y48_new.pmsf
   ```
3. Convert the `.pmsf` file for every PR region in your design to `.rbf` file format. The `.rbf` format is used to store the bitstream in an external flash memory. This command should be run in the same directory where the files are located:

   ```
   quartus_cpf -o scrub.txt -c <pr_revision >.pmsf
   <pr_revision>.rbf
   ```

   for example:

   ```
   quartus_cpf -o scrub.txt -c x7y48_new.pmsf x7y48.rbf
   ```

When you do not have an option text file such as `scrub.txt`, the files generated would be for AND/OR mode of PR, rather than SCRUB mode.

1.10.2. Generate PR Programming Files with the Convert Programming Files Dialog Box

In the Intel Quartus Prime software, the flow to generate PR programming files is supported in the Convert Programming Files dialog box. You can specify how the Intel Quartus Prime software processes file types such as `.msf`, `.pmsf`, and `.sof` to create `.rbf` and merged `.msf` and `.pmsf` files.

You can create

- A `.pmsf` output file, from `.msf` and `.sof` input files
- A `.rbf` output file from a `.pmsf` input file
- A merged `.msf` file from two or more `.msf` input files
- A merged `.pmsf` file from two or more `.pmsf` input files

Convert Programming Files dialog box also allows you to enable the option bit for bitstream decompression during partial reconfiguration, when converting the base `.sof` (full design `.sof`) to any supported file type.

1.10.2.1. Generating a .pmsf File from a .msf and .sof Input File

Perform the following steps in the Intel Quartus Prime software to generate the `.pmsf` file in the **Convert Programming Files** dialog box.

1. Open the **Convert Programming Files** dialog box.
2. Specify the programming file type as `Partial-Masked SRAM Object File (.pmsf)`.
3. Specify the output file name.
4. Select input files to convert (only a single `.msf` and `.sof` file are allowed). Click **Add**.
5. Click **Generate** to generate the `.pmsf` file.

1.10.2.2. Generating a .rbf File from a .pmsf Input File

Perform the following steps in the Intel Quartus Prime software to generate the partial reconfiguration `.rbf` file in the **Convert Programming Files** dialog box.

1. From the File menu, click **Convert Programming Files**.
2. Specify the programming file type as `Raw Binary File for Partial Reconfiguration (.rbf)`.
3. Specify the output file name.
4. Select input file to convert. Only a single `.pmsf` input file is allowed. Click **Add**.
5. Select the new `.pmsf` and click **Properties**.
6. Turn the **Compression**, **Enable SCRUB mode**, **Write memory contents**, and **Generate encrypted bitstream** options on or off depending on the requirements of your design. Click **Generate** to generate the `.rbf` file for partial reconfiguration.

- **Compression**: Enables compression on the PR bitstream.
- **Enable SCRUB mode**: Default is based on AND/OR mode. This option is valid only when your design does not contain vertically overlapped PR masks. The `.rbf` generation fails otherwise.
- **Write memory contents**: Turn this on when you have a `.mif` that was used during compilation. Otherwise, turning this option on forces you to use double PR in AND/OR mode.
- **Generate encrypted bitstream**: If this option is enabled, you must specify the Encrypted Key Programming (`.ekp`) file, which generated when converting a base `.sof` to an encrypted bitstream. The same `.ekp` must be used to encrypt the PR bitstream.

When you turn on **Compression**, you must present each `PR_DATA[15:0]` word for exactly four clock cycles.

Turn on the **Write memory contents** option only if you are using AND/OR mode and have M20K blocks in your PR design that need to be initialized. When you check this box, you must to perform double PR for regions with initialized M20K blocks.

Related Information

Initializing M20K Blocks with a Double PR Cycle on page 45

1.10.2.3. Create a Merged .msf File from Multiple .msf Files

You can merge two or more `.msf` files in the **Convert Programming Files** window.

1. Open the **Convert Programming Files** window.
2. Specify the programming file type as `Merged Mask Settings File (.msf)`.
3. Specify the output file name.
4. Select **MSF Data** in the **Input files to convert** window.
5. Click Add File to add input files. You must specify two or more files for merging.
6. Click **Generate**to generate the merged file.

To merge two or more `.msf` files from the command line, type:

```
quartus_cpf --merge_msf=<number of merged files>
<msf_input_file_1> <msf_input_file_2> <msf_input_file_etc>
<msf_output_file>
```

For example, to merge two `.msf` files, type:

```
quartus_cpf --merge_msf=<2> <msf_input_file_1> <msf_input_file_2>
<msf_output_file>
```

1.10.2.4. Generating a Merged .pmsf File from Multiple .pmsf Files

You can merge two or more `.pmsf` files in the **Convert Programming Files** window.

1. Open the **Convert Programming Files** window.
2. Specify the programming file type as `Merged Partial-Mask SRAM Object File (.pmsf)`.
3. Specify the output file name.
4. Select **PMSF Data** in the **Input files to convert** window.
5. Click **Add File** to add input files. You must specify two or more files for merging.
6. Click **Generate** to generate the merged file.

To merge two or more `.pmsf` files from the command line, type:

```
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:

```
quartus_cpf --merge_pmsf=<2> <pmsf_input_file_1>
<pmsf_input_file_2> <pmsf_output_file>
```

The merge operation checks for any bit conflict on the input files, and the operation fails with error message if a bit conflict is detected. In most cases, a successful file merge operation indicates input files do not have any bit conflict.

1.10.2.5. Enable Partial Reconfiguration Bitstream Decompression when Configuring Base Design SOF file in JTAG mode

In the Intel Quartus Prime software, the **Convert Programming Files** window provides the option in the `.sof` file properties to enable bitstream decompression during partial reconfiguration.

This option is available when converting base .sof to any supported programming file types, such as .rbf, .pof, and .jic.

In order to view this option, the base .sof must be targeted on Stratix V devices in the .sof **File Properties**. This option must be turned on if you turned on the **Compression** option during .pmsf to .rbf file generation.

1.10.2.6. Enable Bitstream Decryption Option

The **Convert Programming Files** window provides the option in the .sof file properties to enable bitstream decryption during partial reconfiguration.

This option is available when converting base .sof to any supported programming file types, such as .rbf, .pof, and .jic.

The base .sof must have partial reconfiguration enabled and the base .sof generated from a design that has a PR Control Block instantiated, to view this option in the .sof **File Properties**. This option must be turned on if you wants to turn on the Generate encrypted bitstream option during .pmsf to .rbf file generation.

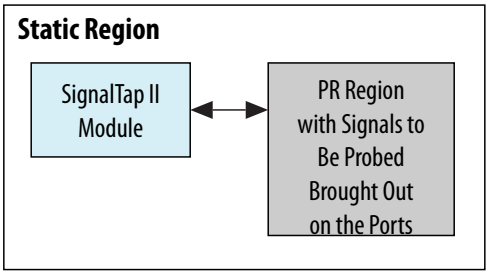
1.11. On-Chip Debug for PR Designs

You cannot instantiate a Signal Tap block inside a PR region. If you must monitor signals within a PR region for debug purposes, bring those signals to the ports of the PR region.

The Intel Quartus Prime software does not support the Incremental Signal Tap feature for PR designs. After you instantiate the Signal Tap block inside the static region, you must recompile your design. When you recompile your design, the static region may have a modified implementation and you must also recompile your PR revisions. If you modify an existing Signal Tap instance you must also recompile your entire design; base revision and reconfigurable revisions.

Figure 20. Using Signal Tap with a PR Design

You can instantiate the SignalTap II block in the static region of the design and probe the signals you want to monitor.



You can use other on-chip debug features in the Intel Quartus Prime software, such as the In-System Sources and Probes or Signal Probe, to debug a PR design. As in the case of SignalTap, In-System Sources and Probes can only be instantiated within the static region of a PR design. If you have to probe any signal inside the PR region, you must bring those signals to the ports of the PR region in order to monitor them within the static region of the design.

1.12. Partial Reconfiguration Known Limitations

There are restrictions that derive from hardware limitations in specific Stratix V devices.

The restrictions in the following sections apply only if your design uses M20K blocks as RAMs or ROMs in your PR project.

1.12.1. Memory Blocks Initialization Requirement for PR Designs

For a non-PR design, the power up value for the contents of a M20K RAM or a MLAB RAM are all set at zero. However, at the end of performing a partial reconfiguration, the contents of a M20K or MLAB memory block are unknown. You must intentionally initialize the contents of all the memory to zero, if required by the functionality of the design, and not rely upon the power on values.

1.12.2. M20K RAM Blocks in PR Designs

When your PR design uses M20K RAM blocks in Stratix V devices, there are some restrictions which limit how you utilize the respective memory blocks as ROMs or as RAMs with initial content.

Related Information

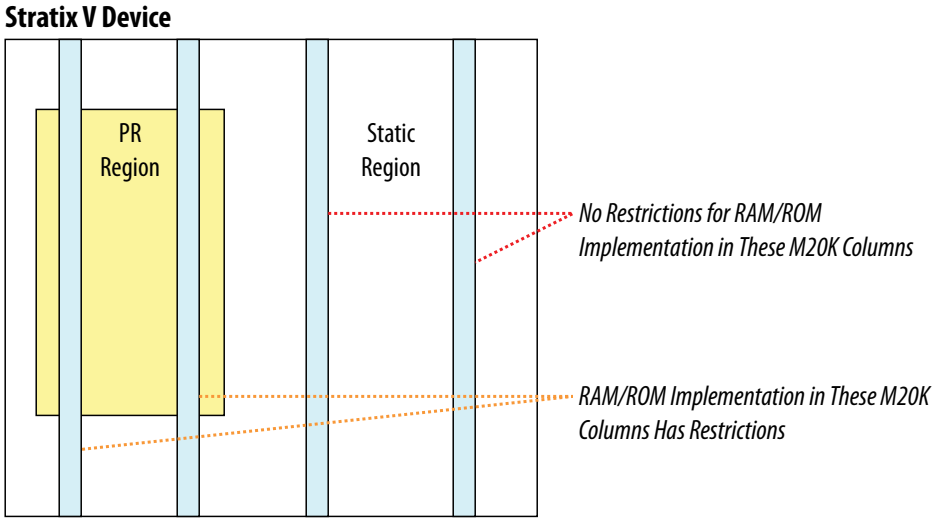
Implementing Memories with Initialized Content on page 43
If your design requires initialized memory content either as a ROM or a RAM inside a PR region, you must follow these guidelines.

1.12.2.1. Limitations When Using Stratix V Production Devices

These workarounds allow your design to use M20K blocks with PR.

Figure 21. Limitations for Using M20Ks in PR Regions

If you implement a M20K block in your PR region as a ROM or a RAM with initialized content, when the PR region is reconfigured, any data read from the memory blocks in static regions in columns that cross the PR region is incorrect.



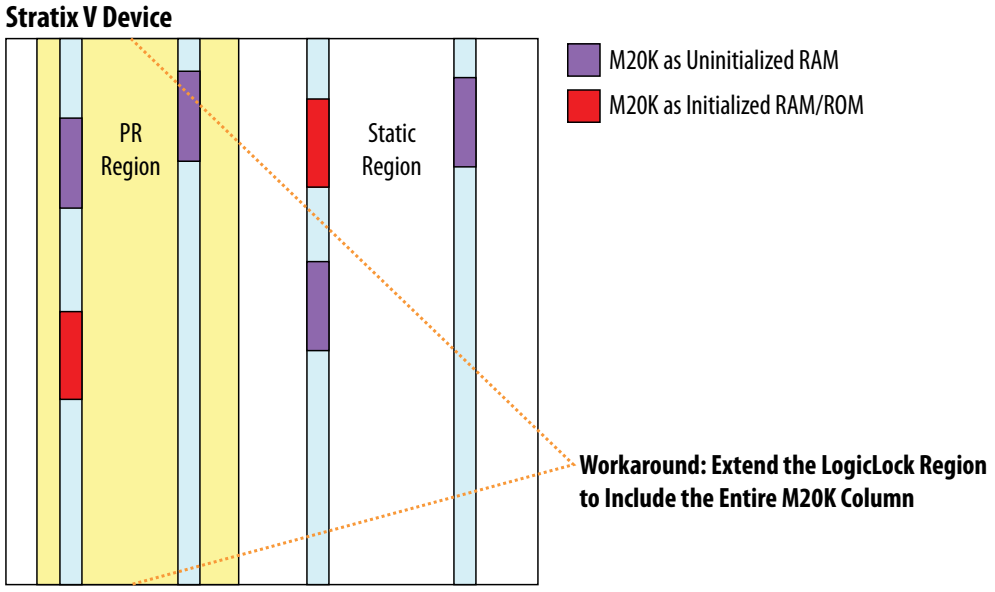
If the functionality of the static region depends on any data read out from M20K RAMs in the static region, the design will malfunction.

Use one of the following workarounds, which are applicable to both AND/OR and SCRUB modes of partial reconfiguration:

- Do not use ROMs or RAMs with initialized content inside PR regions.
- If this is not possible for your design, you can program the memory content for M20K blocks with a `.mif` using the suggested workarounds.
- Make sure your PR region extends vertically all the way through the device, in such a way that the M20K column lies entirely inside a PR region.

Figure 22. Workaround for Using M20Ks in PR Regions

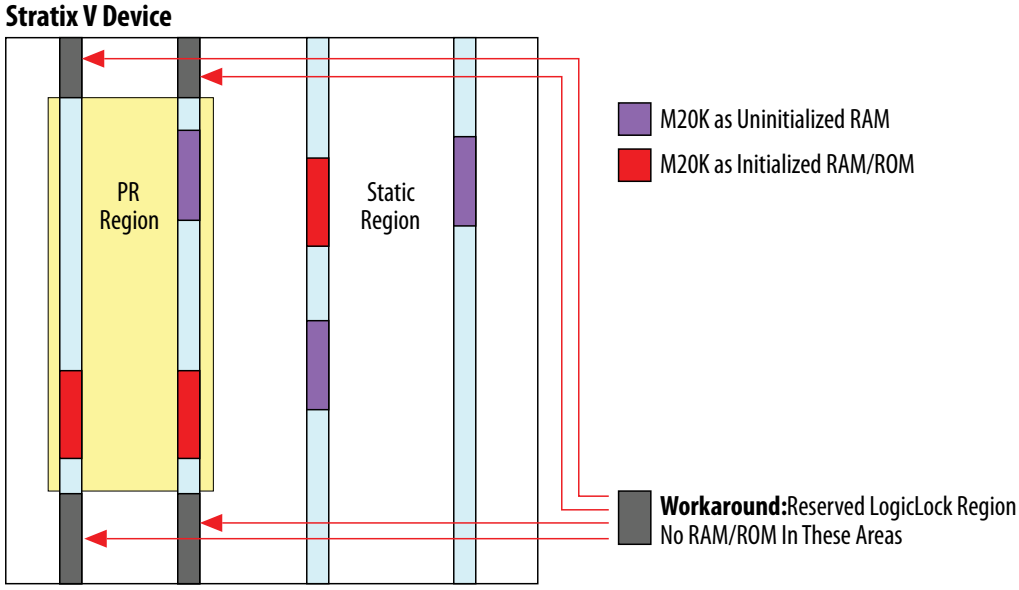
This figure shows the LogicLock region extended as a rectangle reducing the area available for the static region. However, you can create non-rectangular LogicLock regions to allocate the resources required for the partition more optimally. If saving area is a concern, extend the LogicLock region to include M20K columns entirely.



-

Figure 23. Alternative Workaround for Using M20Ks in PR Region

Using Reserved LogicLock Regions, block all the M20K columns that are not inside a PR region, but that are in columns above or below a PR region. In this case, you may choose to under-utilize M20K resources, in order to gain ROM functionality within the PR region.



For more information including a list of the Stratix V production devices, refer to the *Errata Sheet for Stratix V Devices*.

1.12.3. MLAB Blocks in PR designs

Stratix V devices include dual-purpose blocks called MLABs, which can be used to implement RAMs or LABs for user logic.

This section describes the restrictions while using MLAB blocks (sometimes also referred to as LUT-RAM) in Stratix V devices for your PR designs.

If your design uses MLABS as LUT RAM, you must use all available MLAB bits within the region.

Table 6. RAM Implementation Restrictions Summary

The following table shows a summary of the LUT-RAM Restrictions.

PR Mode	Type of memory in PR region	Stratix V Production
SCRUB mode	LUT RAM (no initial content)	OK
	LUT ROM and LUT RAM with your initial content	OK
AND/OR mode	LUT RAM (no initial content)	*While design is running:* Write 1s to all locations before partial reconfiguration *At compile time:* Explicitly initialize all memory locations in each new persona to 1 via initialization file (**. mif**).
	LUT ROM and LUT RAM with your initial content	No

If your design does not use any MLAB blocks as RAMs, the following discussion does not apply. The restrictions listed below are the result of hardware limitations in specific devices.

Limitations with Stratix V Production Devices

When using SCRUB mode:

- LUT-RAMs without initialized content, LUT-RAMs with initialized content, and LUT-ROMs can be implemented in MLABs within PR regions without any restriction.

When using AND/OR mode:

- LUT-RAMs with initialized content or LUT-ROMs cannot be implemented in a PR region.
- LUT-RAMs without initialized content in MLABs inside PR regions are supported with the following restrictions.
- MLAB blocks contain 640 bits of memory. The LUT RAMs in PR regions in your design must occupy all MLAB bits, you should not use partial MLABs.
- You must include control logic in your design with which you can write to all MLAB locations used inside PR region.
- Using this control logic, write '1' at each MLAB RAM bit location in the PR region before starting the PR process. This is to work around a false EDCRC error during partial reconfiguration.
- You must also specify a `.mif` that sets all MLAB RAM bits to '1' immediately after PR is complete.
- ROMs cannot be implemented in MLABs (LUT-ROMs).
- There are no restrictions to using MLABs in the static region of your PR design.

For more information, refer to the following documents in the *Stratix V Handbook*:

1.12.4. Implementing Memories with Initialized Content

If your Stratix V PR design implements ROMs, RAMs with initialization, or ROMs within the PR regions, using either M20K blocks or LUT-RAMs, then you must follow the following design guidelines to determine what is applicable in your case.

Table 7. Implementing Memory with Initialized Content in PR Designs

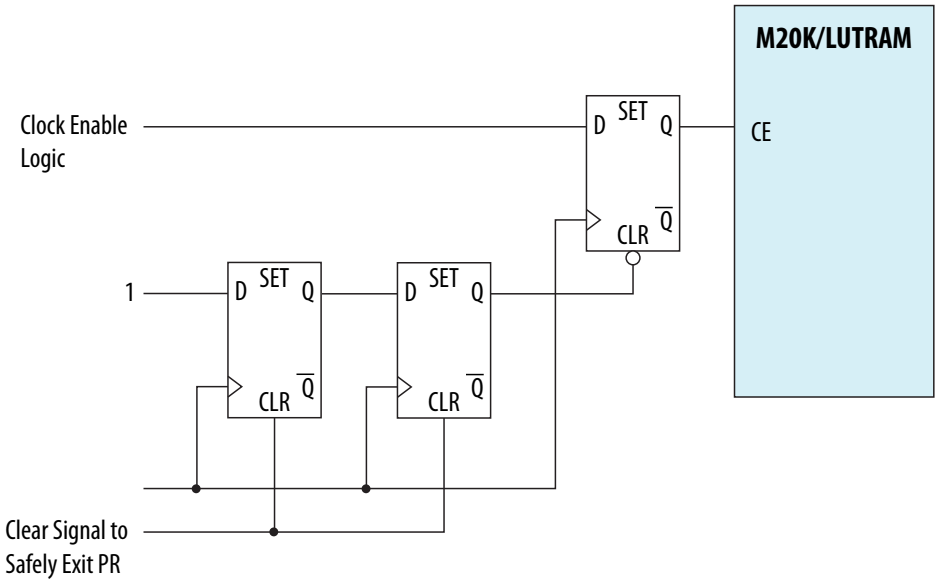
Mode		Production Devices	
		AND/OR	SCRUB
LUT-RAM without initialization	**Suggested Method**	*While design is running:* Write '1' to all locations before partial reconfiguration. *At compile time:* Explicitly initialize all memory locations in each new persona to '1' via initialization file (`.mif`)	No special method required

continued...

Mode		Production Devices	
		AND/OR	SCRUB
		Make sure no spurious write on PR entry (1)	
	Without Suggested Method	CRC Error	No special method required
LUT-RAM with initialization	**Suggested Method**	Not supported	Make sure no spurious write on PR exit (1)
	Without Suggested Method		Incorrect results
M20K without initialization	**Suggested Method**	No special method required	
	Without Suggested Method	No special method required	
M20K with initialization	**Suggested Method**	Use double PR cycle (2) Make sure no spurious write on PR exit (1)	No special method required
	Without Suggested Method	Incorrect results	No special method required

Figure 24. M20K/LUTRAM

To avoid spurious writes during PR entry and exit, implement the following clock enable circuit in the same PR region as the RAM.



The circuit depends on an active- high clear signal from the static region. Before entering PR, freeze this signal in the same manner as all PR inputs. Your host control logic should de-assert the clear signal as the final step in the PR process.

(1) Use the circuit shown in the **M20K/LUTRAM** figure to create clock enable logic to safely exit partial reconfiguration without spurious writes.

(2) Double partial reconfiguration is described in *Initializing M20K Blocks with a Double PR Cycle*

Related Information

Initializing M20K Blocks with a Double PR Cycle on page 45

1.12.5. Initializing M20K Blocks with a Double PR Cycle

When a PR region in your PR design contains an initialized M20K block and is reconfigured via AND/OR mode, your host logic must complete a double PR cycle, instead of a single PR cycle.

The PR IP has a `double_pr` input port, that must be asserted high when your PR region contains RAM blocks that must be initialized. The PR IP core handles the timing relations between the first and the second PR cycles of a Double PR operation. From your user logic, assert the `double_pr` signal when you assert the `pr_start` signal, and you deassert the `double_pr` signal when the `freeze` signal is deasserted by the PR IP. This method is also applicable in cases when the programming bitstream is compressed or encryted.

1.13. Document Revision History

Table 8. Document Revision History

Date	Version	Changes
2017.11.06	17.1.0	• Updated PR Bitstream Compression and Encryption topic to clarify FPGA family differences.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2015.05.04	15.0.0	• Correct Verilog HDL partial reconfiguration instantiation code example. • Added clear/set method to SCRUB mode option.
2015.12.15	14.1.0	Minor revisions to some topics to resolve design refinements: • Implementing Memories with Initialized Content • Instantiating the PR Control Block and CRC Block in Verilog HDL • Partial Reconfiguration Pins
June 2014	14.0.0	Minor updates to "Programming File Sizes for a Partial Reconfiguration Project" and code samples in "Freeze Logic for PR Regions" sections.
November 2013	13.1.0	Added support for merging multiple .msf and .pmsf files. Added support for PR Megafunction. Updated for revisions on timing requirements.
May 2013	13.0.0	Added support for encrypted bitstreams. Updated support for double PR.
November 2012	12.1.0	Initial release.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.

Explore more resources
Altera® Design Hub

altera
An Intel Company

Intel® Quartus® Prime Standard Edition User Guide

Third-party Simulation

Updated for Quartus® Prime Design Suite: **23.1**

Answers to Top FAQs:

Q ***What do I need for simulation?***
A Simulation Essential Elements on page 4

Q ***What simulators do you support?***
A Supported Simulators on page 19

Q ***What are the simulation stages?***
A Overview of Simulation Tool Flow on page 6

Q ***What are logical libraries?***
A Specifying Logical Libraries on page 9

Q ***How do I compile into libraries?***
A Compiling Files Into Library Directories on page 9

Q ***What is the simulation workflow?***
A Generic Simulation Workflow on page 16

Q ***What are the known issues and limitations?***
A Intel FPGA Support Forums: Simulation

Q ***Do you have training on simulation?***
A Intel FPGA Simulation Training

Contents

1. Intel FPGA Simulation Basics....4
1.1. Intel FPGA Simulation Essential Elements....4
1.2. Overview of Simulation Tool Flow....6
1.2.1. Compilation Stage....6
1.2.2. Elaboration Stage....7
1.2.3. Simulation Stage....8
1.3. Simulation Tool Flow....8
1.3.1. Specifying Logical Libraries....9
1.3.2. Compiling Files Into Library Directories....9
1.3.3. The Intel Quartus Prime Simulation Library....11
1.3.4. Understanding Elaboration....13
1.3.5. Commands To Configure and Run Simulation....15
1.3.6. Intel FPGA Simulation Generic Workflow....16
1.4. Supported Simulation Types....17
1.5. Supported Simulation Flows....17
1.6. Supported Hardware Description Languages....18
1.7. Supported Simulators....19
1.8. Using NativeLink Simulation (Intel Quartus Prime Standard Edition)....19
1.8.1. Setting Up NativeLink Simulation (Intel Quartus Prime Standard Edition)....20
1.8.2. Running RTL Simulation (NativeLink Flow)....20
1.8.3. Running Gate-Level Simulation (NativeLink Flow)....20
1.9. Intel FPGA Simulation Basics Revision History....21

2. Siemens EDA QuestaSim Simulator Support....22
2.1. Quick Start Example (QuestaSim with Verilog)....22
2.2. QuestaSim Simulator Guidelines....23
2.2.1. Passing Parameter Information from Verilog HDL to VHDL....23
2.2.2. Viewing Simulation Messages....23
2.2.3. Generating Signal Activity Data for Power Analysis....24
2.2.4. Viewing Simulation Waveforms....26
2.3. QuestaSim Simulation Setup Script Example....27
2.4. Sourcing QuestaSim Simulator Setup Scripts....27
2.5. Unsupported Features....28
2.6. Siemens EDA QuestaSim Simulator Support Revision History....29

3. Synopsys VCS and VCS MX Support....30
3.1. Quick Start Example (VCS with Verilog)....30
3.2. VCS and VCS MX Guidelines....30
3.2.1. Simulating Transport Delays....31
3.2.2. Disabling Timing Violation on Registers....31
3.2.3. Generating Power Analysis Files....32
3.3. VCS Simulation Setup Script Example....32
3.4. Sourcing Synopsys VCS MX Simulator Setup Scripts....33
3.5. Sourcing Synopsys VCS Simulator Setup Scripts....34
3.6. Synopsys VCS and VCS MX Support Revision History....36

4. Cadence Xcelium Parallel Simulator Support....37
4.1. Generating Simulator Setup Script Templates....37

4.2. Sourcing Cadence Xcelium Simulator Setup Scripts..... 37
4.3. Cadence Xcelium Parallel Simulator Support Revision History..... 40

5. Aldec Active-HDL and Riviera-PRO Support.....41
5.1. Quick Start Example (Active-HDL VHDL)..... 41
5.2. Aldec Active-HDL and Riviera-PRO Guidelines..... 42
5.2.1. Compiling SystemVerilog Files..... 42
5.2.2. Disabling Timing Violation on Registers..... 42
5.3. Using Simulation Setup Scripts..... 42
5.4. Sourcing Aldec ActiveHDL* or Riviera Pro* Simulator Setup Scripts..... 43
5.5. Aldec Active-HDL and Riviera-PRO * Support Revision History..... 46

A. Intel Quartus Prime Standard Edition User Guides.....47

1. Intel FPGA Simulation Basics

This chapter is a high-level explanation of Intel FPGA simulation basic concepts and workflows for all simulators that the Intel® 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. Intel FPGA Simulation Essential Elements

The following describes the essential elements required for Intel FPGA design simulation.

Design

An Intel 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. Intel Quartus Prime designs typically consist of several modules that you define in multiple HDL files. These files can include HDL files generated by Intel Quartus Prime tool, such as Intel 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.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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 Intel Quartus Prime software installation provides various simulation library files, as The Intel 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

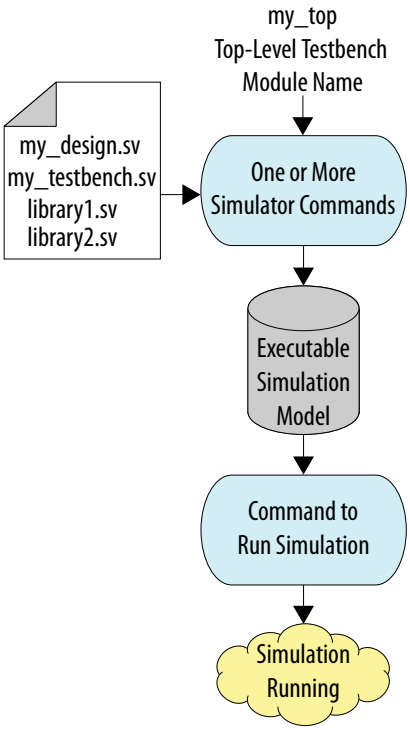
In order to simulate a design you must first generate an executable simulation model of the top-level testbench by running a set of simulator specific commands. You must then run the executable model to perform simulation. Running the executable model may require simulator specific commands. The executable 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 Intel 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.

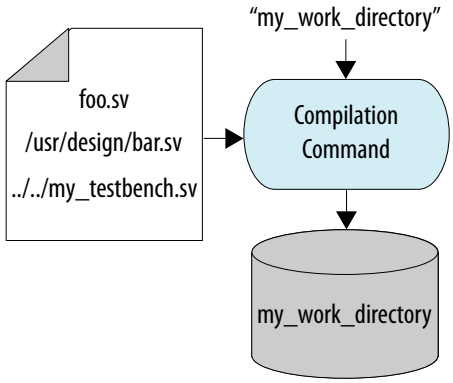
The library directory contains the definitions of all modules that are defined in the files that you compiled into the library. You use these module definitions in the elaboration stage. You may need to run multiple compilation commands to compile all the files into library directories.

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.

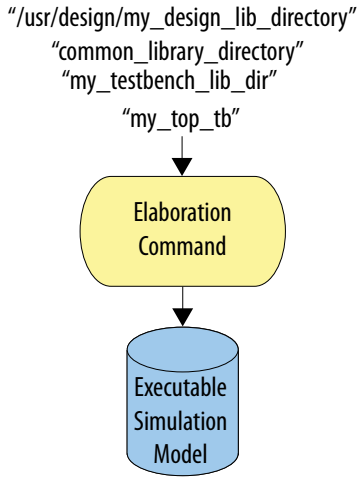
Figure 2. File and Directory Name Input to Simulator Specific Compilation Command



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



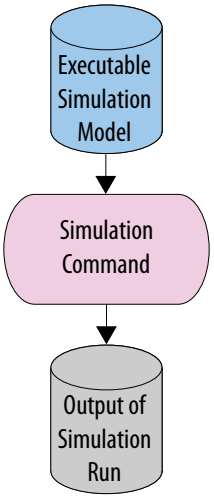
The output of the elaboration command is the executable simulation model for the top-level testbench. The executable simulation model comprises one or more simulator-specific files and directories. For more details about the elaboration stage, refer to Understanding Elaboration.

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 Intel 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 Intel Quartus Prime Simulation Library

The Intel Quartus Prime software includes the Intel 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 Intel 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,

continued...

Low-Level Blocks	Description
	and periphery 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 Intel Quartus Prime software generates. Since the Intel 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 Intel Quartus Prime Simulation Library Compiler

The Intel Quartus Prime Simulation Library Compiler is an Intel Quartus Prime software GUI and command-line tool that generates simulation scripts. You can use these scripts to automatically compile the Intel 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 Intel Quartus Prime Simulation Library Compiler in a terminal without launching the Intel 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 device family.

```
quartus_sh –simlib_comp -family cyclonev -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 Intel 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.

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 Intel 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.

Questa Intel FPGA Edition installation includes its own library mapping in the `modelsim.ini` file. This file maps the above logical library names to physical directories within the installation path. Therefore when you run elaboration command `vsim -L altera_ver` in Questa Intel FPGA Edition, the tool locates the correct physical library corresponding to logical library `altera_ver`.

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.

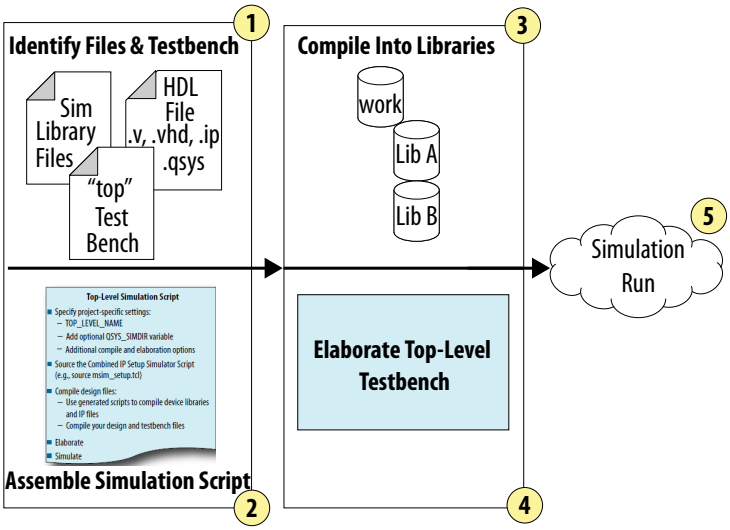
  The simulator writes the waveforms of these signals to a simulator proprietary database during simulation. You can view the waveforms in a GUI after simulation.

  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. Intel FPGA Simulation Generic Workflow

The following describes the high level workflow for simulation of any Intel Quartus Prime design using any supported simulator:

Figure 5. Generic Intel 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, as Commands to Configure and Run Simulation 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 Intel Quartus Prime software can generate simulator-specific simulation scripts to automate some of the simulation processing in your preferred simulation environment.

(1) In general, you can compile most HDL simulation files into the default `work` library.

The Intel 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 Intel 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 Types

You can run different types of simulation, depending on the stage of the Intel Quartus Prime design flow:

Table 2. 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 Intel Quartus Prime software. These RTL files typically also include the files that the Intel 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 Intel Quartus Prime Synthesis
Post-Synthesis Simulation (Gate-Level)	The Intel 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 Intel 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.	Must perform after Intel Quartus Prime synthesis
Post-Fit Simulation (Gate-Level)	The Intel 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 Intel Quartus Prime Fitter

Note: the Intel Quartus Prime software supports post-fit functional simulation, but does not support post-fit timing simulation.

1.5. Supported Simulation Flows

The Intel Quartus Prime software supports scripted and specialized simulation flows.

(2) You must first convert the non-HDL files to HDL files prior to simulation

Table 3. 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 Intel Quartus Prime-generated IP simulation setup scripts. The Intel Quartus Prime software can generate a combined simulator setup script for all IP cores, for each supported simulator.
NativeLink Simulation Flow	NativeLink automates Intel Quartus Prime integration with your EDA simulator. Setup NativeLink to generate simulation scripts, compile simulation libraries, and automatically launch your simulator following design compilation. Specify your own compilation, elaboration, and simulation scripts for testbench and simulation model files. Do not use NativeLink if you require direct control over every aspect of simulation.
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 (Standard) Systems* in *Intel Quartus Prime Standard Edition User Guide: Platform Designer*. • For simulation of the Nios® V processor, refer to the *Nios V Embedded Processor Design Handbook*.

1.6. Supported Hardware Description Languages

The Intel Quartus Prime software provides the following hardware description language (HDL) support for EDA simulators.

Table 4. 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 Intel FPGA simulation libraries. • Many of the Intel 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 Intel FPGA 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 Intel 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 Intel 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 Intel Quartus Prime software, generate a gate-level Verilog HDL or VHDL simulation netlist, and perform simulation on the gate-level netlist.

1.7. Supported Simulators

The Intel Quartus Prime software supports the following EDA simulator versions for RTL and gate-level simulation.

Table 5. Intel 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*	2023.04.082	Windows, Linux, 64-bit	No
Cadence	Xcelium* Parallel Simulator	23.03.003	Linux 64-bit	Yes
Intel FPGA	Questa Intel FPGA Edition	2023.3	Windows, Linux, 64-bit	Yes
Siemens EDA	QuestaSim* Simulator(3)	2023.2	Windows, Linux, 64-bit	Yes
Synopsys*	VCS*, VCS MX	U/2023.03-1	Linux 64-bit	Yes

Table 6. Intel 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
Intel	Questa Intel FPGA Edition	2023.3	Windows, Linux
Siemens EDA	QuestaSim	2023.2	Windows, Linux
Synopsys	VCS VCS MX	U/2023.03-1	Linux

Related Information

- Questa Intel FPGA Edition Simulation User Guide
- IBIS Models for Intel FPGA Devices

1.8. Using NativeLink Simulation (Intel Quartus Prime Standard Edition)

The NativeLink feature integrates your EDA simulator with the Intel Quartus Prime Standard Edition software by automating the following:

- Generation of simulator-specific files and simulation scripts.
- Compilation of simulation libraries.
- Launches your simulator automatically following Intel Quartus Prime Analysis & Elaboration, Analysis & Synthesis, or after a full compilation.

(3) QuestaSim is the generic name for Questa Core and Questa Prime simulators from Siemens EDA.

1.8.1. Setting Up NativeLink Simulation (Intel Quartus Prime Standard Edition)

Before running NativeLink simulation, specify settings for your simulator in the Intel Quartus Prime software.

To specify NativeLink settings in the Intel Quartus Prime Standard Edition software, follow these steps:

1. Open an Intel Quartus Prime Standard Edition project.
2. Click **Tools > Options** and specify the location of your simulator executable file.

Table 7. Execution Paths for EDA Simulators

Simulator	Path
Questa Intel FPGA Edition	*<drive letter>*:*<simulator install path>*\ (Windows) /*<simulator install path>*/bin (Linux)
Siemens EDA QuestaSim	*<drive letter>*:*<simulator install path>*\ (Windows) *<simulator install path>*/bin (Linux)
Synopsys VCS/VCS MX	*<simulator install path>*/bin (Linux)
Cadence Incisive Enterprise/ Xcelium	*<simulator install path>*/tools/bin (Linux)
Aldec Active-HDL Aldec Riviera-PRO	*<drive letter>*:*<simulator install path>*\bin (Windows) *<simulator install path>*/bin (Linux)

3. Click **Assignments ➤ Settings** and specify options on the **Simulation** page and the **More NativeLink Settings** dialog box. Specify default options for simulation library compilation, netlist and tool command script generation, and for launching RTL or gate-level simulation automatically following compilation.
4. If your design includes a testbench, turn on **Compile test bench**. Click **Test Benches** to specify options for each testbench. Alternatively, turn on **Use script to compile testbench** and specify the script file.
5. To use a script to setup a simulation, turn on **Use script to setup simulation**.

1.8.2. Running RTL Simulation (NativeLink Flow)

To run RTL simulation using the NativeLink flow, follow these steps:

1. Set up the simulation environment.
2. Click **Processing ➤ Start ➤ Start Analysis & Elaboration**.
3. Click **Tools ➤ Run Simulation ➤ Run Simulation Tool**. NativeLink compiles simulation libraries and launches and runs your RTL simulator automatically according to the NativeLink settings.
4. Review and analyze the simulation results in your simulator. Correct any functional errors in your design. If necessary, re-simulate the design to verify correct behavior.

1.8.3. Running Gate-Level Simulation (NativeLink Flow)

To run gate-level simulation with the NativeLink flow, follow these steps:

1. Set up the simulation environment.
2. To generate only a functional (rather than timing) gate-level netlist, click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ More EDA Netlist Writer Settings**. Turn on **Generate netlist for functional simulation only**.
3. To synthesize the design, follow one of these steps:
   - To generate a post-fit functional or post-fit timing netlist and then automatically simulate your design according to your NativeLink settings, Click **Processing ➤ Start Compilation**. Skip to step 6.
   - To synthesize the design for post-synthesis functional simulation only, click **Processing ➤ Start ➤ Start Analysis & Synthesis**.
4. To generate the simulation netlist, click **Start ➤ Start EDA Netlist Writer**.
5. Click **Tools ➤ Run Simulation Tool ➤ Gate Level Simulation**.
6. Review and analyze the simulation results in your simulator. Correct any unexpected or incorrect conditions found in your design. Simulate the design again until you verify correct behavior.

1.9. Intel FPGA Simulation Basics Revision History

Document Version	Intel Quartus Prime Version	Changes
2024.02.05	23.1	• Replaced all content in chapter with newly developed content more suitable for basic understanding of FPGA design simulation. • Replaced "Mentor Graphics" with "Siemens EDA" to reflect current company name. • Updated simulator versions supported and provided link to other resources in *Supported Simulators* topic.
2022.11.07	22.1	• Updated simulator versions supported and provided link to other resources in *Simulator Support* topic. • Replaced support for Cadence Incisive Enterprise (ncsim) simulator with Xcelium simulator support throughout. • Removed support for ModelSim - Intel FPGA Edition simulator throughout. • Added precompiled libraries information 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.
2018/09/24	18.1	• Removed *Scripting IP Simulation* and *Generating a Combined Simulation Script* topics. These features are supported only for Intel Arria 10 devices in Intel Quartus Prime Standard Edition. • Added link to *Scripting IP Simulation* in the *Introduction to Intel FPGA IP Cores*.

2. Siemens EDA QuestaSim Simulator Support

This chapter provides guidelines for simulation of Intel 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 Intel 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:
   - Run NativeLink RTL simulation to compile required design files, simulation models, and run your simulator. Verify results in your simulator. If you complete this step you can ignore the remaining steps.
   - 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.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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 Intel 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 Intel 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:

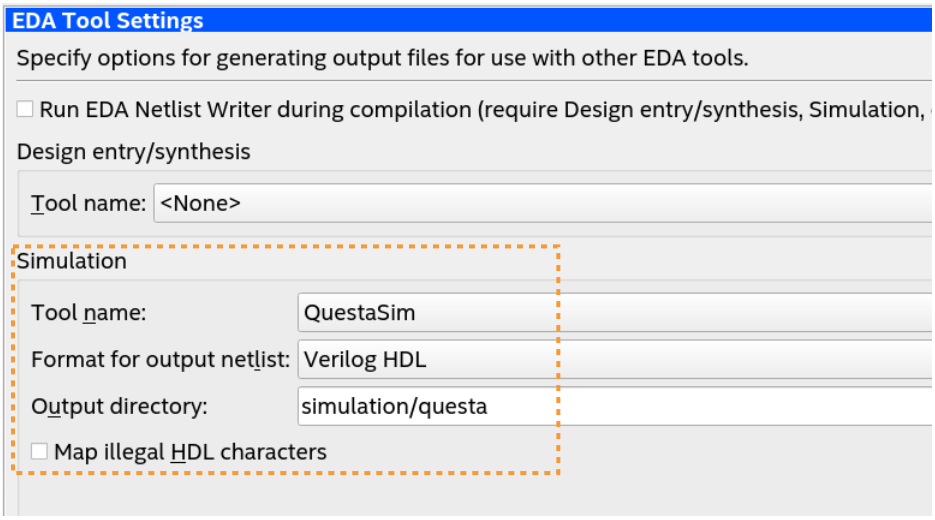
```
# vcom Message # 1136:
# The specified name was referenced but was not found. This indicates
# that either the name specified does not exist or is not visible at
# this point in the code.
```

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 6. EDA Tool Settings for Simulation



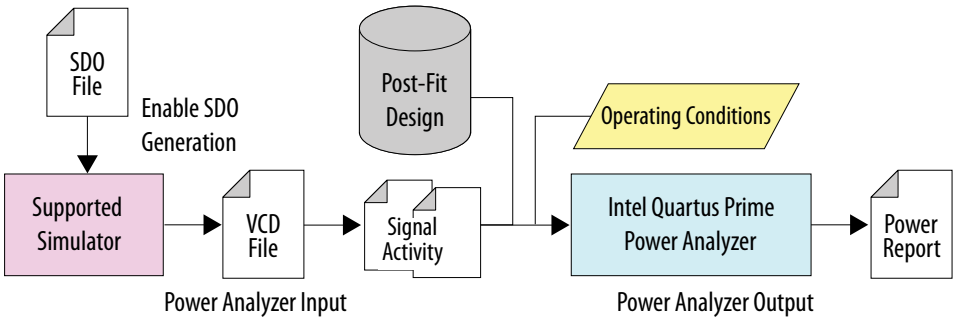
3. Turn on **Map illegal HDL characters**. This setting directs the EDA Netlist Writer to map illegal characters for VHDL or Verilog HDL, and results in more accurate data for power analysis.
4. Click the **Power Analyzer Settings** page.
5.
6. To specify a `.vcd` for power analysis, click **Add** and 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 Analyzer** tab. View the toggle rates in the power analysis results.

Note: To improve accuracy of power analysis, the Intel 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 Questa Intel FPGA Edition) for Intel Stratix® 10 designs. The EDA Netlist Writer does not currently support .sdo file generation for any other simulator or device family.

2.2.3.1. Generating Standard Delay Output for Power Analysis

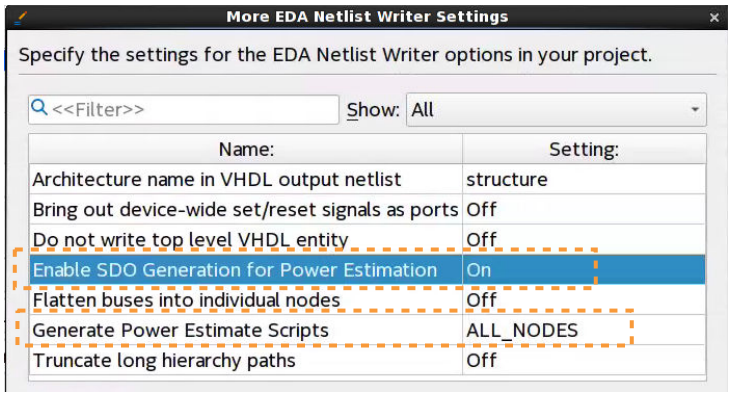
To improve accuracy of power analysis, you can generate a Standard Delay Output (.sdo) file that includes back-annotated delay estimates for QuestaSim simulation. QuestaSim simulation can then output a more accurate .vcd for use as power analysis input. You must run **Fitter (Finalize)** before generating the .sdo.

Figure 7. 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 8. 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: (4)

```
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.

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.

(4) Specify the full hierarchical path in the testbench, not just the instance name. For example, specify a|b|c, not just c.

2.3. QuestaSim Simulation Setup Script Example

The Intel 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.

2.4. 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
```

2.5. Unsupported Features

The Intel 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.

2.6. Siemens EDA QuestaSim Simulator Support Revision History

Document Version	Intel Quartus Prime Version	Changes
2024.02.05	23.1	• 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*. • Replaced "Mentor Graphics" with "Siemens EDA" to reflect current company name. • Updated tool name to QuestaSim in *Generating Standard Delay Output for Power Analysis* topic.
2022.11.07	22.1	• Added support for Questa* Intel FPGA Edition simulator throughout. • Replaced "Mentor Graphics" with "Siemens EDA" to reflect current company name. • Removed support for ModelSim - Intel FPGA Edition simulator throughout. • Replaced support for Cadence Incisive Enterprise (ncsim) simulator with Xcelium simulator support throughout. • Revised *Using Questa* Intel FPGA Edition Precompiled Libraries* topic. • Corrected syntax error in *ModelSim Simulation Setup Script* Example
2017.05.08	18.1	• Changed title to *ModelSim - Intel FPGA Edition, ModelSim®, and QuestaSim Support** • Stated no support for Intel Arria® 10 timing simulation in *Simulating Transport Delays* and *Disabling Timing Violations on Registers* topics. • 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 Intel Quartus Prime design flow. This document provides guidelines for simulation of Intel 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 Intel 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:
   - Run NativeLink RTL simulation to compile required design files, simulation models, and run your simulator. Verify results in your simulator. If you complete this step you can ignore the remaining steps.
   - 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:

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- 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.

3.2.1. Simulating Transport Delays

By default, the VCS and VCS MX software filter out all pulses that are shorter than the propagation delay between primitives. Turning on the **transport delay** options in the VCS and VCS MX software prevents the simulator from filtering out these pulses. Intel Arria 10 devices do not support timing simulation.

Table 8. Transport Delay Simulation Options (VCS and VCS MX)

Option	Description
`+transport_path_delays`	Use when simulation pulses are shorter than the delay in a gate-level primitive. You must include the `+pulse_e/number` and `+pulse_r/number` options.
`+transport_int_delays`	Use when simulation pulses are shorter than the interconnect delay between gate-level primitives. You must include the `+pulse_int_e/number` and `+pulse_int_r/number` options.

Note: The `+transport_path_delays` and `+transport_path_delays` options apply automatically during NativeLink gate-level timing simulation.

The following VCS and VCS MX software command runs a post-synthesis simulation:

```
vcs -R <testbench>.v <gate-level netlist>.v -v <Intel FPGA device family \
library>.v +transport_int_delays +pulse_int_e/0 +pulse_int_r/0 \
+transport_path_delays +pulse_e/0 +pulse_r/0
```

3.2.2. Disabling Timing Violation on Registers

In certain situations, you may want to ignore timing violations on registers and disable the "X" propagation that occurs. For example, this technique may be helpful to eliminate timing violations in internal synchronization registers in asynchronous clock-domain crossing. Intel Arria 10 devices do not support timing simulation. Intel Arria 10 devices do not support timing simulation.

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 Intel Quartus Prime Settings File (`.qsf`).

```
set_instance_assignment -name X_ON_VIOLATION_OPTION OFF -to \
<register_name>
```

3.2.3. Generating Power Analysis Files

You can generate a Verilog Value Change Dump File (.vcd) for power analysis in the Intel Quartus Prime software, and then run the **.vcd** from the VCS software. Use this **.vcd** for power analysis in the Intel Quartus Prime power analyzer.

To generate and use a **.vcd** for power analysis, follow these steps:

1. In the Intel Quartus Prime software, click **Assignments ➤ Settings**.
2. Under **EDA Tool Settings**, click **Simulation**.
3. Turn on **Generate Value Change Dump file script**, specify the type of output signals to include, and specify the top-level design instance name in your testbench.
4. Click **Processing ➤ Start Compilation**.
5. Use the following command to include the script in your testbench where the design under test (DUT) is instantiated:
   `include <revision_name>_dump_all_vcd_nodes.v`

   Note: Include the script within the testbench module block. If you include the script outside of the testbench module block, syntax errors occur during compilation.
6. Run the simulation with the VCS command. Exit the VCS software when the simulation is finished and the *<revision_name>***.vcd** file is generated in the simulation directory.

3.3. VCS Simulation Setup Script Example

The Intel 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	Intel Quartus Prime Version	Changes
2024.02.05	23.1	• 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*.
2017.11.06	17.1	• Stated no support for Intel Arria 10 timing simulation in Simulating Transport Delays and Disabling Timing Violations on Registers topics. • Added Simulation Library Compiler details and another step to Quick Start Example

4. Cadence Xcelium Parallel Simulator Support

You can include your supported EDA simulator in the *Intel Quartus Prime* design flow. This chapter provides specific guidelines for simulation of *Intel Quartus Prime* designs with the Cadence Incisive Enterprise (IES) software.

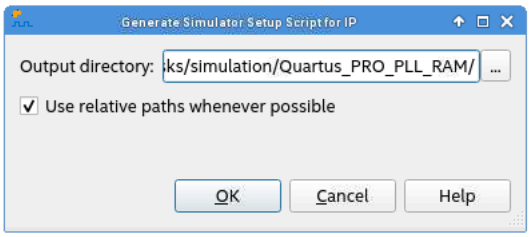
4.1. Generating Simulator Setup Script Templates

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 9. Generate Simulator Setup Script for IP Dialog Box



4.2. 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
```

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

```
# # 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.

4.3. Cadence Xcelium Parallel Simulator Support Revision History

Document Version	Intel Quartus Prime Version	Changes
2024.02.05	23.1	• Updated chapter to reflect end of support for ModelSim.
2022.11.07	22.1	• Replaced support for Cadence Incisive Enterprise (ncsim) simulator with Xcelium simulator support throughout. • Renamed chapter for Xcelium Parallel Simulator support. • Added Xcelium command-line support.
2017.11.06	17.1	• Stated no support for Intel Arria 10 timing simulation in *Simulating Transport Delays* and *Disabling Timing Violations on Registers* topics. • Added Simulation Library Compiler details and another step to *Quick Start Example*.

5. Aldec Active-HDL and Riviera-PRO Support

You can include your supported EDA simulator in the Intel Quartus Prime design flow. This chapter provides specific guidelines for simulation of Intel Quartus Prime designs with the Aldec Active-HDL or Riviera-PRO software.

5.1. 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 Intel 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:
   - Run NativeLink RTL simulation to compile required design files, simulation models, and run your simulator. Verify results in your simulator. If you complete this step you can ignore the remaining steps.
   - 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>
   ```

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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.

5.2. 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.

5.2.1. Compiling SystemVerilog Files

If your design includes multiple SystemVerilog files, you must compile the System Verilog 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.

5.2.2. Disabling Timing Violation on Registers

In certain situations, you may want to ignore timing violations on registers and disable the "X" propagation that occurs. For example, this technique may be helpful to eliminate timing violations in internal synchronization registers in asynchronous clock-domain crossing. Intel Arria 10 devices do not support timing simulation.

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 Intel Quartus Prime Settings File (`.qsf`).

```
set_instance_assignment -name X_ON_VIOLATION_OPTION OFF -to \
<register_name>
```

5.3. Using Simulation Setup Scripts

The Intel 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 Intel Quartus Prime software generates for use with the QuestaSim simulator.

5.4. 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
   ```

5.5. Aldec Active-HDL and Riviera-PRO * Support Revision History

Document Version	Intel Quartus Prime Version	Changes
2024.02.05	23.1	• Updated chapter to reflect end of support for generation of ModelSim files in favor of QuestaSim. • Updated script content in *Sourcing Aldec ActiveHDL or Riviera Pro* Simulator Setup Scripts* topic. • Revised name of Questa Intel FPGA Edition and QuestaSim for latest guidelines throughout.
2017.11.06	17.1	• Stated no support for Intel Arria 10 timing simulation in Simulating Transport Delays and Disabling Timing Violations on Registers topics. • Added Simulation Library Compiler details to *Quick Start Example*

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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.

© Altera Corporation. Altera, the Altera logo, the 'a' logo, and other Altera marks are trademarks of Altera Corporation. Altera and Intel warrant performance of its FPGA and semiconductor products to current specifications in accordance with Altera's or Intel's standard warranty as applicable, but reserves the right to make changes to any products and services at any time without notice. Altera and Intel assume no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to inwriting by Altera or Intel. Altera and Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Third-party Synthesis

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Synopsys Synplify* Support....4
1.1. About Synplify Support....4
1.2. Design Flow....4
1.3. Hardware Description Language Support....6
1.4. Intel Device Family Support....6
1.5. Tool Setup....6
1.5.1. Specifying the Intel Quartus Prime Software Version....6
1.5.2. Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration....6
1.6. Synplify Software Generated Files....8
1.7. Design Constraints Support....9
1.7.1. Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script....10
1.7.2. Passing Timing Analyzer SDC Timing Constraints to the Intel Quartus Prime Software....10
1.8. Simulation and Formal Verification....11
1.9. Synplify Optimization Strategies....11
1.9.1. Using Synplify Premier to Optimize Your Design....12
1.9.2. Using Implementations in Synplify Pro or Premier....12
1.9.3. Timing-Driven Synthesis Settings....12
1.9.4. FSM Compiler....14
1.9.5. Optimization Attributes and Options....15
1.9.6. Intel-Specific Attributes....17
1.10. Guidelines for Intel FPGA IP Cores and Architecture-Specific Features....18
1.10.1. Instantiating Intel FPGA IP Cores with the IP Catalog....19
1.10.2. Including Files for Intel Quartus Prime Placement and Routing Only....23
1.10.3. Inferring Intel FPGA IP Cores from HDL Code....23
1.11. Incremental Compilation and Block-Based Design....28
1.11.1. Design Flow for Incremental Compilation....29
1.11.2. Creating a Design with Separate Netlist Files for Incremental Compilation....29
1.11.3. Using MultiPoint Synthesis with Incremental Compilation....30
1.11.4. Creating Multiple .vqm Files for a Incremental Compilation Flow With Separate Synplify Projects....34
1.11.5. Performing Incremental Compilation in the Intel Quartus Prime Software....38
1.12. Synopsys Synplify* Support Revision History....39

2. Mentor Graphics Precision* Synthesis Support....41
2.1. About Precision RTL Synthesis Support....41
2.2. Design Flow....41
2.2.1. Timing Optimization....44
2.3. Intel Device Family Support....44
2.4. Precision Synthesis Generated Files....44
2.5. Creating and Compiling a Project in the Precision Synthesis Software....45
2.6. Mapping the Precision Synthesis Design....45
2.6.1. Setting Timing Constraints....46
2.6.2. Setting Mapping Constraints....46
2.6.3. Assigning Pin Numbers and I/O Settings....46
2.6.4. Assigning I/O Registers....47

2.6.5. Disabling I/O Pad Insertion....47
2.6.6. Controlling Fan-Out on Data Nets....48
2.7. Synthesizing the Design and Evaluating the Results....48
2.7.1. Obtaining Accurate Logic Utilization and Timing Analysis Reports....49
2.8. Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration....49
2.8.1. Running the Intel Quartus Prime Software from within the Precision Synthesis Software....49
2.8.2. Running the Intel Quartus Prime Software Manually Using the Precision Synthesis-Generated Tcl Script....50
2.8.3. Using the Intel Quartus Prime Software to Run the Precision Synthesis Software....50
2.8.4. Passing Constraints to the Intel Quartus Prime Software....51
2.9. Guidelines for Intel FPGA IP Cores and Architecture-Specific Features....54
2.9.1. Instantiating IP Cores With IP Catalog-Generated Verilog HDL Files....54
2.9.2. Instantiating IP Cores With IP Catalog-Generated VHDL Files....55
2.9.3. Instantiating Intellectual Property With the IP Catalog and Parameter Editor....55
2.9.4. Instantiating Black Box IP Functions With Generated Verilog HDL Files....56
2.9.5. Instantiating Black Box IP Functions With Generated VHDL Files....56
2.9.6. Inferring Intel FPGA IP Cores from HDL Code....57
2.10. Incremental Compilation and Block-Based Design....61
2.10.1. Creating a Design with Precision RTL Plus Incremental Synthesis....61
2.10.2. Creating Multiple Mapped Netlist Files With Separate Precision Projects or Implementations....63
2.10.3. Creating Black Boxes to Create Netlists....64
2.10.4. Creating Intel Quartus Prime Projects for Multiple Netlist Files....66
2.10.5. Hierarchy and Design Considerations....67
2.11. Mentor Graphics Precision* Synthesis Support Revision History....67

A. Intel Quartus Prime Standard Edition User Guides....69

1. Synopsys Synplify* Support

1.1. 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.
- Exporting designs and constraints to the 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

1.2. Design Flow

The following steps describe a basic Intel Quartus Prime software design flow using 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:

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

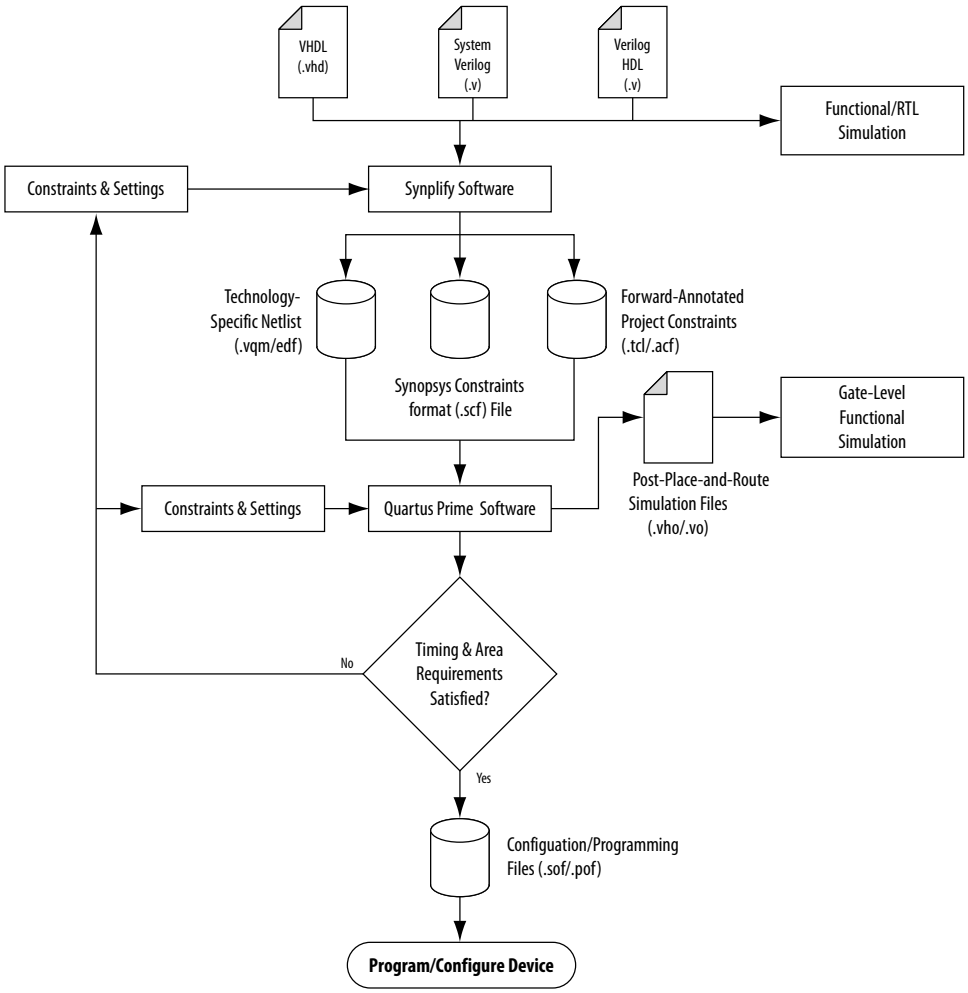
ISO 9001:2015 Registered

- 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

- Running the Intel Quartus Prime Software from within the Synplify Software on page 7
- Synplify Software Generated Files on page 8
- Design Constraints Support on page 9

1.3. 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 18

1.4. Intel Device Family Support

Support for newly released device families may require an overlay. Contact Synopsys for more information.

Related Information

Synopsys Website

1.5. Tool Setup

1.5.1. 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 1. Specifying Intel Quartus Prime Software Version at the Command Line

```
set_option -quartus_version <version number>
```

1.5.2. Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration

The NativeLink feature in the Intel Quartus Prime software facilitates the seamless transfer of information between the Intel Quartus Prime software and EDA tools, and allows you to run other EDA design entry or synthesis, simulation, and timing analysis

tools automatically from within the Intel Quartus Prime software. After a design is synthesized in the Synplify software, a `.vqm` netlist file, an `.scf` file for Timing Analyzer timing constraints, and `.tcl` files are used to import the design into the Intel Quartus Prime software for place-and-route. You can run the Intel Quartus Prime software from within the Synplify software or as a stand-alone application. After you import the design into the Intel Quartus Prime software, you can specify different options to further optimize the design.

Note: When you are using NativeLink integration, the path to your project must not contain empty spaces. The Synplify software uses Tcl scripts to communicate with the Intel Quartus Prime software, and the Tcl language does not accept arguments with empty spaces in the path.

Use NativeLink integration to integrate the Synplify software and Intel Quartus Prime software with a single GUI for both synthesis and place and-route operations. NativeLink integration allows you to run the Intel Quartus Prime software from within the Synplify software GUI, or to run the Synplify software from within the Intel Quartus Prime software GUI.

1.5.2.1. Running the Intel Quartus Prime Software from within the Synplify Software

To run the Intel Quartus Prime software from within the Synplify software, you must set the *QUARTUS_ROOTDIR* environment variable to the Intel Quartus Prime software installation directory located in <*Intel Quartus Prime system directory*>\altera\<*version number*>**\quartus**. You must set this environment variable to use the Synplify and Intel Quartus Prime software together. Synplify also uses this variable to open the Intel Quartus Prime software in the background and obtain detailed information about the Intel FPGA IP cores used in the design.

For the Windows operating system, do the following:

1. Point to **Start**, and click **Control Panel**.
2. Click **System** >**Advanced system settings** >**Environment Variables**.
3. Create a *QUARTUS_ROOTDIR* system variable.

For the Linux operating system, do the following:

- Create an environment variable *QUARTUS_ROOTDIR* that points to the <*home directory*>/altera <*version number*> location.

You can create new place and route implementations with the **New P&R** button in the Synplify software GUI. Under each implementation, the Synplify Pro software creates a place-and-route implementation called **pr_**<*number*> **Altera Place and Route**. To run the Intel Quartus Prime software in command-line mode after each synthesis run, use the text box to turn on the place-and-route implementation. The results of the place-and-route are written to a log file in the **pr_** <*number*> directory under the current implementation directory.

You can also use the commands in the Intel Quartus Prime menu to run the Intel Quartus Prime software at any time following a successful completion of synthesis. In the Synplify software, on the Options menu, click **Intel Quartus Prime** and then choose one of the following commands:

- **Launch Quartus** —Opens the Intel Quartus Prime software GUI and creates a Intel Quartus Prime project with the synthesized output file, forward-annotated timing constraints, and pin assignments. Use this command to configure options for the project and to execute any Intel Quartus Prime commands.
- **Run Background Compile**—Runs the Intel Quartus Prime software in command-line mode with the project settings from the synthesis run. The results of the place-and-route are written to a log file.

The <*project_name*>**_cons.tcl** file is used to set up the Intel Quartus Prime project and directs the <*project_name*>**.tcl** file to pass constraints from the Synplify software to the Intel Quartus Prime software. By default, the <*project_name*>**.tcl** file contains device, timing, and location assignments. The <*project_name*>**.tcl** file contains the command to use the Synplify-generated **.scf** constraints file with the Timing Analyzer.

Related Information

Design Flow on page 4

1.5.2.2. Using the Intel Quartus Prime Software to Run the Synplify Software

You can set up the Intel Quartus Prime software to run the Synplify software for synthesis with NativeLink integration. This feature allows you to use the Synplify software to quickly synthesize a design as part of a standard compilation in the Intel Quartus Prime software. When you use this feature, the Synplify software does not use any timing constraints or assignments that you have set in the Intel Quartus Prime software.

Note: For best results, Synopsys recommends that you set constraints in the Synplify software and use a Tcl script to pass these constraints to the Intel Quartus Prime software, instead of opening the Synplify software from within the Intel Quartus Prime software.

To set up the Intel Quartus Prime software to run the Synplify software, do the following:

1. On the Tools menu, click **Options**.
2. In the **Options** dialog box, click **EDA Tool Options** and specify the path of the Synplify or Synplify Pro software under **Location of Executable**.

Running the Synplify software with NativeLink integration is supported on both floating network and node-locked fixed PC licenses. Both types of licenses support batch mode compilation.

1.6. Synplify Software Generated Files

During synthesis, the Synplify software produces several intermediate and output files.

Table 1. 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`(1)	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`(2)	Synthesis Report file.

Related Information

Design Flow on page 4

1.7. 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.

---

(1) If your design uses the Classic Timing Analyzer for timing analysis in the Intel Quartus Prime software versions 10.0 and earlier, the Synplify software generates timing constraints in the Tcl Constraints File (`.tcl`). If you are using the Intel Quartus Prime software versions 10.1 and later, you must use the Timing Analyzer for timing analysis.

(2) 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.

Related Information

- Design Flow on page 4
- Synplify Optimization Strategies on page 11

1.7.1. 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 **Utility Windows** 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
```

1.7.2. 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 12

1.7.2.1. 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.

1.7.2.2. 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.

1.7.2.3. 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.

1.7.2.4. 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.

1.8. 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.

1.9. 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 9

1.9.1. 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.

1.9.2. 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.

1.9.3. 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 10

1.9.3.1. 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.

1.9.3.2. 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.

1.9.3.3. 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.

1.9.3.4. 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.

1.9.3.5. 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.

1.9.4. 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 2. `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 2. 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.

1.9.4.1. 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.

1.9.5. Optimization Attributes and Options

1.9.5.1. 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.

1.9.5.2. 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.

1.9.5.3. 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.

1.9.5.4. 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.

1.9.5.5. 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**.

1.9.5.6. 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**.

1.9.5.7. 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 3. Input and Output Register Delay

```
define_reg_input_delay {<register>} -route <delay in ns>
define_reg_output_delay {<register>} -route <delay in ns>
```

1.9.5.8. 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 */ ;
```

1.9.5.9. 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 4. 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.*

1.9.6. 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.

1.9.6.1. 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 5. 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";
```

1.9.6.2. 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**.

1.9.6.3. 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.

1.10. 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 6

1.10.1. 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 6
- Using the Intel Quartus Prime Software to Run the Synplify Software on page 8

1.10.1.1. 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.

1.10.1.2. 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.

1.10.1.3. 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.

Related Information

- Including Files for Intel Quartus Prime Placement and Routing Only on page 23
- 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.

1.10.1.4. 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 23

1.10.1.5. 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 6. 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
```

1.10.1.6. 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 7. 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;
```

1.10.1.7. 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 8. 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*.

1.10.2. 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 9. 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"
```

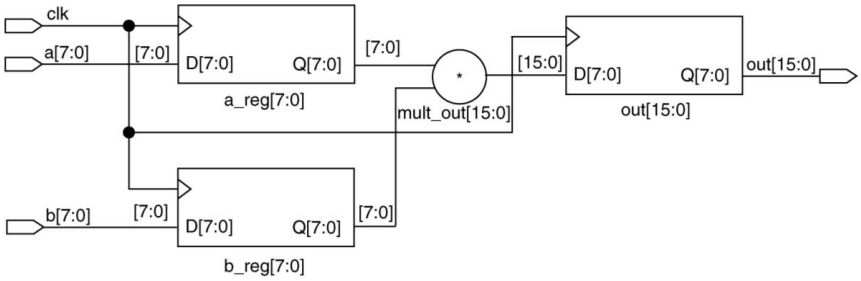
1.10.3. Inferring Intel FPGA IP Cores from HDL Code

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.

1.10.3.1. 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)



1.10.3.1.1. 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.

1.10.3.1.2. 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.

1.10.3.1.3. 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 3. 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 10. 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 11. 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;
```

1.10.3.2. 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 12. 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 13. 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;
```

1.10.3.3. 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 14. 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 15. 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";
```

1.10.3.4. 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.

1.10.3.5. 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.

1.11. Incremental Compilation and Block-Based Design

As designs become more complex and designers work in teams, a block-based incremental design flow is often an effective design approach. In an incremental compilation flow, you can make changes to part of the design while maintaining the placement and performance of unchanged parts of the design. Design iterations are made dramatically faster by focusing new compilations on particular design partitions and merging results with previous compilation results of other partitions. You can perform optimization on individual subblocks and then preserve the results before you integrate the blocks into a final design and optimize it at the top-level.

MultiPoint synthesis, which is available for certain device technologies in the Synplify Pro and Premier software, provides an automated block-based incremental synthesis flow. The MultiPoint feature manages a design hierarchy to let you design incrementally and synthesize designs that take too long for synthesis of the entire project. MultiPoint synthesis allows different netlist files to be created for different sections of a design hierarchy and supports the Intel Quartus Prime incremental compilation methodology. This feature also ensures that only those sections of a design that have been updated are resynthesized when the design is compiled, reducing synthesis run time and preserving the results for the unchanged blocks. You can change and resynthesize one section of a design without affecting other sections.

You can also partition your design and create different netlist files manually with the Synplify software by creating a separate project for the logic in each partition of the design. Creating different netlist files for each partition of the design also means that each partition can be independent of the others.

Hierarchical design methodologies can improve the efficiency of your design process, providing better design reuse opportunities and fewer integration problems when working in a team environment. When you use these incremental synthesis methodologies, you can take advantage of incremental compilation in the Intel Quartus Prime software. You can perform placement and routing on only the changed partitions of the design, which reduces place-and-route time and preserves your fitting results.

1.11.1. Design Flow for Incremental Compilation

The following steps describe the general incremental compilation flow when using these features of the Intel Quartus Prime software:

1. Create Verilog HDL or VHDL design files.
2. Determine which hierarchical blocks you want to treat as separate partitions in your design.
3. Set up your design using the MultiPoint synthesis feature or separate projects so that a separate netlist file is created for each design partition.
4. If using separate projects, disable I/O pad insertion in the implementations for lower-level partitions.
5. Compile and map each partition in the Synplify software, making constraints as you would in a non-incremental design flow.
6. Import the **.vqm** netlist and **.tcl** file for each partition into the Intel Quartus Prime software and set up the Intel Quartus Prime project(s) for incremental compilation.
7. Compile your design in the Intel Quartus Prime software and preserve the compilation results with the post-fit netlist in incremental compilation.
8. When you make design or synthesis optimization changes to part of your design, resynthesize only the partition you modified to generate a new netlist and **.tcl** file. Do not regenerate netlist files for the unmodified partitions.
9. Import the new netlist and **.tcl** file into the Intel Quartus Prime software and recompile the design in the Intel Quartus Prime software with incremental compilation.

1.11.2. Creating a Design with Separate Netlist Files for Incremental Compilation

The first stage of a hierarchical or incremental design flow is to ensure that different parts of your design do not affect each other. Ensure that you have separate netlists for each partition in your design so you can take advantage of incremental compilation in the Intel Quartus Prime software. If the entire design is in one netlist file, changes in one partition might affect other partitions because of possible node name changes when you resynthesize the design.

To ensure proper functionality of the synthesis flow, create separate netlist files only for modules and entities. In addition, each module or entity requires its own design file. If two different modules are in the same design file, but are defined as being part of different partitions, incremental compilation cannot be maintained since both partitions must be recompiled when one module is changed.

Intel recommends that you register all inputs and outputs of each partition. This makes logic synchronous, and avoids any delay penalty on signals that cross partition boundaries.

If you use boundary tri-states in a lower-level block, the Synplify software pushes, or bubbles, the tri-states through the hierarchy to the top-level to use the tri-state drivers on output pins of Intel devices. Because bubbling tri-states requires optimizing through hierarchies, lower-level tri-states are not supported with a block-based compilation methodology. Use tri-state drivers only at the external output pins of the device and in the top-level block in the hierarchy.

You can generate multiple **.vqm** netlist files with the MultiPoint synthesis flow in the Synplify Pro and Premier software, or by manually creating separate Synplify projects and creating a black box for each block that you want to designate as a separate design partition.

In the MultiPoint synthesis flow in the Synplify Pro and Premier software, you create multiple **.vqm** netlist files from one easy-to-manage, top-level synthesis project. By using the manual black box method, you have multiple synthesis projects, which might be required for certain team-based or bottom-up designs where a single top-level project is not desired.

After you have created multiple **.vqm** files using one of these two methods, you must create the appropriate Intel Quartus Prime projects to place-and-route the design.

1.11.3. Using MultiPoint Synthesis with Incremental Compilation

This topic describes how to generate multiple **.vqm** files using the Synplify Pro and Premier software MultiPoint synthesis flow. You must first set up your constraint file and Synplify options, then apply the appropriate Compile Point settings to write multiple **.vqm** files and create design partition assignments for incremental compilation.

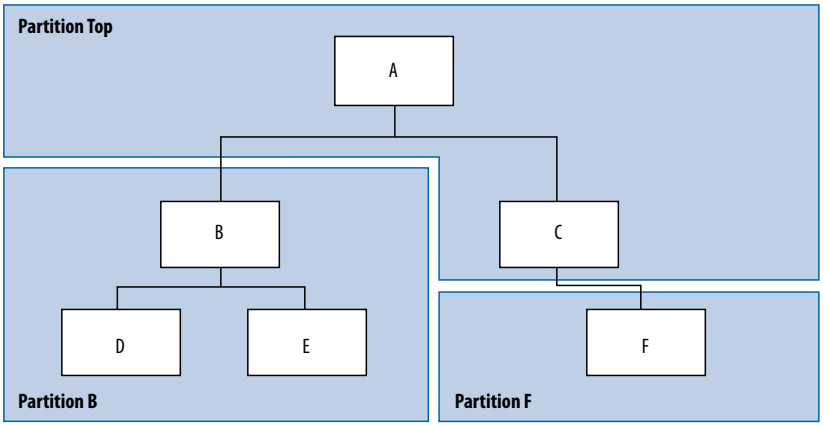
1.11.3.1. Set Compile Points and Create Constraint Files

The MultiPoint flow lets you segment a design into smaller synthesis units, called Compile Points. The synthesis software treats each Compile Point as a partition for incremental mapping, which allows you to isolate and work on each Compile Point module as independent segments of the larger design without impacting other design modules. A design can have any number of Compile Points, and Compile Points can be nested. The top-level module is always treated as a Compile Point.

Compile Points are optimized in isolation from their parent, which can be another Compile Point or a top-level design. Each block created with a Compile Point is unaffected by critical paths or constraints on its parent or other blocks. A Compile Point is independent, with its own individual constraints. During synthesis, any Compile Points that have not yet been synthesized are synthesized before the top level. Nested Compile Points are synthesized before the parent Compile Points in which they are contained. When you apply the appropriate setting for the Compile Point, a separate netlist is created for that Compile Point, isolating that logic from any other logic in the design.

The figure shows an example of a design hierarchy that is split into multiple partitions. The top-level block of each partition can be synthesized as a separate Compile Point.

Figure 3. Partitions in a Hierarchical Design



In this case, modules A, B, and F are Compile Points. The top-level Compile Point consists of the top-level block in the design (that is, block A in this example), including the logic that is not defined under another Compile Point. In this example, the design for top-level Compile Point A also includes the logic in one of its subblocks, C. Because block F is defined as its own Compile Point, it is not treated as part of the top-level Compile Point A. Another separate Compile Point B contains the logic in blocks B, D, and E. One netlist is created for the top-level module A and submodule C, another netlist is created for B and its submodules D and E, while a third netlist is created for F.

Apply Compile Points to the module, or to the architecture in the Synplify Pro SCOPE spreadsheet, or to the **.sdc** file. You cannot set a Compile Point in the Verilog HDL or VHDL source code. You can set the constraints manually using Tcl, by editing the **.sdc** file, or you can use the GUI.

1.11.3.1.1. Defining Compile Points With .tcl or .sdc Files

To set Compile Points with a **.tcl** or **.sdc** file, use the `define_compile_point` command.

Example 16. The define_compile_point Command

```
define_compile_point [-disable] {<objname>} -type {locked, partition}
```

<objname> represents any module in the design. The Compile Point type `{locked, partition}` indicates that the Compile Point represents a partition for the Intel Quartus Prime incremental compilation flow.

Each Compile Point has a set of constraint files that begin with the `define_current_design` command to set up the SCOPE environment, as follows:

```
define_current_design {<my_module>}
```

1.11.3.2. Additional Considerations for Compile Points

To ensure that changes to a Compile Point do not affect the top-level parent module, turn off the **Update Compile Point Timing Data** option in the **Implementation Options** dialog box. If this option is turned on, updates to a child module can impact the top-level module.

You can apply the `syn_allowed_resources` attribute to any Compile Point view to restrict the number of resources for a particular module.

When using Compile Points with incremental compilation, be aware of the following restrictions:

- To use Compile Points effectively, you must provide timing constraints (timing budgeting) for each Compile Point; the more accurate the constraints, the better your results are. Constraints are not automatically budgeted, so manual time budgeting is essential. Intel recommends that you register all inputs and outputs of each partition. This avoids any logic delay penalty on signals that cross-partition boundaries.
- When using the Synplify attribute `syn_useioff` to pack registers in the I/O Elements (IOEs) of Intel devices, these registers must be in the top-level module. Otherwise, you must direct the Intel Quartus Prime software to perform I/O register packing instead of the `syn_useioff` attribute. You can use the **Fast Input Register** or **Fast Output Register** options, or set I/O timing constraints and turn on **Optimize I/O cell register placement for timing** on the **Advanced Settings (Fitter)** dialog box in the Intel Quartus Prime software.
- There is no incremental synthesis support for top-level logic; any logic in the top-level is resynthesized during every compilation in the Synplify software.

For more information about using Compile Points and setting Synplify attributes and constraints for both top-level and lower-level Compile Points, refer to the *Synopsys FPGA Synthesis User Guide* and the *Synopsys FPGA Synthesis Reference Manual*.

1.11.3.3. Creating a Intel Quartus Prime Project for Compile Points and Multiple .vqm Files

During compilation, the Synplify Pro and Premier software creates a <*top-level project*>**.tcl** file that provides the Intel Quartus Prime software with the appropriate constraints and design partition assignments, creating a partition for each **.vqm** file along with the information to set up a Intel Quartus Prime project.

Depending on your design methodology, you can create one Intel Quartus Prime project for all netlists or a separate Intel Quartus Prime project for each netlist. In the standard incremental compilation design flow, you create design partition assignments and optional LogicLock™ floorplan location assignments for each partition in the design within a single Intel Quartus Prime project. This methodology allows for the best quality of results and performance preservation during incremental changes to your design.

You might require a bottom-up design flow if each partition must be optimized separately, such as for third-party IP delivery. If you use this flow, Intel recommends you create a design floorplan to avoid placement conflicts between each partition. To follow this design flow in the Intel Quartus Prime software, create separate Intel Quartus Prime projects, export each design partition and incorporate them into a top-level design using the incremental compilation features to maintain placement results.

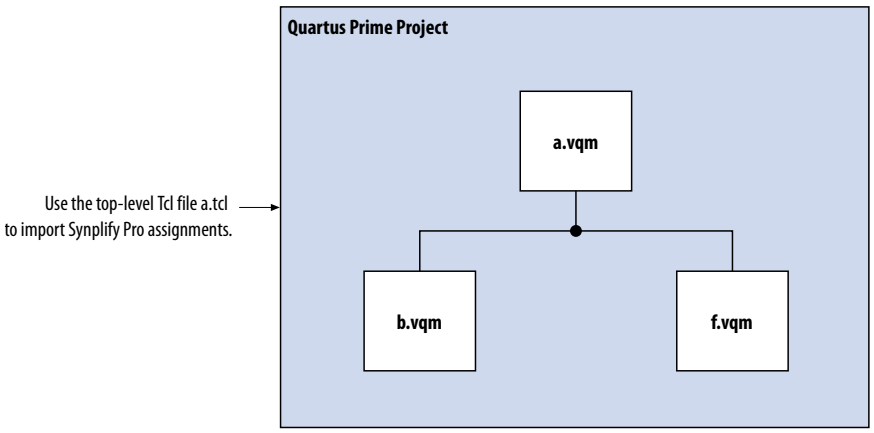
Related Information

Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script on page 10

1.11.3.3.1. Creating a Single Intel Quartus Prime Project for a Standard Incremental Compilation Flow

Use the *<top-level project>*.**tcl** file that contains the Synplify assignments for all partitions within the project. This method allows you to import all the partitions into one Intel Quartus Prime project and optimize all modules within the project at once, while taking advantage of the performance preservation and compilation-time reduction that incremental compilation offers.

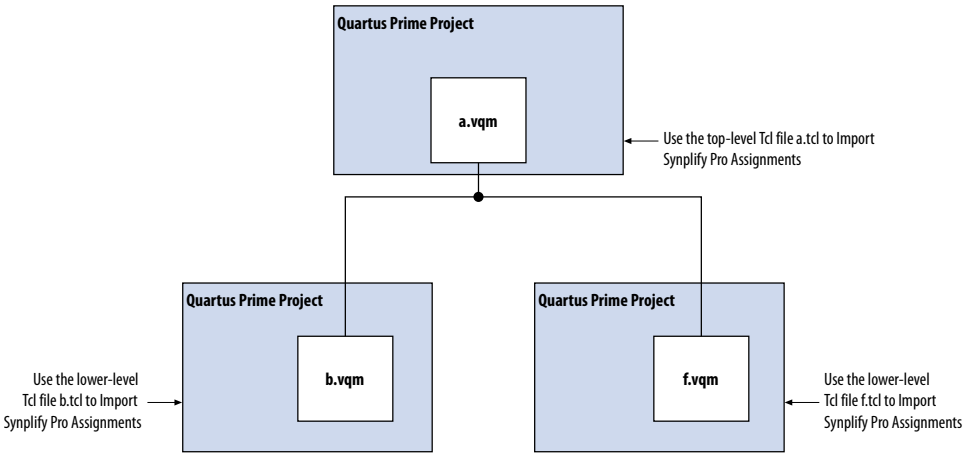
Figure 4. Design Flow Using Multiple .vqm Files with One Intel Quartus Prime Project



1.11.3.3.2. Creating Multiple Intel Quartus Prime Projects for a Bottom-Up Incremental Compilation Flow

Use the *<lower-level compile point>*.**tcl** files that contain the Synplify assignments for each Compile Point. Generate multiple Intel Quartus Prime projects, one for each partition and netlist in the design. The designers in the project can optimize their own partitions separately within the Intel Quartus Prime software and export the results for their own partitions. You can export the optimized subdesigns and then import them into one top-level Intel Quartus Prime project using incremental compilation to complete the design.

Figure 5. Design Flow Using Multiple .vqm Files with Multiple Intel Quartus Prime Projects



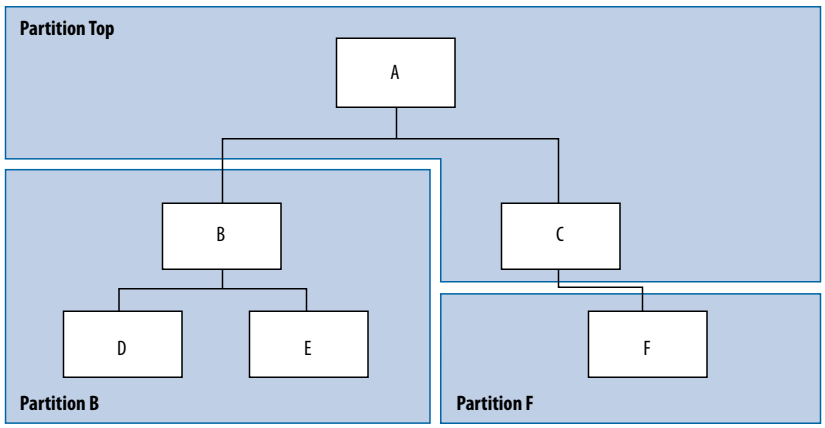
1.11.4. Creating Multiple .vqm Files for a Incremental Compilation Flow With Separate Synplify Projects

You can manually generate multiple **.vqm** files for a incremental compilation flow with black boxes and separate Synplify projects for each design partition. This manual flow is supported by versions of the Synplify software without the MultiPoint Synthesis feature.

1.11.4.1. Manually Creating Multiple .vqm Files With Black Boxes

To create multiple **.vqm** files manually in the Synplify software, create a separate project for each lower-level module and top-level design that you want to maintain as a separate **.vqm** file for an incremental compilation partition. Implement black box instantiations of lower-level partitions in your top-level project.

Figure 6. Partitions in a Hierarchical Design



The partition top contains the top-level block in the design (block A) and the logic that is not defined as part of another partition. In this example, the partition for top-level block A also includes the logic in one of its sub-blocks, block C. Because block F is contained in its own partition, it is not treated as part of the top-level partition A. Another separate partition, partition B, contains the logic in blocks B, D, and E. In a team-based design, engineers can work independently on the logic in different partitions. One netlist is created for the top-level module A and its submodule C, another netlist is created for module B and its submodules D and E, while a third netlist is created for module F.

1.11.4.1.1. Creating Multiple .vqm Files for this Design

To create multiple **.vqm** files for this design, follow these steps:

1. Generate a **.vqm** file for module B. Use **B.v/.vhd**, **D.v/.vhd**, and **E.v/.vhd** as the source files.
2. Generate a **.vqm** file for module F. Use **F.v/.vhd** as the source files.
3. Generate a top-level **.vqm** file for module A. Use **A.v/.vhd** and **C.v/.vhd** as the source files. Ensure that you use black box modules B and F, which were optimized separately in the previous steps.

1.11.4.1.2. Creating Black Boxes in Verilog HDL

Any design block that is not defined in the project, or included in the list of files to be read for a project, is treated as a black box by the software. Use the `syn_black_box` attribute to indicate that you intend to create a black box for the module. In Verilog HDL, you must provide an empty module declaration for a module that is treated as a black box.

The example shows the **A.v** top-level file. Follow the same procedure for lower-level files that also contain a black box for any module beneath the current level hierarchy.

Example 17. Verilog HDL Black Box for Top-Level File A.v

```
module A (data_in, clk, e, ld, data_out);
    input data_in, clk, e, ld;
    output [15:0] data_out;

    wire [15:0] cnt_out;

    B U1 (.data_in (data_in),.clk(clk), .ld (ld),.data_out(cnt_out));
    F U2 (.d(cnt_out), .clk(clk), .e(e), .q(data_out));

    // Any other code in A.v goes here.
endmodule

// Empty Module Declarations of Sub-Blocks B and F follow here.
// These module declarations (including ports) are required for black boxes.

module B (data_in, clk, ld, data_out) /* synthesis syn_black_box */ ;
    input data_in, clk, ld;
    output [15:0} data_out;
endmodule

module F (d, clk, e, q) /* synthesis syn_black_box */ ;
    input [15:0] d;
    input clk, e;
    output [15:0] q;
endmodule
```

1.11.4.1.3. Creating Black Boxes in VHDL

Any design that is not defined in the project, or included in the list of files to be read for a project, is treated as a black box by the software. Use the `syn_black_box` attribute to indicate that you intend to treat the component as a black box. In VHDL, you must have a component declaration for the black box.

Although VHDL is not case-sensitive, a **.vqm** (a subset of Verilog HDL) file is case-sensitive. Entity names and their port declarations are forwarded to the **.vqm** file. Black box names and port declarations are also passed to the **.vqm** file. To prevent case-based mismatches, use the same capitalization for black box and entity declarations in VHDL designs.

The example shows the **A.vhd** top-level file. Follow this same procedure for any lower-level files that contain a black box for any block beneath the current level of hierarchy.

Example 18. VHDL Black Box for Top-Level File A.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY synplify;
USE synplify.attributes.all;

ENTITY A IS
PORT (data_in : IN INTEGER RANGE 0 TO 15;
    clk, e, ld : IN STD_LOGIC;
    data_out : OUT INTEGER RANGE 0 TO 15 );
```

```
END A;

ARCHITECTURE a_arch OF A IS

COMPONENT B PORT(
    data_in : IN INTEGER RANGE 0 TO 15;
    clk, ld : IN STD_LOGIC;
    d_out : OUT INTEGER RANGE 0 TO 15);
END COMPONENT;

COMPONENT F PORT(
    d : IN INTEGER RANGE 0 TO 15;
    clk, e: IN STD_LOGIC;
    q : OUT INTEGER RANGE 0 TO 15);
END COMPONENT;

attribute syn_black_box of B: component is true;
atrribute syn_black_box of F: component is true;

-- Other component declarations in A.vhd go here
signal cnt_out : INTEGER RANGE 0 TO 15;

BEGIN

U1 : B
PORT MAP (
    data_in => data_in,
    clk => clk,
    ld => ld,
    d_out => cnt_out );

U2 : F
PORT MAP (
    d => cnt_out,
    clk => clk,
    e => e,
    q => data_out );

-- Any other code in A.vhd goes here

END a_arch;
```

After you complete the steps above, you have a netlist for each partition of the design. These files are ready for use with the incremental compilation flow in the Intel Quartus Prime software.

1.11.4.2. Creating a Intel Quartus Prime Project for Multiple .vqm Files

The Synplify software creates a **.tcl** file for each **.vqm** file that provides the Intel Quartus Prime software with the appropriate constraints and information to set up a project.

Depending on your design methodology, you can create one Intel Quartus Prime project for all netlists or a separate Intel Quartus Prime project for each netlist. In the standard incremental compilation design flow, you create design partition assignments and optional LogicLock floorplan location assignments for each partition in the design within a single Intel Quartus Prime project. This methodology allows for the best quality of results and performance preservation during incremental changes to your design. You might require a bottom-up design flow where each partition must be optimized separately, such as for third-party IP delivery.

To perform this design flow in the Intel Quartus Prime software, create separate Intel Quartus Prime projects, export each design partition and incorporate it into a top-level design using the incremental compilation features to maintain the results.

Related Information

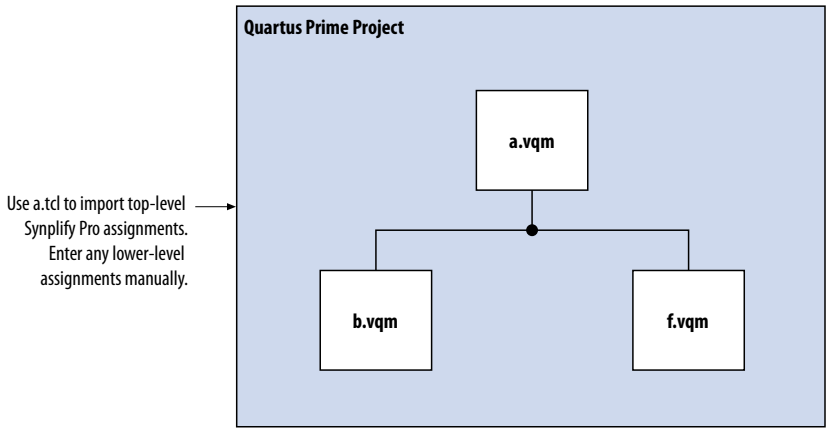
Running the Intel Quartus Prime Software Manually With the Synplify-Generated Tcl Script on page 10

1.11.4.2.1. Creating a Single Intel Quartus Prime Project for a Standard Incremental Compilation Flow

Use the *<top-level project>*.**tcl** file that contains the Synplify assignments for the top-level design. This method allows you to import all of the partitions into one Intel Quartus Prime project and optimize all modules within the project at once, taking advantage of the performance preservation and compilation time reduction offered by incremental compilation.

All of the constraints from the top-level project are passed to the Intel Quartus Prime software in the top-level **.tcl** file, but constraints made in the lower-level projects within the Synplify software are not forward-annotated. Enter these constraints manually in your Intel Quartus Prime project.

Figure 7. Design Flow Using Multiple .vqm Files with One Intel Quartus Prime Project

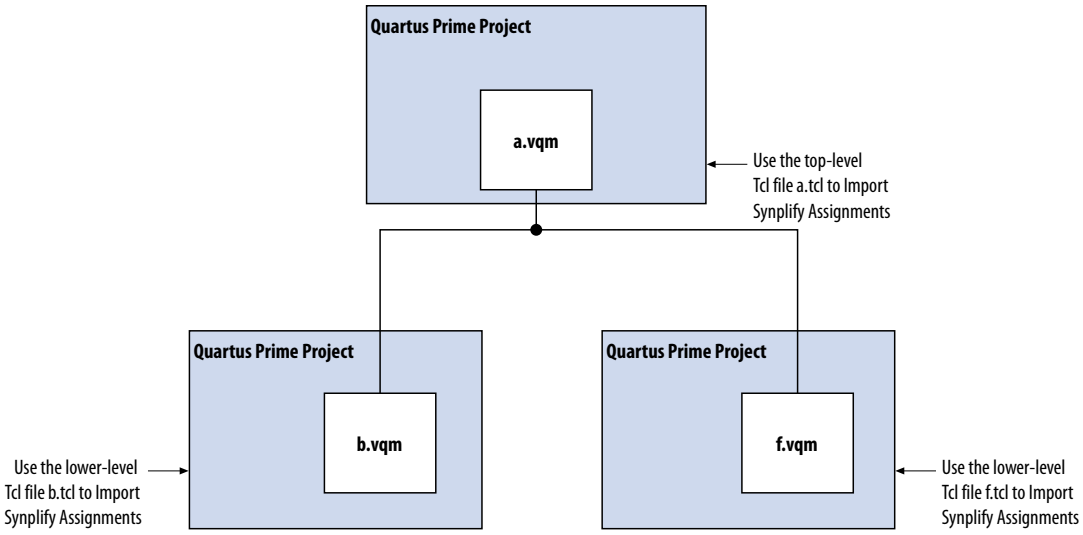


1.11.4.2.2. Creating Multiple Intel Quartus Prime Projects for a Bottom-Up Incremental Compilation Flow

Use the **.tcl** file that is created for each **.vqm** file by the Synplify software for each Synplify project. This method generates multiple Intel Quartus Prime projects, one for each block in the design. The designers in the project can optimize their own blocks separately within the Intel Quartus Prime software and export the placement of their own blocks.

Designers should create a LogicLock region to create a design floorplan for each block to avoid conflicts between partitions. The top-level designer then imports all the blocks and assignments into the top-level project. This method allows each block in the design to be optimized separately and then imported into one top-level project.

Figure 8. Design Flow Using Multiple Synplify Projects and Multiple Intel Quartus Prime Projects



1.11.5. Performing Incremental Compilation in the Intel Quartus Prime Software

In a standard design flow using Multipoint Synthesis, the Synplify software uses the Intel Quartus Prime top-level **.tcl** file to ensure that the two tools databases stay synchronized. The Tcl file creates, changes, or deletes partition assignments in the Intel Quartus Prime software for Compile Points that you create, change, or delete in the Synplify software. However, if you create, change, or delete a partition in the Intel Quartus Prime software, the Synplify software does not change your Compile Point settings. Make any corresponding change in your Synplify project to ensure that you create the correct **.vqm** files.

Note: If you use the NativeLink integration feature, the Synplify software does not use any information about design partition assignments that you have set in the Intel Quartus Prime software.

If you create netlist files with multiple Synplify projects, or if you do not use the Synplify Pro or Premier-generated **.tcl** files to update constraints in your Intel Quartus Prime project, you must ensure that your Synplify **.vqm** netlists align with your Intel Quartus Prime partition settings.

After you have set up your Intel Quartus Prime project with **.vqm** netlist files as separate design partitions, set the appropriate Intel Quartus Prime options to preserve your compilation results. On the Assignments menu, click **Design Partitions Window**. Change the **Netlist Type** to **Post-Fit** to preserve the previous compilation's post-fit placement results. If you do not make these settings, the Intel Quartus Prime software does not reuse the placement or routing results from the previous compilation.

You can take advantage of incremental compilation with your Synplify design to reduce compilation time in the Intel Quartus Prime software and preserve the results for unchanged design blocks.

Related Information

Using the Intel Quartus Prime Software to Run the Synplify Software on page 8

1.12. Synopsys Synplify* Support Revision History

Date	Version	Changes
2016.05.03	16.0.0	• Noted limitations of NativeLink synthesis.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
November 2013	13.1.0	Dita conversion. Restructured content.
June 2012	12.0.0	Removed survey link.
November 2011	10.1.1	Template update.
December 2010	10.1.0	• Changed to new document template. • Removed Classic Timing Analyzer support. • Removed the "altera_implement_in_esb or altera_implement_in_eab" section. • Edited the "Creating a Intel Quartus Prime Project for Compile Points and Multiple .vqm Files" on page 14–33 section for changes with the incremental compilation flow. • Edited the "Creating a Intel Quartus Prime Project for Multiple .vqm Files" on page 14–39 section for changes with the incremental compilation flow. • Editorial changes.
July 2010	10.0.0	• Minor updates for the Intel Quartus Prime software version 10.0 release.
November 2009	9.1.0	• Minor updates for the Intel Quartus Prime software version 9.1 release.
March 2009	9.0.0	• Added new section "Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration" on page 14–14. • Minor updates for the Intel Quartus Prime software version 9.0 release. • Chapter 10 was previously Chapter 9 in software version 8.1.
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size • Changed the chapter title from "Synplicity Synplify & Synplify Pro Support" to "Synopsys Synplify Support" • Replaced references to Synplicity with references to Synopsys • Added information about Synplify Premier • Updated supported device list • Added SystemVerilog information to Figure 14–1
May 2008	8.0.0	• Updated supported device list • Updated constraint annotation information for the Timing Analyzer • Updated RAM and MAC constraint limitations • Revised Table 9–1 • Added new section "Changing Synplify's Default Behavior for Instantiated Altera Megafunctions" • Added new section "Instantiating Intellectual Property Using the MegaWizard Plug-In Manager and IP Toolbench" • Added new section "Including Files for Intel Quartus Prime Placement and Routing Only" • Added new section "Additional Considerations for Compile Points" • Removed section "Apply the LogicLock Attributes" • Modified Figure 9–4, 9–43, 9–47. and 9–48 • Added new section "Performing Incremental Compilation in the Intel Quartus Prime Software" • Numerous text changes and additions throughout the chapter • Renamed several sections • Updated "Referenced Documents" section

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

2. Mentor Graphics Precision* Synthesis Support

2.1. About Precision RTL Synthesis Support

This manual delineates the support for the Mentor Graphics® Precision RTL Synthesis and Precision RTL Plus Synthesis software in the Intel Quartus Prime software, as well as key design flows, methodologies and techniques for improving your results for Intel devices. This manual assumes that you have set up, licensed, and installed the Precision Synthesis software and the Intel Quartus Prime software.

Note: You must set up, license, and install the Precision RTL Plus Synthesis software if you want to use the incremental synthesis feature for incremental compilation and block-based design.

To obtain and license the Precision Synthesis software, refer to the Mentor Graphics website. To install and run the Precision Synthesis software and to set up your work environment, refer to the *Precision Synthesis Installation Guide* in the Precision Manuals Bookcase. To access the Manuals Bookcase in the Precision Synthesis software, click **Help** and select **Open Manuals Bookcase**.

Related Information

Mentor Graphics website

2.2. Design Flow

The following steps describe a basic Intel Quartus Prime design flow using the Precision Synthesis software:

1. Create Verilog HDL or VHDL design files.
2. Create a project in the Precision Synthesis software that contains the HDL files for your design, select your target device, and set global constraints.
3. Compile the project in the Precision Synthesis 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 Synthesis software, you can identify and improve circuit area and performance issues using prelayout timing estimates.

   Note: For best results, Mentor Graphics 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 Synthesis software.
6. Create an Intel Quartus Prime project and import the following files generated by the Precision Synthesis software into the Intel Quartus Prime project:

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

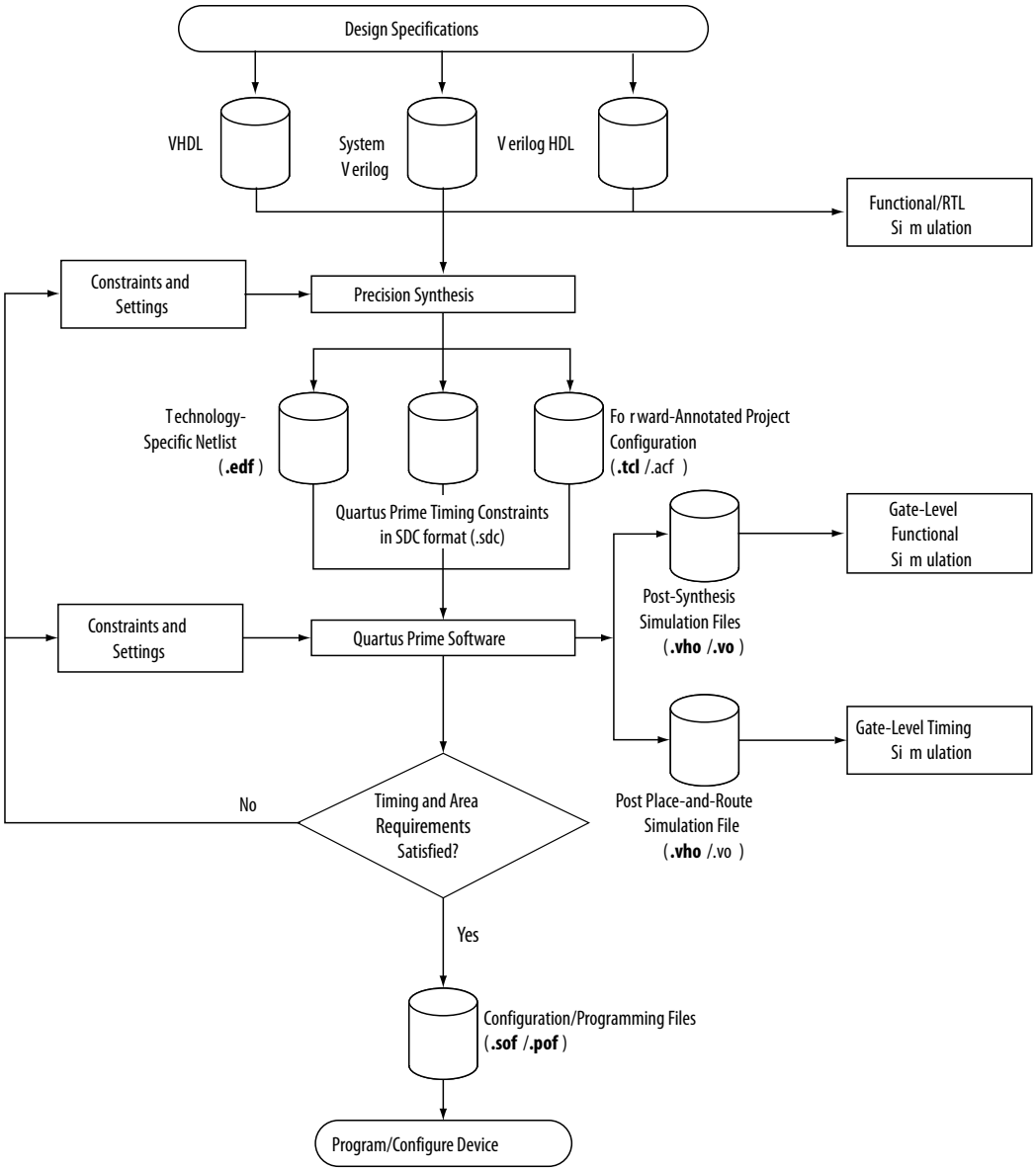
- The Verilog Quartus Mapping File (`.vqm`) netlist
- Synopsys Design Constraints File (`.sdc`) for Timing Analyzer constraints
- Tcl Script Files (`.tcl`) to set up your Intel Quartus Prime project and pass constraints

Note: If your design uses the Classic Timing Analyzer for timing analysis in the Intel Quartus Prime software versions 10.0 and earlier, the Precision Synthesis software generates timing constraints in the Tcl Constraints File (`.tcl`). If you are using the Intel Quartus Prime software versions 10.1 and later, you must use the Timing Analyzer for timing analysis.

7. After obtaining place-and-route results that meet your requirements, configure or program the Intel device.

You can run the Intel Quartus Prime software from within the Precision Synthesis software, or run the Precision Synthesis software using the Intel Quartus Prime software.

Figure 9. Design Flow Using the Precision Synthesis Software and Intel Quartus Prime Software



Related Information

- Running the Intel Quartus Prime Software from within the Precision Synthesis Software on page 49
- Using the Intel Quartus Prime Software to Run the Precision Synthesis Software on page 50

2.2.1. Timing Optimization

If your area or timing requirements are not met, you can change the constraints and resynthesize the design in the Precision Synthesis 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.

2.3. Intel Device Family Support

The Precision Synthesis software supports active devices available in the current version of the Intel Quartus Prime software. Support for newly released device families may require an overlay. Contact Mentor Graphics for more information.

2.4. Precision Synthesis Generated Files

During synthesis, the Precision Synthesis software produces several intermediate and output files.

Table 4. Precision Synthesis Software Intermediate and Output Files

File Extension	File Description
`.psp`	Precision Synthesis Project File.
`.xdb`	Mentor Graphics Design Database File.
`.rep`(3)	Synthesis Area and Timing Report File.
`.vqm`(4)	Technology-specific netlist in `.vqm` file format. By default, the Precision Synthesis software creates `.vqm` files for Arria series, Cyclone series, and Stratix series devices. The Precision Synthesis software defaults to creating `.vqm` files when the device is supported.

continued...

(3) 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.

(4) The Precision Synthesis software-generated VQM file is supported by the Intel Quartus Prime software version 10.1 and later.

File Extension	File Description
.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.
.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.
.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

- Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration on page 49
- Synthesizing the Design and Evaluating the Results on page 48

2.5. Creating and Compiling a Project in the Precision Synthesis Software

After creating your design files, create a project in the Precision Synthesis software that contains the basic settings for compiling the design.

2.6. Mapping the Precision Synthesis Design

In the next steps, you set constraints and map the design to technology-specific cells. The Precision Synthesis 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 Synthesis software performs static timing analysis to determine the location of the critical timing paths. The Precision Synthesis 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.

Mentor Graphics 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 Synthesis software to generate the file automatically the first time you synthesize your design. By default, the Precision Synthesis software saves all timing constraints and attributes in two files: precision_rtl.sdc and precision_tech.sdc. The 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 `.sdc` file contains all the constraints for the Precision Synthesis 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.

2.6.1. Setting Timing Constraints

The Precision Synthesis 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 Synthesis 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 in 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*.

2.6.2. 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.

2.6.3. Assigning Pin Numbers and I/O Settings

The Precision Synthesis 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, 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 Synthesis 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 Synthesis software constraint file.

Table 5. 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>`

continued...

Constraint	Entry Format for Precision Constraint File
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 19. Verilog HDL Pin Assignment

```
//pragma attribute clk pin_number P10;
```

Example 20. 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 Synthesis Reference Manual.*

2.6.4. Assigning I/O Registers

The Precision Synthesis 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.

For the Stratix series, Cyclone series, and the MAX II device families, the Precision Synthesis software can move an internal register to an I/O register without any restrictions on design hierarchy.

For more mature devices, the Precision Synthesis software can move an internal register to an I/O register only when the register exists in the top-level of the hierarchy. If the register is buried in the hierarchy, you must flatten the hierarchy so that the buried registers are moved to the top-level of the design.

2.6.5. Disabling I/O Pad Insertion

The Precision Synthesis 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 the software 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 Synthesis software with more information about the top-level pins in the design.

2.6.5.1. Preventing the Precision Synthesis 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 Synthesis software from adding I/O pads:

- You can use the Precision Synthesis GUI or add the following command to the project file:

```
setup_design -addio=false
```

2.6.5.2. Preventing the Precision Synthesis 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 Synthesis 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.

2.6.6. 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 Synthesis 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 Synthesis 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.

2.7. Synthesizing the Design and Evaluating the Results

During synthesis, the Precision Synthesis 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 Synthesis 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.

2.7.1. 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.

2.8. Exporting Designs to the Intel Quartus Prime Software Using NativeLink Integration

The NativeLink feature in the Intel Quartus Prime software facilitates the seamless transfer of information between the Intel Quartus Prime software and EDA tools, which allows you to run other EDA design entry/synthesis, simulation, and timing analysis tools automatically from within the Intel Quartus Prime software.

After a design is synthesized in the Precision Synthesis software, the technology-mapped design is written to the current implementation directory as an EDIF netlist file, along with a Intel Quartus Prime Project Configuration File and a place-and-route constraints file. You can use the Project Configuration script, *<project name>*.**tcl**, to create and compile a Intel Quartus Prime project for your EDIF or VQM netlist. This script makes basic project assignments, such as assigning the target device specified in the Precision Synthesis software. If you select a newer Intel device, the constraints are written in SDC format to the *<project name>*_**pnr_constraints.sdc** file by default, which is used by the Fitter and the Timing Analyzer in the Intel Quartus Prime software.

Use the following Precision Synthesis software command before compilation to generate the *<project name>*_**pnr_constraints.sdc**:

```
setup_design -timequest_sdc
```

With this command, the file is generated after synthesis.

2.8.1. Running the Intel Quartus Prime Software from within the Precision Synthesis Software

The Precision Synthesis software also has a built-in place-and-route environment that allows you to run the Intel Quartus Prime Fitter and view the results in the Precision Synthesis GUI. This feature is useful when performing an initial compilation of your design to view post-place-and-route timing and device utilization results. Not all the advanced Intel Quartus Prime options that control the compilation process are available when you use this feature.

Two primary Precision Synthesis software commands control the place-and-route process. Use the `setup_place_and_route` command to set the place-and-route options. Start the process with the `place_and_route` command.

Precision Synthesis software uses individual Intel Quartus Prime executables, such as analysis and synthesis, Fitter, and the Timing Analyzer for improved runtime and memory utilization during place and route. This flow is referred to as the **Intel Quartus Prime Modular** flow option in the Precision Synthesis software. By default, the Precision Synthesis software generates a Intel Quartus Prime Project Configuration File (.**tcl** file) for current device families. Timing constraints that you set during synthesis are exported to the Intel Quartus Prime place-and-route constraints file *<project name>*_**pnr_constraints**.**sdc**.

After you compile the design in the Intel Quartus Prime software from within the Precision Synthesis software, you can invoke the Intel Quartus Prime GUI manually and then open the project using the generated Intel Quartus Prime project file. You can view reports, run analysis tools, specify options, and run the various processing flows available in the Intel Quartus Prime software.

For more information about running the Intel Quartus Prime software from within the Precision Synthesis software, refer to the *Intel Quartus Prime Integration* chapter in the *Precision Synthesis Reference Manual*.

2.8.2. Running the Intel Quartus Prime Software Manually Using the Precision Synthesis-Generated Tcl Script

You can run the Intel Quartus Prime software using a Tcl script generated by the Precision Synthesis software. To run the Tcl script generated by the Precision Synthesis software to set up your project and start a full compilation, perform the following steps:

1. Ensure the .**vqm** file, .**tcl** files, and .**sdc** file are located in the same directory. The files should be located in the implementation directory by default.
2. In the Intel Quartus Prime software, on the View menu, point to **Utility Windows** and click **Tcl Console**.
3. At the Tcl Console command prompt, type the command:

```
source <path>/<project name>.tcl
```

4. On the File menu, click **Open Project**. Browse to the project name and click **Open**.
5. Compile the project in the Intel Quartus Prime software.

2.8.3. Using the Intel Quartus Prime Software to Run the Precision Synthesis Software

With NativeLink integration, you can set up the Intel Quartus Prime software to run the Precision Synthesis software. This feature allows you to use the Precision Synthesis software to synthesize a design as part of a standard compilation. When you use this feature, the Precision Synthesis software does not use any timing constraints or assignments that you have set in the Intel Quartus Prime software.

2.8.4. Passing Constraints to the Intel Quartus Prime Software

The place-and-route constraints script forward-annotates timing constraints that you made in the Precision Synthesis software. This integration allows you to enter these constraints once in the Precision Synthesis software, and then pass them automatically to the Intel Quartus Prime software.

The following constraints are translated by the Precision Synthesis software and are applicable to the Timing Analyzer:

- `create_clock`
- `set_input_delay`
- `set_output_delay`
- `set_max_delay`
- `set_min_delay`
- `set_false_path`
- `set_multicycle_path`

2.8.4.1. create_clock

You can specify a clock in the Precision Synthesis software.

Example 21. Specifying a Clock Using create_clock

```
create_clock -name <clock_name> -period <period in ns> \
-waveform {<edge_list>} -domain <ClockDomain> <pin>
```

The period is specified in units of nanoseconds (ns). If no clock domain is specified, the clock belongs to a default clock domain `main`. All clocks in the same clock domain are treated as synchronous (related) clocks. If no *<clock_name>* is provided, the default name `virtual_default` is used. The *<edge_list>* sets the rise and fall edges of the clock signal over an entire clock period. The first value in the list is a rising transition, typically the first rising transition after time zero. The waveform can contain any even number of alternating edges, and the edges listed should alternate between rising and falling. The position of any edge can be equal to or greater than zero but must be equal to or less than the clock period.

If `-waveform` *<edge_list>* is not specified and `-period` *<period in ns>* is specified, the default waveform has a rising edge of 0.0 and a falling edge of *<period_value>*/2.

The Precision Synthesis software maps the clock constraint to the Timing Analyzer `create_clock` setting in the Intel Quartus Prime software.

The Intel Quartus Prime software supports only clock waveforms with two edges in a clock cycle. If the Precision Synthesis software finds a multi-edge clock, it issues an error message when you synthesize your design in the Precision Synthesis software.

2.8.4.2. set_input_delay

This port-specific input delay constraint is specified in the Precision Synthesis software.

Example 22. Specifying set_input_delay

```
set_input_delay {<delay_value> <port_pin_list>} \
-clock <clock_name> -rise -fall -add_delay
```

This constraint is mapped to the `set_input_delay` setting in the Intel Quartus Prime software.

When the reference clock *<clock_name>* is not specified, all clocks are assumed to be the reference clocks for this assignment. The input pin name for the assignment can be an input pin name of a time group. The software can use the `clock_fall` option to specify delay relative to the falling edge of the clock.

Note: Although the Precision Synthesis software allows you to set input delays on pins inside the design, these constraints are not sent to the Intel Quartus Prime software, and a message is displayed.

2.8.4.3. set_output_delay

This port-specific output delay constraint is specified in the Precision Synthesis software.

Example 23. Using the set_output_delay Constraint

```
set_output_delay {<delay_value> <port_pin_list>} \
-clock <clock_name> -rise -fall -add_delay
```

This constraint is mapped to the `set_output_delay` setting in the Intel Quartus Prime software.

When the reference clock *<clock_name>* is not specified, all clocks are assumed to be the reference clocks for this assignment. The output pin name for the assignment can be an output pin name of a time group.

Note: Although the Precision Synthesis software allows you to set output delays on pins inside the design, these constraints are not sent to the Intel Quartus Prime software.

2.8.4.4. set_max_delay and set_min_delay

The maximum delay and minimum delay for a point-to-point timing path constraint is specified in the Precision Synthesis software.

Example 24. Using the set_max_delay Constraint

```
set_max_delay -from {<from_node_list>} -to {<to_node_list>} <delay_value>
```

Example 25. Using the set_min_delay Constraint

```
set_min_delay -from {<from_node_list>} -to {<to_node_list>} <delay_value>
```

The `set_max_delay` and `set_min_delay` commands specify that the maximum and minimum respectively, required delay for any start point in *<from_node_list>* to any endpoint in *<to_node_list>* must be less than or greater than *<delay_value>*. Typically, you use these commands to override the default setup constraint for any path with a specific maximum or minimum time value for the path.

The node lists can contain a collection of clocks, registers, ports, pins, or cells. The `-from` and `-to` parameters specify the source (start point) and the destination (endpoint) of the timing path, respectively. The source list (<*from_node_list*>) cannot include output ports, and the destination list (<*to_node_list*>) cannot include input ports. If you include more than one node on a list, you must enclose the nodes in quotes or in braces ({ }).

If you specify a clock in the source list, you must specify a clock in the destination list. Applying `set_max_delay` or `set_min_delay` setting between clocks applies the exception from all registers or ports driven by the source clock to all registers or ports driven by the destination clock. Applying exceptions between clocks is more efficient than applying them for specific node-to-node, or node-to-clock paths. If you want to specify pin names in the list, the source must be a clock pin and the destination 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 for those driven by the clock pin.

2.8.4.5. set_false_path

The false path constraint is specified in the Precision Synthesis software.

Example 26. Using the set_false_path Constraint

```
set_false_path -to <to_node_list> -from <from_node_list> -reset_path
```

The node lists can be a list of clocks, ports, instances, and pins. Multiple elements in the list can be represented using wildcards such as * and ?.

In a place-and-route Tcl constraints file, this false path setting in the Precision Synthesis software is mapped to a `set_false_path` setting. The Intel Quartus Prime software supports `setup`, `hold`, `rise`, or `fall` options for this assignment.

The node lists for this assignment represents top-level ports and/or nets connected to instances (end points of timing assignments).

Any false path setting in the Precision Synthesis software can be mapped to a setting in the Intel Quartus Prime software with a `through` path specification.

2.8.4.6. set_multicycle_path

The multicycle path constraint is specified in the Precision Synthesis software.

Example 27. Using the set_multicycle_path Constraint

```
set_multicycle_path <multiplier_value> [-start] [-end] \
-to <to_node_list> -from <from_node_list> -reset_path
```

The node list can contain clocks, ports, instances, and pins. Multiple elements in the list can be represented using wildcards such as * and ?. Paths without multicycle path definitions are identical to paths with multipliers of 1. To add one additional cycle to the datapath, use a multiplier value of 2. The option `start` indicates that source clock cycles should be considered for the multiplier. The option `end` indicates that destination clock cycles should be considered for the multiplier. The default is to reference the end clock.

In the place-and-route Tcl constraints file, the multicycle path setting in the Precision Synthesis software is mapped to a `set_multicycle_path` setting. The Intel Quartus Prime software supports the `rise` or `fall` options on this assignment.

The node lists represent top-level ports and/or nets connected to instances (end points of timing assignments). The node lists can contain wildcards (such as *); the Intel Quartus Prime software automatically expands all wildcards.

Any multicycle path setting in Precision Synthesis software can be mapped to a setting in the Intel Quartus Prime software with a `-through` specification.

2.9. 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 Synthesis 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 57

2.9.1. 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 Synthesis 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 Synthesis project to describe the port connections of the black box. Adding the IP core wrapper file <*output file*>.v in your Precision Synthesis project is optional, but you must add it to your Intel Quartus Prime project along with the Precision Synthesis generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file <*output file*>.v in your Precision Synthesis project and turn on the **Exclude file from Compile Phase** option in the Precision Synthesis 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.

2.9.2. 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 Synthesis 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 Synthesis project is optional, but you must add the file to your Intel Quartus Prime project along with the Precision Synthesis-generated EDIF or VQM netlist.

Alternatively, you can include the IP core wrapper file <*output file*>.v in your Precision Synthesis project and turn on the **Exclude file from Compile Phase** option in the Precision Synthesis 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.

2.9.3. 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 Synthesis software can use to synthesize and optimize logic around the IP efficiently. As a result, the Precision Synthesis 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 Synthesis 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.

Note: For information about creating a gray box netlist file from the command line, search Altera's Knowledge Database.

2.9.4. 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 Synthesis 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 28. 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
```

2.9.5. 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 29. 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;
```

2.9.6. Inferring Intel FPGA IP Cores from HDL Code

The Precision Synthesis 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 Synthesis 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 Synthesis Style Guide*.

2.9.6.1. Multipliers

The Precision Synthesis 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 Synthesis software also allows you to control the device resources that are used to implement individual multipliers.

2.9.6.1.1. Controlling DSP Block Inference for Multipliers

By default, the Precision Synthesis software uses DSP blocks available in Stratix series devices to implement multipliers. The default setting is **AUTO**, which allows the Precision Synthesis software to map to logic look-up tables (LUTs) or DSP blocks, depending on the size of the multiplier. You can use the Precision Synthesis GUI or HDL attributes for direct mapping to only logic elements or to only DSP blocks.

Table 6. Options for dedicated_mult Parameter to Control Multiplier Implementation in Precision Synthesis

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.

2.9.6.2. Setting the Use Dedicated Multiplier Option

To set the `Use Dedicated Multiplier` option in the Precision Synthesis GUI, compile the design, and then in the Design Hierarchy browser, right-click the operator for the desired multiplier and click **Use Dedicated Multiplier**.

2.9.6.3. 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 30. Setting the dedicated_mult Attribute in Verilog HDL

```
//synthesis attribute <signal name> dedicated_mult <value>
```

Example 31. 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 Synthesis 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 32. Setting the preserve_signal Attribute in Verilog HDL

```
//synthesis attribute <signal name> preserve_signal TRUE
```

Example 33. Setting the preserve_signal Attribute in VHDL

```
ATTRIBUTE preserve_signal: BOOLEAN;
ATTRIBUTE preserve_signal OF <signal name>: SIGNAL IS TRUE;
```

Example 34. 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 35. 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;
```

2.9.6.4. Multiplier-Accumulators and Multiplier-Adders

The Precision Synthesis 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 Synthesis 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 Synthesis 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 device family handbook and device-specific documentation. For details about which functions a given DSP block can implement, refer to the DSP Solutions Center on the Altera website.

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 Mentor Graphics *Precision Synthesis Style Guide*.

Related Information

Altera DSP Solutions website

2.9.6.5. Controlling DSP Block Inference

By default, the Precision Synthesis 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 7. 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 36. Setting the extract_mac Attribute in Verilog HDL

```
//synthesis attribute <module name> extract_mac <value>
```

Example 37. 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 38. 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 39. 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;
```

2.9.6.6. RAM and ROM

The Precision Synthesis 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 software then places these functions in memory blocks.

The software supports inference for these functions only if the target device family has dedicated memory blocks.

For more information about inferring RAM and ROM IP cores in HDL code, refer to the *Precision Synthesis Style Guide*.

2.10. Incremental Compilation and Block-Based Design

As designs become more complex and designers work in teams, a block-based incremental design flow is often an effective design approach. In an incremental compilation flow, you can make changes to one part of the design while maintaining the placement and performance of unchanged parts of the design. Design iterations can be made dramatically faster by focusing new compilations on particular design partitions and merging results with the results of previous compilations of other partitions. You can perform optimization on individual blocks and then integrate them into a final design and optimize the design at the top-level.

The first step in an incremental design flow is to make sure that different parts of your design do not affect each other. You must ensure that you have separate netlists for each partition in your design. If the whole design is in one netlist file, changes in one partition affect other partitions because of possible node name changes when you resynthesize the design.

You can create different implementations for each partition in your Precision Synthesis project, which allows you to switch between partitions without leaving the current project file. You can also create a separate project for each partition if you require separate projects for a team-based design flow. Alternatively, you can use the incremental synthesis capability in the Precision RTL Plus software.

2.10.1. Creating a Design with Precision RTL Plus Incremental Synthesis

The Precision RTL Plus incremental synthesis flow for Intel Quartus Prime incremental compilation uses a partition-based approach to achieve faster design cycle time.

Using the incremental synthesis feature, you can create different netlist files for different partitions of a design hierarchy within one partition implementation, which makes each partition independent of the others in an incremental compilation flow. Only the portions of a design that have been updated must be recompiled during design iterations. You can make changes and resynthesize one partition in a design to create a new netlist without affecting the synthesis results or fitting of other partitions.

The following steps show a general flow for partition-based incremental synthesis with Intel Quartus Prime incremental compilation:

1. Create Verilog HDL or VHDL design files.
2. Determine which hierarchical blocks you want to treat as separate partitions in your design, and designate the partitions with the `incr_partition` attribute.
3. Create a project in the Precision RTL Plus Synthesis software and add the HDL design files to the project.
4. Enable incremental synthesis in the Precision RTL Plus Synthesis software using one of these methods:
   - Use the Precision RTL Plus Synthesis GUI to turn on **Enable Incremental Synthesis**.
   - Run the following command in the Transcript Window:

     ```
     setup_design -enable_incr_synth
     ```
5. Run the basic Precision Synthesis flow of compilation, synthesis, and place-and-route on your design. In subsequent runs, the Precision RTL Plus Synthesis software processes only the parts of the design that have changed, resulting in a shorter iteration than the initial run. The performance of the unchanged partitions is preserved.

   The Precision RTL Plus Synthesis software sets the netlist types of the unchanged partitions to **Post Fit** and the changed partitions to **Post Synthesis**. You can change the netlist type during timing closure in the Intel Quartus Prime software to obtain the best QoR.
6. Import the EDIF or VQM netlist for each partition and the top-level .**tcl** file into the Intel Quartus Prime software, and set up the Intel Quartus Prime project to use incremental compilation.
7. Compile your Intel Quartus Prime project.
8. If you want, you can change the Intel Quartus Prime incremental compilation netlist type for a partition with the **Design Partitions Window**. You can change the **Netlist Type** to one of the following options:
   - To preserve the previous post-fit placement results, change the **Netlist Type** of the partition to **Post-Fit**.
   - To preserve the previous routing results, set the **Fitter Preservation Level** of the partition to **Placement and Routing**.

2.10.1.1. Creating Partitions with the incr_partition Attribute

Partitions are set using the HDL `incr_partition` attribute. The Precision Synthesis software creates or deletes partitions by reading this attribute during compilation iterations. The attribute can be attached to either the design unit definition or an instance.

To delete partitions, you can remove the attribute or set the attribute value to false.

Note: The Precision Synthesis software ignores partitions set in a black box.

Example 40. Using incr_partition Attribute to Create a Partition in Verilog HDL

```
Design unit partition:

module my_block(
   input clk;
   output reg [31:0] data_out) /* synthesis incr_partition */ ;

Instance partition:

my_block my_block_inst(.clk(clk), .data_out(data_out));
// synthesis attribute my_block_inst incr_partition true
```

Example 41. Using incr_partition Attribute to a Create Partition in VHDL

```
Design unit partition:

entity my_block is
   port(
      clk : in std_logic;
      data_out : out std_logic_vector(31 downto 0)
   );
   attribute incr_partition : boolean;
   attribute incr_partition of my_block : entity is true;
end entity my_block;

Instance partition:

component my_block is
   port(
      clk : in std_logic;
      data_out : out std_logic_vector(31 downto 0)
   );
end component;

attribute incr_partition : boolean;
attribute incr_partition of my_block_inst : label is true;

my_block_inst my_block
   port map(clk, data_out);
```

2.10.2. Creating Multiple Mapped Netlist Files With Separate Precision Projects or Implementations

You can manually generate multiple netlist files, which can be VQM or EDIF files, for incremental compilation using black boxes and separate Precision projects or implementations for each design partition. This manual flow is supported in versions of the Precision software that do not include the incremental synthesis feature. You might also use this feature if you perform synthesis in a team-based environment without a top-level synthesis project that includes all of the lower-level design blocks.

In the Precision Synthesis software, create a separate implementation, or a separate project, for each lower-level module and for the top-level design that you want to maintain as a separate netlist file. Implement black box instantiations of lower-level modules in your top-level implementation or project.

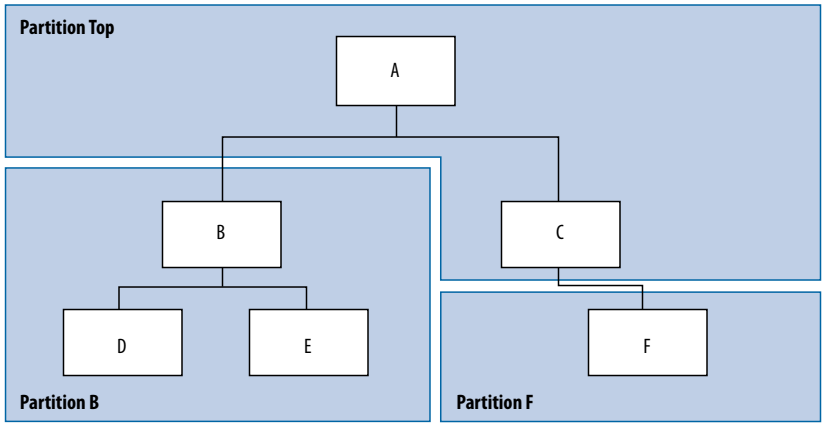
For more information about managing implementations and projects, refer to the *Precision RTL Synthesis User's Manual*.

Note: In a standard Intel Quartus Prime incremental compilation flow, Precision Synthesis software constraints made on lower-level modules are not passed to the Intel Quartus Prime software. Ensure that appropriate constraints are made in the top-level Precision Synthesis project, or in the Intel Quartus Prime project.

2.10.3. Creating Black Boxes to Create Netlists

In the figure below, the top-level partition contains the top-level block in the design (block A) and the logic that is not defined as part of another partition. In this example, the partition for top-level block A also includes the logic in the sub-block C. Because block F is contained in its own partition, it is not treated as part of the top-level partition A. Another separate partition, B, contains the logic in blocks B, D, and E. In a team-based design, different engineers may work on the logic in different partitions. One netlist is created for the top-level module A and its submodule C, another netlist is created for module B and its submodules D and E, while a third netlist is created for module F.

Figure 10. Partitions in a Hierarchical Design



To create multiple EDIF netlist files for this design, follow these steps:

1. Generate a netlist file for module B. Use **B.v**/.**vhd**, **D.v**/.**vhd**, and **E.v**/.**vhd** as the source files.
2. Generate a netlist file for module F. Use **F.v**/.**vhd** as the source file.
3. Generate a top-level netlist file for module A. Use **A.v**/.**vhd** and **C.v**/.**vhd** as the source files. Ensure that you create black boxes for modules B and F, which were optimized separately in the previous steps.

The goal is to individually synthesize and generate a netlist file for each lower-level module and then instantiate these modules as black boxes in the top-level file. You can then synthesize the top-level file to generate the netlist file for the top-level design. Finally, both the lower-level and top-level netlist files are provided to your Intel Quartus Prime project.

Note: When you make design or synthesis optimization changes to part of your design, resynthesize only the changed partition to generate the new netlist file. Do not resynthesize the implementations or projects for the unchanged partitions.

2.10.3.1. Creating Black Boxes in Verilog HDL

Any design block that is not defined in the project or included in the list of files to be read for a project is treated as a black box by the software. In Verilog HDL, you must provide an empty module declaration for any module that is treated as a black box.

A black box for the top-level file **A.v** is shown in the following example. Provide an empty module declaration for any lower-level files, which also contain a black box for any module beneath the current level of hierarchy.

Example 42. Verilog HDL Black Box for Top-Level File A.v

```
module A (data_in, clk, e, ld, data_out);
   input data_in, clk, e, ld;
   output [15:0] data_out;
   wire [15:0] cnt_out;
   B U1 (.data_in (data_in),.clk(clk), .ld (ld),.data_out(cnt_out));
   F U2 (.d(cnt_out), .clk(clk), .e(e), .q(data_out));
   // Any other code in A.v goes here.
endmodule
//Empty Module Declarations of Sub-Blocks B and F follow here.
// These module declarations (including ports) are required for black boxes.
module B (data_in, clk, ld, data_out);
   input data_in, clk, ld;
   output [15:0] data_out;
endmodule
module F (d, clk, e, q);
   input [15:0] d;
   input clk, e;
   output [15:0] q;
endmodule
```

2.10.3.2. Creating Black Boxes in VHDL

Any design block that is not defined in the project or included in the list of files to be read for a project is treated as a black box by the software. In VHDL, you must provide a component declaration for the black box.

A black box for the top-level file **A.vhd** is shown in the example below. Provide a component declaration for any lower-level files that also contain a black box or for any block beneath the current level of hierarchy.

Example 43. VHDL Black Box for Top-Level File A.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY A IS
   PORT ( data_in : IN INTEGER RANGE 0 TO 15;
      clk, e, ld : IN STD_LOGIC;
      data_out : OUT INTEGER RANGE 0 TO 15);
END A;
ARCHITECTURE a_arch OF A IS
COMPONENT B PORT(
   data_in : IN INTEGER RANGE 0 TO 15;
   clk, ld : IN STD_LOGIC;
   d_out : OUT INTEGER RANGE 0 TO 15);
END COMPONENT;
COMPONENT F PORT(
   d : IN INTEGER RANGE 0 TO 15;
   clk, e: IN STD_LOGIC;
   q : OUT INTEGER RANGE 0 TO 15);
END COMPONENT;
-- Other component declarations in A.vhd go here
signal cnt_out : INTEGER RANGE 0 TO 15;
BEGIN
   U1 : B
   PORT MAP (
      data_in => data_in,
      clk => clk,
      ld => ld,
      d_out => cnt_out);
   U2 : F
```

```
   PORT MAP (
      d => cnt_out,
      clk => clk,
      e => e,
      q => data_out);
   -- Any other code in A.vhd goes here
END a_arch;
```

After you complete the steps outlined above, you have different netlist files for each partition of the design. These files are ready for use with incremental compilation in the Intel Quartus Prime software.

2.10.4. Creating Intel Quartus Prime Projects for Multiple Netlist Files

The Precision Synthesis software creates a .**tcl** file for each implementation, and provides the Intel Quartus Prime software with the appropriate constraints and information to set up a project. When using incremental synthesis, the Precision RTL Plus Synthesis software creates only a single .**tcl** file, *<project name>*_**incr_partitions.tcl**, to pass the partition information to the Intel Quartus Prime software.

Depending on your design methodology, you can create one Intel Quartus Prime project for all netlists, or a separate Intel Quartus Prime project for each netlist. In the standard incremental compilation design flow, you create design partition assignments for each partition in the design within a single Intel Quartus Prime project. This methodology provides the best QoR and performance preservation during incremental changes to your design. You might require a bottom-up design flow if each partition must be optimized separately, such as for third-party IP delivery.

To follow this design flow in the Intel Quartus Prime software, create separate Intel Quartus Prime projects and export each design partition and incorporate it into a top-level design using the incremental compilation features to maintain placement results.

Related Information

Running the Intel Quartus Prime Software Manually Using the Precision Synthesis-Generated Tcl Script on page 50

2.10.4.1. Creating a Single Intel Quartus Prime Project for a Standard Incremental Compilation Flow

Use the *<top-level project>*.**tcl** file generated for the top-level partition to create your Intel Quartus Prime project and import all the netlists into this one Intel Quartus Prime project for an incremental compilation flow. You can optimize all partitions within the single Intel Quartus Prime project and take advantage of the performance preservation and compilation time reduction that incremental compilation provides.

All the constraints from the top-level implementation are passed to the Intel Quartus Prime software in the top-level .**tcl** file, but any constraints made only in the lower-level implementations within the Precision Synthesis software are not forward-annotated. Enter these constraints manually in your Intel Quartus Prime project.

2.10.4.2. Creating Multiple Intel Quartus Prime Projects for a Bottom-Up Flow

Use the **.tcl** files generated by the Precision Synthesis software for each Precision Synthesis software implementation or project to generate multiple Intel Quartus Prime projects, one for each partition in the design. Each designer in the project can optimize their block separately in the Intel Quartus Prime software and export the placement of their blocks using incremental compilation. Designers should create a LogicLock region to provide a floorplan location assignment for each block; the top-level designer should then import all the blocks and assignments into the top-level project.

2.10.5. Hierarchy and Design Considerations

To ensure the proper functioning of the synthesis flow, you can create separate partitions only for modules, entities, or existing netlist files. In addition, each module or entity must have its own design file. If two different modules are in the same design file, but are defined as being part of different partitions, incremental synthesis cannot be maintained because both regions must be recompiled when you change one of the modules.

Intel recommends that you register all inputs and outputs of each partition. This makes logic synchronous and avoids any delay penalty on signals that cross partition boundaries.

If you use boundary tri-states in a lower-level block, the Precision Synthesis software pushes the tri-states through the hierarchy to the top-level to make use of the tri-state drivers on output pins of Intel devices. Because pushing tri-states requires optimizing through hierarchies, lower-level tri-states are not supported with a block-based compilation methodology. You should use tri-state drivers only at the external output pins of the device and in the top-level block in the hierarchy.

2.11. Mentor Graphics Precision* Synthesis Support Revision History

Date	Version	Changes
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	• Dita conversion. • Removed obsolete devices. • Replaced Intel FPGA IP, MegaWizard, and IP Toolbench content with IP Catalog and Parameter Editor content.
June 2012	12.0.0	• Removed survey link.
November 2011	10.1.1	• Template update. • Minor editorial changes.
December 2010	10.1.0	• Changed to new document template. • Removed Classic Timing Analyzer support. • Added support for **. vqm** netlist files. • Edited the "Creating Intel Quartus Prime Projects for Multiple EDIF Files" on page 15–30 section for changes with the incremental compilation flow. • Editorial changes.
July 2010	10.0.0	• Minor updates for the Intel Quartus Prime software version 10.0 release
November 2009	9.1.0	• Minor updates for the Intel Quartus Prime software version 9.1 release

continued...

Date	Version	Changes
March 2009	9.0.0	• Updated list of supported devices for the Intel Quartus Prime software version 9.0 release • Chapter 11 was previously Chapter 10 in software version 8.1
November 2008	8.1.0	• Changed to 8-1/2 x 11 page size • Title changed to *Mentor Graphics Precision Synthesis Support* • Updated list of supported devices • Added information about the Precision RTL Plus incremental synthesis flow • Updated Figure 10-1 to include SystemVerilog • Updated "Guidelines for Intel FPGA IP and Architecture-Specific Features" on page 10–19 • Updated "Incremental Compilation and Block-Based Design" on page 10–28 • Added section "Creating Partitions with the incr_partition Attribute" on page 10–29
May 2008	8.0.0	• Removed Mercury from the list of supported devices • Changed Precision version to 2007a update 3 • Added note for Stratix IV support • Renamed "Creating a Project and Compiling the Design" section to "Creating and Compiling a Project in the Precision RTL Synthesis Software" • Added information about constraints in the Tcl file • Updated document based on the Intel Quartus Prime software version 8.0

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Debug Tools

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Online Version
Send Feedback
UG-20182
683552
2018.09.24

Contents

1. System Debugging Tools Overview 7
1.1. System Debugging Tools Portfolio 7
1.1.1. System Debugging Tools Comparison 7
1.1.2. Suggested Tools for Common Debugging Requirements 8
1.1.3. Debugging Ecosystem 9
1.2. Tools for Monitoring RTL Nodes 10
1.2.1. Resource Usage 10
1.2.2. Pin Usage 12
1.2.3. Usability Enhancements 12
1.3. Stimulus-Capable Tools 13
1.3.1. In-System Sources and Probes 14
1.3.2. In-System Memory Content Editor 14
1.3.3. System Console 15
1.4. Virtual JTAG Interface Intel FPGA IP 15
1.5. System-Level Debug Fabric 16
1.6. System Debugging Tools Overview Revision History 16

2. Analyzing and Debugging Designs with System Console 17
2.1. Introduction to System Console 17
2.2. System Console Debugging Flow 18
2.3. IP Cores that Interact with System Console 19
2.3.1. Services Provided through Debug Agents 19
2.4. Starting System Console 20
2.4.1. Starting System Console from Nios II Command Shell 20
2.4.2. Starting Stand-Alone System Console 20
2.4.3. Starting System Console from Platform Designer (Standard) 20
2.4.4. Starting System Console from Intel Quartus Prime 20
2.4.5. Customizing Startup 20
2.5. System Console GUI 21
2.5.1. System Explorer Pane 22
2.6. System Console Commands 23
2.7. Running System Console in Command-Line Mode 25
2.8. System Console Services 26
2.8.1. Locating Available Services 26
2.8.2. Opening and Closing Services 27
2.8.3. SLD Service 27
2.8.4. In-System Sources and Probes Service 28
2.8.5. Monitor Service 30
2.8.6. Device Service 32
2.8.7. Design Service 33
2.8.8. Bytestream Service 34
2.8.9. JTAG Debug Service 35
2.9. Working with Toolkits 36
2.9.1. Convert your Dashboard Scripts to Toolkit API 36
2.9.2. Creating a Toolkit Description File 36
2.9.3. Registering a Toolkit 37
2.9.4. Launching a Toolkit 37

2.9.5. Matching Toolkits with IP Cores....38
2.9.6. Toolkit API....38
2.10. ADC Toolkit....75
2.10.1. ADC Toolkit Terms....78
2.10.2. Setting the Frequency of the Reference Signal....78
2.10.3. Tuning the Signal Generator....79
2.10.4. Running a Signal Quality Test....81
2.10.5. Running a Linearity Test....82
2.10.6. ADC Toolkit Data Views....82
2.11. System Console Examples and Tutorials....85
2.11.1. Nios II Processor Example....85
2.12. On-Board Intel FPGA Download Cable II Support....87
2.13. MATLAB and Simulink* in a System Verification Flow....87
2.13.1. Supported MATLAB API Commands....89
2.13.2. High Level Flow....89
2.14. Deprecated Commands....89
2.15. Analyzing and Debugging Designs with the System Console Revision History....90

3. Debugging Transceiver Links....92
3.1. Channel Manager....92
3.1.1. Channel Display Modes....94
3.2. Transceiver Debugging Flow Walkthrough....94
3.3. Modifying the Design to Enable Transceiver Debug....94
3.3.1. Adapting an Intel FPGA Design Example....94
3.3.2. Stratix V Debug System Configuration....97
3.3.3. Instantiating and Parameterizing Intel Arria 10 Debug IP cores....103
3.4. Programming the Design into an Intel FPGA....105
3.5. Loading the Design in the Transceiver Toolkit....106
3.6. Linking Hardware Resources....106
3.6.1. Linking One Design to One Device....108
3.6.2. Linking Two Designs to Two Devices....108
3.6.3. Linking One Design on Two Devices....108
3.6.4. Linking Designs and Devices on Separate Boards....109
3.6.5. Verifying Hardware Connections....109
3.7. Identifying Transceiver Channels....110
3.7.1. Controlling Transceiver Channels....110
3.8. Creating Transceiver Links....110
3.9. Running Link Tests....110
3.9.1. Running BER Tests....111
3.9.2. Signal Eye Margin Testing (Stratix V only)....111
3.9.3. Running Custom Traffic Tests (Stratix V only)....113
3.9.4. Link Optimization Tests....114
3.10. Controlling PMA Analog Settings....115
3.10.1. Intel Arria 10 and Intel Cyclone 10 GX PMA Settings....115
3.11. User Interface Settings Reference....119
3.12. Troubleshooting Common Errors....123
3.13. Scripting API Reference....123
3.13.1. Transceiver Toolkit Commands....123
3.13.2. Data Pattern Generator Commands....130
3.13.3. Data Pattern Checker Commands....132
3.14. Debugging Transceiver Links Revision History....134

4. Quick Design Debugging Using Signal Probe..... 136
4.1. Design Flow Using Signal Probe.....136
4.1.1. Perform a Full Compilation.....136
4.1.2. Reserve Signal Probe Pins..... 137
4.1.3. Assign Signal Probe Sources..... 137
4.1.4. Add Registers Between Pipeline Paths and Signal Probe Pins..... 137
4.1.5. Perform a Signal Probe Compilation..... 138
4.1.6. Analyze the Results of a Signal Probe Compilation..... 138
4.1.7. What a Signal Probe Compilation Does..... 139
4.1.8. Understanding the Results of a Signal Probe Compilation..... 139
4.2. Scripting Support.....141
4.2.1. Making a Signal Probe Pin..... 141
4.2.2. Deleting a Signal Probe Pin.....141
4.2.3. Enabling a Signal Probe Pin..... 142
4.2.4. Disabling a Signal Probe Pin.....142
4.2.5. Performing a Signal Probe Compilation.....142
4.2.6. Reserving Signal Probe Pins.....142
4.2.7. Adding Signal Probe Sources.....143
4.2.8. Assigning I/O Standards.....143
4.2.9. Adding Registers for Pipelining..... 143
4.2.10. Running Signal Probe Immediately After a Full Compilation..... 144
4.2.11. Running Signal Probe Manually.....144
4.2.12. Enabling or Disabling All Signal Probe Routing..... 144
4.2.13. Allowing Signal Probe to Modify Fitting Results..... 144
4.3. Quick Design Debugging Using Signal Probe Revision History..... 144

5. Design Debugging with the Signal Tap Logic Analyzer..... 146
5.1. The Signal Tap Logic Analyzer.....146
5.1.1. Hardware and Software Requirements.....147
5.1.2. Signal Tap Logic Analyzer Features and Benefits 147
5.1.3. Backward Compatibility with Previous Versions of Intel Quartus Prime Software.....148
5.2. Signal Tap Logic Analyzer Task Flow Overview.....148
5.2.1. Add the Signal Tap Logic Analyzer to Your Design..... 149
5.2.2. Configure the Signal Tap Logic Analyzer..... 149
5.2.3. Define Trigger Conditions..... 150
5.2.4. Compile the Design..... 150
5.2.5. Program the Target Device or Devices.....150
5.2.6. Run the Signal Tap Logic Analyzer..... 150
5.2.7. View, Analyze, and Use Captured Data.....151
5.3. Configuring the Signal Tap Logic Analyzer.....151
5.3.1. Assigning an Acquisition Clock..... 151
5.3.2. Adding Signals to the Signal Tap File.....152
5.3.3. Adding Signals with a Plug-In..... 155
5.3.4. Adding Finite State Machine State Encoding Registers..... 156
5.3.5. Specifying Sample Depth..... 157
5.3.6. Capture Data to a Specific RAM Type.....157
5.3.7. Select the Buffer Acquisition Mode.....158
5.3.8. Specifying Pipeline Settings..... 160
5.3.9. Filtering Relevant Samples..... 160
5.3.10. Manage Multiple Signal Tap Files and Configurations.....167

5.4. Defining Triggers....169
5.4.1. Basic Trigger Conditions....169
5.4.2. Comparison Trigger Conditions....170
5.4.3. Advanced Trigger Conditions....172
5.4.4. Custom Trigger HDL Object....175
5.4.5. Trigger Condition Flow Control....178
5.4.6. Specify Trigger Position....190
5.4.7. Power-Up Triggers....191
5.4.8. External Triggers....193
5.5. Compiling the Design....193
5.5.1. Faster Compilations with Intel Quartus Prime Incremental Compilation....194
5.5.2. Prevent Changes Requiring Recompilation....195
5.5.3. Verify Whether You Need to Recompile Your Project....196
5.5.4. Incremental Route with Rapid Recompile....196
5.5.5. Timing Preservation with the Signal Tap Logic Analyzer....198
5.5.6. Performance and Resource Considerations....198
5.6. Program the Target Device or Devices....199
5.6.1. Ensure Setting Compatibility Between .stp and .sof Files....200
5.7. Running the Signal Tap Logic Analyzer....200
5.7.1. Runtime Reconfigurable Options....201
5.7.2. Signal Tap Status Messages....203
5.8. View, Analyze, and Use Captured Data....204
5.8.1. Capturing Data Using Segmented Buffers....204
5.8.2. Differences in Pre-Fill Write Behavior Between Different Acquisition Modes....206
5.8.3. Creating Mnemonics for Bit Patterns....207
5.8.4. Automatic Mnemonics with a Plug-In....207
5.8.5. Locating a Node in the Design....208
5.8.6. Saving Captured Data....208
5.8.7. Exporting Captured Data to Other File Formats....209
5.8.8. Creating a Signal Tap List File....209
5.9. Other Features....209
5.9.1. Creating Signal Tap File from Design Instances....209
5.9.2. Using the Signal Tap MATLAB MEX Function to Capture Data....211
5.9.3. Using Signal Tap in a Lab Environment....213
5.9.4. Remote Debugging Using the Signal Tap Logic Analyzer....213
5.9.5. Using the Signal Tap Logic Analyzer in Devices with Configuration Bitstream Security....214
5.9.6. Monitor FPGA Resources Used by the Signal Tap Logic Analyzer....214
5.10. Design Example: Using Signal Tap Logic Analyzers....214
5.11. Custom Triggering Flow Application Examples....214
5.11.1. Design Example 1: Specifying a Custom Trigger Position....215
5.11.2. Design Example 2: Trigger When triggercond1 Occurs Ten Times between triggercond2 and triggercond3....216
5.12. Signal Tap Scripting Support....216
5.12.1. Signal Tap Command-Line Options....216
5.12.2. Data Capture from the Command Line....217
5.13. Design Debugging with the Signal Tap Logic Analyzer Revision History....218

7. In-System Debugging Using External Logic Analyzers....220

7.1. About the Intel Quartus Prime Logic Analyzer Interface....220
7.2. Choosing a Logic Analyzer....220

7.2.1. Required Components....221
7.3. Flow for Using the LAI....222
7.3.1. Defining Parameters for the Logic Analyzer Interface....223
7.3.2. Mapping the LAI File Pins to Available I/O Pins....223
7.3.3. Mapping Internal Signals to the LAI Banks....224
7.3.4. Compiling Your Intel Quartus Prime Project....224
7.3.5. Programming Your Intel-Supported Device Using the LAI....225
7.4. Controlling the Active Bank During Runtime....225
7.4.1. Acquiring Data on Your Logic Analyzer....225
7.5. Using the LAI with Incremental Compilation....226
7.6. LAI Core Parameters....226
7.7. In-System Debugging Using External Logic Analyzers Revision History....227

8. In-System Modification of Memory and Constants....228
8.1. Setting Up In-System Modifiable Memories and Constants....228
8.2. Running the In-System Memory Content Editor....229
8.2.1. Instance Manager....229
8.2.2. Editing Data Displayed in the Hex Editor Pane....229
8.2.3. Importing and Exporting Memory Files....230
8.2.4. Scripting Support....230
8.2.5. Programming the Device with the In-System Memory Content Editor....230
8.2.6. Example: Using the In-System Memory Content Editor with the Signal Tap Logic Analyzer....230
8.3. In-System Modification of Memory and Constants Revision History....231

9. Design Debugging Using In-System Sources and Probes....232
9.1. Hardware and Software Requirements....234
9.2. Design Flow Using the In-System Sources and Probes Editor....234
9.2.1. Instantiating the In-System Sources and Probes IP Core....235
9.2.2. In-System Sources and Probes IP Core Parameters....236
9.3. Compiling the Design....236
9.4. Running the In-System Sources and Probes Editor....237
9.4.1. In-System Sources and Probes Editor GUI....237
9.4.2. Programming Your Device With JTAG Chain Configuration....237
9.4.3. Instance Manager....238
9.4.4. In-System Sources and Probes Editor Pane....238
9.5. Tcl interface for the In-System Sources and Probes Editor....240
9.6. Design Example: Dynamic PLL Reconfiguration....242
9.7. Design Debugging Using In-System Sources and Probes Revision History....244

A. Intel Quartus Prime Standard Edition User Guides....246

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	• Provides real-time in-system debugging capabilities. • 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.	You need to perform system-level debugging. For example, if you have an Avalon®-MM slave or Avalon-ST interfaces, you can debug the design at a transaction level.
Transceiver Toolkit	• Allows you to test and tune transceiver link signal quality through a combination of metrics. • Auto Sweeping of physical medium attachment (PMA) settings help you find optimal parameter values.	You need to debug or optimize signal integrity of a board layout even before finishing the design.
Signal Tap Logic Analyzer	• Uses FPGA resources. • Samples test nodes, and outputs the information to the Intel Quartus Prime software for display and analysis.	You have spare on-chip memory and you want functional verification of a design running in hardware.

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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.	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.

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. 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.

continued...

Requirement	Signal Probe	Logic Analyzer Interface (LAI)	Signal Tap Logic Analyzer	Description
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	—	X	Signal Probe and Signal Tap do not require a host for debugging purposes. A FPGA design with the LAI requires 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.
2. Valid when you use incremental compilation.

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.

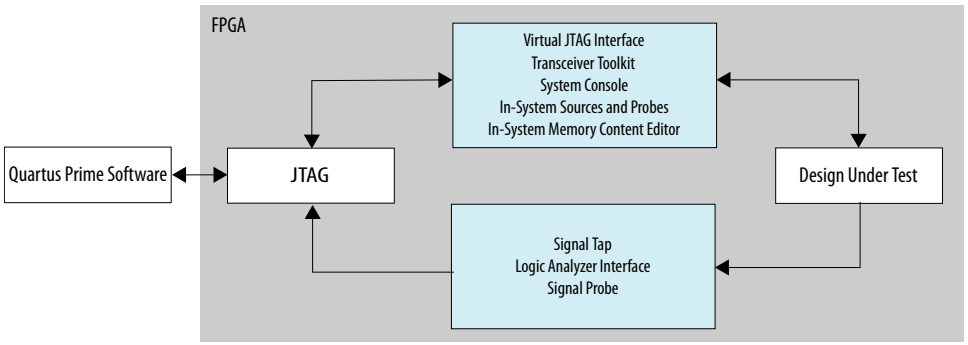
A very important distinction in the system debugging tools is how they interact with the design. All debugging tools in the Intel Quartus Prime software allow you to read the information from the design node, but only a subset allow you to input data at runtime:

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: • Read data from breakpoints that you define • Input values into your design during runtime
Virtual JTAG Interface			
System Console			
Transceiver Toolkit			
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 or CPLD.

- In cases when the design has spare logic and memory resources, the Signal Tap Logic Analyzer can providing fast functional verification of the design running on actual hardware.

  Note: CPLDs do not support the Signal Tap Logic Analyzer, because these devices do not have available memory resources.

- 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

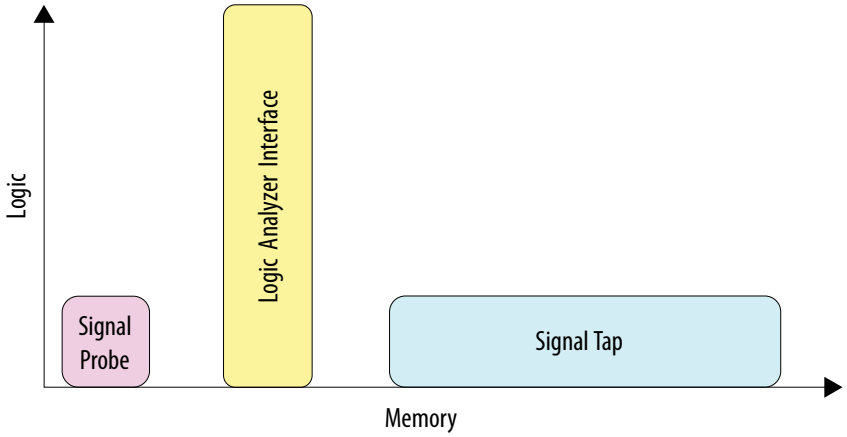
- Design Debugging with the Signal Tap Logic Analyzer on page 146
- In-System Debugging Using External Logic Analyzers on page 220

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.

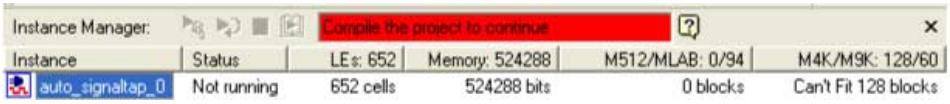
1.2.1.1.3. For Logic Analyzer Interface

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.

1.2.1.2. Resource Estimation

The resource estimation feature for the Signal Tap Logic Analyzer and the LAI allows you to quickly judge if enough on-chip resources are available before compiling the tool with your design.

Figure 3. Resource Estimator



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 Compilation

All three tools allow you to find and configure your debugging setup quickly. In addition, the Intel Quartus Prime incremental compilation feature and the Intel Quartus Prime incremental routing feature allow for a fast turnaround time for your programming file, increasing productivity and enabling fast debugging closure.

Both the LAI and Signal Tap Logic Analyzer support incremental compilation. With incremental compilation, you can add a Signal Tap Logic Analyzer instance or an LAI instance incrementally into your placed-and-routed design. This has the benefit of both preserving your timing and area optimizations from your existing design, and

decreasing the overall compilation time when any changes are necessary during the debugging process. With incremental compilation, you can save up to 70% compile time of a full compilation.

1.2.3.2. 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.

1.2.3.3. 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.

1.2.3.4. Remote Debugging

You can perform remote debugging of a system with the Intel Quartus Prime software via the System Console. This feature allows you to debug equipment deployed in the field through an existing TCP/IP connection.

- For information about setting up a Nios II system with the System Console to perform remote debugging, refer to Application Note 624
- For information about setting up an Intel FPGA SoC to perform remote debugging with the Intel Quartus Prime SLD tools, refer to Application Note 693.

Related Information

- Application Note 624: Debugging with System Console over TCP/IP
- Application Note 693: Remote Debugging over TCP/IP for Intel FPGA SoC

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 232

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

- Toolkit API on page 38
- Signal Tap Scripting Support on page 216

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 228

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 (Standard) 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 (Standard) 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 17

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. Test Link Signal Integrity with Transceiver Toolkit

Transceiver Toolkit runs from the System Console framework, and allows you to run automatic tests of your transceiver links for debugging and optimizing your transceiver designs. You can use the Transceiver Toolkit GUI to set up channel links in your transceiver devices and change parameters at runtime to measure signal integrity. For selected devices, the Transceiver Toolkit can also run and display eye contour tests.

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

1.5. System-Level Debug Fabric

During compilation, the Intel Quartus Prime generates the System-Level Debugging 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 System-Level Debugging Hub manages the sharing of JTAG resources.

Note: For System Console, you explicitly insert debug IP cores into the design to enable debugging.

The System-Level Debugging Hub appears in the project's design hierarchy as `sld_hub:sld_hub_inst`.

1.6. System Debugging Tools Overview Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide.
2018.05.07	18.0.0	• Moved here information about debug fabric on PR designs from the *Design Debugging with the Signal Tap Logic Analyzer* chapter.
2017.05.08	17.0.0	• Combined Altera JTAG Interface and Required Arbitration Logic topics into a new updated topic named System-Level Debugging Infrastructure.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	Added information that System Console supports the Tk toolkit.
November 2013	13.1.0	Dita conversion. Added link to Remote Debugging over TCP/IP for Altera SoC Application Note.
June 2012	12.0.0	Maintenance release.
November 2011	10.0.2	Maintenance release. Changed to new document template.
December 2010	10.0.1	Maintenance release. Changed to new document template.
July 2010	10.0.0	Initial release

2. Analyzing and Debugging Designs with System Console

2.1. Introduction to System Console

System Console provides visibility into your design and allows you to perform system-level debug on a FPGA at run-time. System Console performs tests on debug-enabled Platform Designer (Standard) instantiated IP cores. A variety of debug services provide read and write access to elements in your design. You can perform the following tasks with System Console and the tools built on top of System Console:

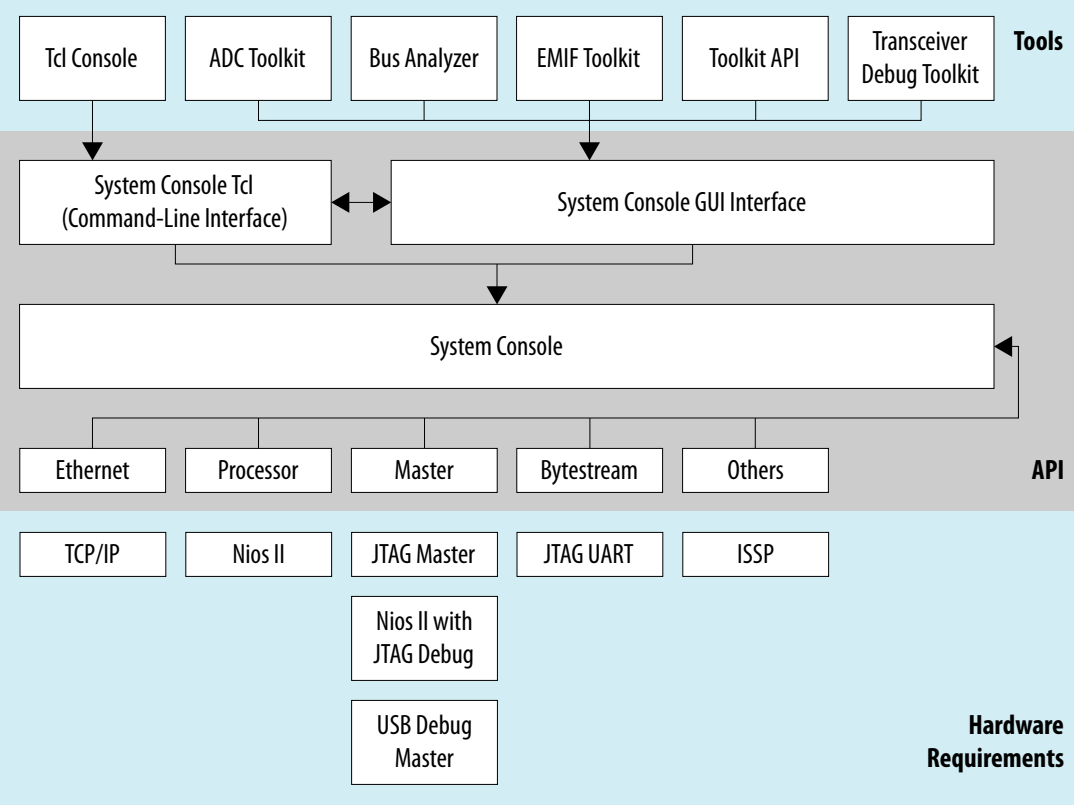
- Bring up boards with both finalized and partially complete designs.
- Perform remote debug with internet access.
- Automate run-time verification through scripting across multiple devices in your system.
- Test serial links with point-and-click configuration tuning in the Transceiver Toolkit.
- Debug memory interfaces with the External Memory Interface Toolkit.
- Integrate your debug IP into the debug platform.
- Test the performance of your ADC and analog chain on a Intel MAX® 10 device with the ADC Toolkit.
- Perform system verification with MATLAB*/Simulink.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 4. System Console Tools

(*Tools*) shows the applications that interact with System Console. The System Console API supports services that access your design in operation. Some services have specific hardware requirements.



Note: Use debug links to connect the host to the target you are debugging.

Related Information

- Introduction to Intel Memory Solution
  In *External Memory Interface Handbook Volume 1*
- Debugging Transceiver Links on page 92
- AN 693: Remote Hardware Debugging over TCP/IP for Intel SoC
- AN 624: Debugging with System Console over TCP/IP
- White Paper 01208: Hardware in the Loop from the MATLAB/Simulink Environment
- System Console Online Training

2.2. System Console Debugging Flow

To debug a design with the System Console, you must perform a series of steps:

1. Add an IP Core to the Platform Designer (Standard) system.
2. Generate the Platform Designer (Standard) system.
3. Compile the design.
4. Connect a board and program the FPGA.

5. Start the System Console.
6. Locate and open a System Console service.
7. Perform debug operations with the service.
8. Close the service.

2.3. 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 soft-logic embedded in some IP cores that enable debug communication with the host computer.

You instantiate debug IP cores using the Platform Designer (Standard) IP Catalog. Some IP cores are enabled for debug by default, while you can enable debug for other IP cores through options in the parameter editor. Some debug agents have multiple purposes.

When you use IP cores with embedded debug in your design, you can make large portions of the design accessible. Debug agents allow you to read and write to memory and alter peripheral registers from the host computer.

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`) provides the System Console with channel communication information. When System Console opens in the Intel Quartus Prime software or Platform Designer (Standard) while your design is open, any existing `.sof` is automatically found and linked to the detected running device. In a complex system, you may need to link the design and device manually.

2.3.1. 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 4. Common Services for System Console

Service	Function	Debug Agent Providing Service
master	Access memory-mapped (Avalon-MM or AXI) slaves connected to the master interface.	• Nios II with debug • JTAG to Avalon Master Bridge • USB Debug Master
slave	Allows the host to access a single slave without needing to know the location of the slave in the host's memory map. Any slave that is accessible to a System Console master 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 slaves controlled by a debug master provide the slave service.
processor	• Start, stop, or step the processor. • Read and write processor registers.	Nios II with debug
JTAG UART	The JTAG UART is an Avalon-MM slave device that you can use in conjunction with System Console to send and receive byte streams.	JTAG UART

Note: The following 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
- SLD Hub Controller System
- USB Debug Link

Related Information

- System Console Examples and Tutorials on page 85
- System Console Commands on page 23

2.4. Starting System Console

2.4.1. Starting System Console from 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`.
3. Type `--help` for System Console help.
4. Type `system-console --project_dir=<project directory>` to point to a directory that contains `.qsf` or `.sof` files.

2.4.2. Starting Stand-Alone System Console

You can get the stand-alone version of System Console as part of the Intel Quartus Prime software Programmer and Tools installer on the Altera website.

1. Navigate to the **Download Center** page and click the **Additional Software** tab.
2. On the Windows Start menu, click **All Programs ➤ Intel FPGA <*version*> ➤ Programmer and Tools ➤ System Console**.

Related Information

Intel Download Center

2.4.3. Starting System Console from Platform Designer (Standard)

Click **Tools ➤ System Console**.

2.4.4. Starting System Console from Intel Quartus Prime

Click **Tools ➤ System Debugging Tools ➤ System Console**.

2.4.5. Customizing Startup

You can customize your System Console 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.

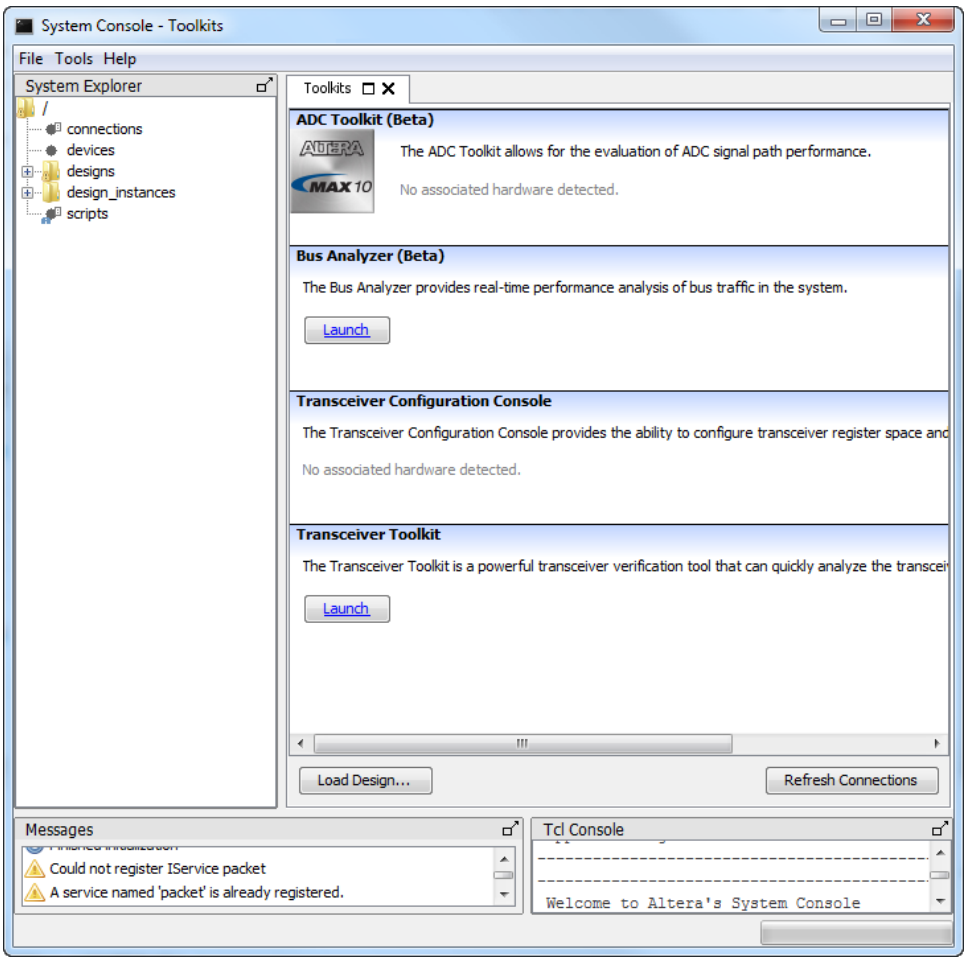
On startup, System Console automatically runs the Tcl commands in these files. The commands in the `system_console_rc.tcl` file run first, followed by the commands in the `rc_script.tcl` file.

2.5. System Console GUI

The System Console GUI consists of a main window with multiple panes, and allows you to interact with the design currently running on the host computer.

- **System Explorer**—Displays the hierarchy of the System Console virtual file system in your design, including board connections, devices, designs, and scripts.
- **Workspace**—Displays available toolkits including the ADC Toolkit, Transceiver Toolkit, Toolkits, GDB Server Control Panel, and Bus Analyzer. Click the **Tools** menu to launch applications.
- **Tcl Console**—A window that allows you to interact with your design using Tcl scripts, for example, sourcing scripts, writing procedures, and using System Console API.
- **Messages**—Displays status, warning, and error messages related to connections and debug actions.

Figure 5. System Console GUI



2.5.1. System Explorer Pane

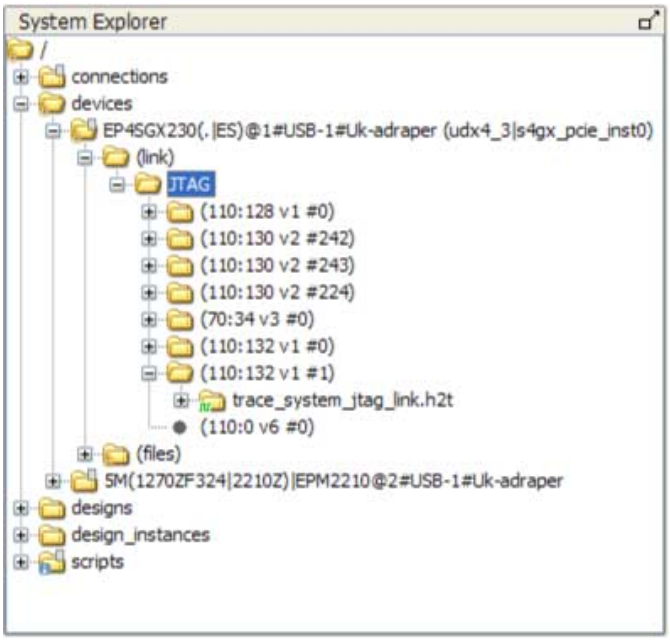
The **System Explorer** pane displays the virtual file system for all connected debugging IP cores, and contains the following information:

- **Devices** folder—Displays information about all devices connected to the System Console.
- **Scripts** folder—Stores scripts for easy execution.
- **Connections** folder—Displays information about the board connections visible to the System Console, such as Intel FPGA Download Cable. Multiple connections are possible.
- **Designs** folder—Displays information about Intel Quartus Prime designs connected to the System Console. Each design represents a loaded `.sof` file.

The **Devices** folder contains a sub-folder for each device connected to the System Console. Each device sub-folder contains a **(link)** folder, and may contain a **(files)** folder. The **(link)** folder shows debug agents (and other hardware) that System Console can access. The **(files)** folder contains information about the design files loaded from the Intel Quartus Prime project for the device.

Figure 6. System Explorer Pane

The figure shows the **EP4SGX230** folder under the **Device** folder, which contains a **(link)** folder. The **(link)** folder contains a **JTAG** folder, which describes the active debug connections to this device, for example, JTAG, USB, Ethernet, and agents connected to the EP4SGX230 device via a JTAG connection.



- Folders with a context menu display a context menu icon. Right-click these folders to view the context menu. For example, the **Connections** folder above shows a context menu icon.
- Folders that have messages display a message icon. Mouse-over these folders to view the messages. For example, the **Scripts** folder in the example has a message icon.
- Debug agents that sense the clock and reset state of the target show an information or error message with a clock status icon. The icon indicates whether the clock is running (information, green), stopped (error, red), or running but in reset (error, red). For example, the **trace_system_jtag_link.h2t** folder in the figure has a running clock.

2.6. System Console Commands

The console commands enable testing. Use console commands to identify a service by its path, and to open and close the connection. The `path` that identifies a service is the first argument to most System Console commands.

To initiate a service connection, do the following:

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

Note: For all Tcl commands, the *<format>* argument must come first.

Table 5. System Console Commands

Command	Arguments	Function
`get_service_types`	N/A	Returns a list of service types that System Console manages. Examples of service types include master, bytestream, processor, sld, jtag_debug, device, and design.
`get_service_paths`	• *<service-type>* • *<device>*—Returns services in the same specified device. The argument can be a device or another service in the device. • *<hpath>*—Returns services whose `hpath` starts with the specified prefix. • *<type>*—Returns services whose debug type matches this value. Particularly useful when opening slave services. • *<type>*—Returns services on the same development boards as the argument. Specify a board service, or any other service on the same board.	Allows you to filter the services which are returned.
`claim_service`	• *<service-type>* • *<service-path>* • *<claim-group>* • *<claims>*	Provides finer control of the portion of a service you want to use. `claim_service` returns a new path which represents a use of that service. Each use is independent. Calling claim_service multiple times returns different values each time, but each allows access to the service until closed.
`close_service`	• *<service-type>* • *<service-path>*	Closes the specified service type at the specified path.
`is_service_open`	• *<service-type>* • *<service-type>*	Returns 1 if the service type provided by the path is open, 0 if the service type is closed.
`get_services_to_add`	N/A	Returns a list of all services that are instantiable with the `add_service` command.
`add_service`	• *<service-type>* • *<instance-name>* • *optional-parameters*	Adds a service of the specified service type with the given instance name. Run `get_services_to_add` to retrieve a list of instantiable services. This command returns the path where the service was added.

continued...

Command	Arguments	Function
		Run `help add_service` <*service-type*> to get specific help about that service type, including any parameters that might be required for that service.
`add_service gdbserver`	• <*Processor Service*> • <*port number*>	Instantiates a gdbserver.
`add_service tcp`	• <*instance name*> • <*ip_addr*> • <*port_number*>	Allows you to connect to a TCP/IP port that provides a debug link over ethernet. See AN693 (*Remote Hardware Debugging over TCP/IP for Intel FPGA SoC*) for more information.
`add_service transceiver_channel_rx`	• <*data_pattern_checker*> • <*path*> • <*transceiver path*> • <*transceiver channel address*> • <*reconfig path*> • <*reconfig channel address*>	Instantiates a Transceiver Toolkit receiver channel.
`add_service transceiver_channel_tx`	• <*data_pattern_generator*> • <*path*> • <*transceiver path*> • <*transceiver channel address*> • <*reconfig path*> • <*reconfig channel address*>	Instantiates a Transceiver Toolkit transmitter channel.
`add_service transceiver_debug_link`	• <*transceiver_channel_tx path*> • <*transceiver_channel_rx path*>	Instantiates a Transceiver Toolkit debug link.
`get_version`	N/A	Returns the current System Console version and build number.
`get_claimed_services`	• <*claim*>	For the given claim group, returns a list of services claimed. The returned list consists of pairs of paths and service types. Each pair is one claimed service.
`refresh_connections`	N/A	Scans for available hardware and updates the available service paths if there have been any changes.
`send_message`	• <*level*> • <*message*>	Sends a message of the given level to the message window. Available levels are info, warning, error, and debug.

Related Information

Remote Hardware Debugging over TCP/IP for SoC Devices

2.7. 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.

- `--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.

2.8. System Console Services

Intel's System Console services provide access to hardware modules instantiated in your FPGA. Services vary in the type of debug access they provide.

2.8.1. 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 1. 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 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 2. Marker_node_info

Use the `marker_node_info` command to get information about the discovered services.

```
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)
```

2.8.2. 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 3. 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 4. Closing a Service

```
close_service master $claim_path; #Closes the service.
```

2.8.3. 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 5. 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 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

2.8.3.1. SLD Commands

Table 6. 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. 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 throwing a Tcl error. You can use the catch command if you want to handle the error.
`sld_unlock`	*<service-path>*	Unlocks the SLD chain.

2.8.4. In-System Sources and Probes Service

The In-System Sources and Probes (ISSP) service provides scriptable access to the `altsource_probe` IP core in a similar manner to using the **In-System Sources and Probes Editor** in the Intel Quartus Prime software.

Example 6. 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
```

The currently set source data can also be retrieved.

```
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
```

2.8.4.1. In-System Sources and Probes Commands

Note: The valid values for ISSP claims include `read_only`, `normal`, and `exclusive`.

Table 7. 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`

continued...

Command	Arguments	Function
		`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.

2.8.5. Monitor Service

The monitor service builds on top of the master service to allow reads of Avalon-MM slaves 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 master service manually for the reads.

Example 7. Monitor Service

1. Determine the master 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 master, 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 master 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.

2.8.5.1. Monitor Commands

You can use the Monitor commands to read many Avalon-MM slave 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 8. 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 master 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 master service.

continued...

Command	Arguments	Function
		<address> and *<size>* are relative to the master 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`.
`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.

2.8.6. Device Service

The device service supports device-level actions.

Example 8. 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. master 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
```

2.8.6.1. 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 9. 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.

2.8.7. Design Service

You can use design service commands to work with Intel Quartus Prime design information.

Example 9. Load

When you open System Console from the Intel Quartus Prime software or Platform Designer (Standard), the current project's debug information is sourced automatically if the `.sof` has been built. In other situations, you can load manually.

```
set sof_path [file join project_dir output_files project_name.sof]
set design [design_load $sof_path]
```

System Console is now aware that this particular `.sof` has been loaded.

Example 10. Linking

Once a `.sof` is loaded, System Console automatically links design information to the connected device. The resultant link persists and you can choose to unlink or reuse the link on an equivalent device with the same `.sof`.

You can perform manual linking.

```
set device_index 0; # Device index for our target
set device [lindex [get_service_paths device] $device_index]
design_link $design $device
```

Manually 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`.

2.8.7.1. Design Service Commands

Design service commands load and work with your design at a system level.

Table 10. Design Service Commands

Command	Arguments	Function
`design_load`	*<quartus-project-path>*, *<sof-file-path>*, or *<qpf-file-path>*	Loads a model of a 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 a Intel Quartus Prime logical design with a physical device. For example, you can link a 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. 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.

2.8.8. 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 Altera JTAG UART or the Avalon-ST JTAG interface.

Example 11. 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
```

2.8.8.1. Bytestream Commands

Table 11. 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.

2.8.9. 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]
```

2.8.9.1. JTAG Debug Commands

JTAG Debug commands help debug the JTAG Chain connected to a device.

Table 12. 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.

continued...

Command	Argument	Function
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.

2.9. Working with Toolkits

The Toolkit API allows you to create custom tools to visualize and interact with your design debug data. The Toolkit API provides graphical widgets in the form of buttons and text fields, which can leverage user input to interact with debug logic. You can use Toolkit API with the Intel Quartus Prime software versions 14.1 and later. The Toolkit API is the successor to the Dashboard service.

Toolkits you create with the Toolkit API require the following files:

- XML file that describes the toolkit (`.toolkit` file).
- Tcl file that implements the toolkit GUI.

2.9.1. Convert your Dashboard Scripts to Toolkit API

Convert your Dashboard scripts to work with the Toolkit API by following these steps:

1. Create a `.toolkit` file.
2. Modify your dashboard script:
   a. Remove the `add_service dashboard <name of service>` command.
   b. Change `dashboard_<command>` to `toolkit_<command>`.
   c. Change `open_service` to `claim_service`

   For example:

```
open_service slave $path
master_read_memory $path address count
```

   becomes

```
set c [claim_service slave $path lib {}]
master_read_memory $c address count
```

2.9.2. Creating a Toolkit Description File

A toolkit description file (`.toolkit`) is a XML file which provides the registration data for a toolkit.

Include the following attributes in your toolkit description file:

Table 13. Attributes in Toolkit Description File

Attribute name	Purpose
`name`	Internal toolkit file name.
`displayName`	Toolkit display name to appear in the GUI.
`addMenuItem`	Whether the System Console **Tools ➤ Toolkits** menu displays the toolkit.

Table 14. Toolkit child elements

Attribute name	Purpose
`description`	Description of the purpose of the toolkit.
`file`	Path to `.tcl` file containing the toolkit implementation.
`icon`	Path to icon to display as the toolkit launcher button in System Console *Note:* The `.png` 64x64 format is preferred. If the icon does not take up the whole space, ensure that the background is transparent.
`requirement`	If the toolkit works with a particular type of hardware, this attribute specifies the debug type name of the hardware. This attribute enables automatic discovery of the toolkit. The syntax of a toolkit's debug type name is: • Name of the `hw.tcl` component. • dot. • Name of the interface within that component which the toolkit uses. For example: `<hw.tcl name>.<interface name>`.

Example 12. .toolkit Description File

```
<?xml version="1.0" encoding="UTF-8"?>
        <toolkit name="toolkit_example" displayName="Toolkit Example"
addMenuItem="true">
        <file> toolkit_example.tcl </file>
        </toolkit>
```

Related Information

Matching Toolkits with IP Cores on page 38

2.9.3. Registering a Toolkit

Use the `toolkit_register` command in the System Console to make your toolkit available. Remember to specify the path to the `.toolkit` file. Registering a toolkit does not create an instance of the toolkit GUI.

```
toolkit_register <toolkit_file>
```

2.9.4. Launching a Toolkit

With the System Console, you can launch pre-registered toolkits in a number of ways:

- Click **Tools ➤ Toolkits**.
- Use the **Toolkits** tab. Each toolkit has a description, a detected hardware list, and a launch button.
- Use following command:

```
toolkit_open <.toolkit_file_name>
```

You can launch a toolkit in the context of a hardware resource associated with a toolkit type. If you use the command:

```
toolkit_open <toolkit_name> <context>
```

the toolkit Tcl can retrieve the context by typing

```
set context [toolkit_get_context]
```

Related Information

toolkit_get_context on page 49

2.9.5. Matching Toolkits with IP Cores

You can match your toolkit with any IP core:

- When searching for IP, the toolkit looks for debug markers and matches IP cores to the toolkit requirements. In the toolkit file, use the requirement attribute to specify a debug type, as follows:

```
<requirement><type>debug.type-name</type></requirement
```

- Create debug assignments in the `hw.tcl` for an IP core. `hw.tcl` files are available when you load the design in System Console.
- System Console discovers debug markers from identifiers in the hardware and associates with IP, without direct knowledge of the design.

2.9.6. Toolkit API

The Toolkit API service enables you to construct GUIs for visualizing and interacting with debug data. The Toolkit API is a graphical pane for the layout of your graphical widgets, which include buttons and text fields. Widgets pull data from other System Console services. Similarly, widgets use services to leverage user input to act on debug logic in your design.

Properties

Widget properties can push and pull information to the user interface. Widgets have properties specific to their type. For example, when you click a button, the button property `onClick` performs an action. A label widget does not have the same property, because the widget does not perform an action on click operation. However, both the button and label widgets have the `text` property to display text strings.

Layout

The Toolkit API service creates a widget hierarchy where the toolkit is at the top-level. The service implements group-type widgets that contain child widgets. Layout properties dictate layout actions that a parent performs on its children. For example,

the `expandableX` property when set as `True`, expands the widget horizontally to encompass all of the available space. The `visible` property when set as `True` allows a widget to display in the GUI.

User Input

Some widgets allow user interaction. For example, the `textField` widget is a text box that allows user entries. Access the contents of the box with the `text` property. A Tcl script can either get or set the contents of the `textField` widget with the `text` property.

Callbacks

Some widgets perform user-specified actions, referred to as callbacks. The `textField` widget has the `onChange` property, which is called when text contents change. The `button` widget has the `onClick` property, which is called when you click a button. Callbacks update widgets or interact with services based on the contents of a text field, or the state of any other widget.

2.9.6.1. Customizing Toolkit API Widgets

Use the `toolkit_set_property` command to interact with the widgets that you instantiate. The `toolkit_set_property` command is most useful when you change part of the execution of a callback.

2.9.6.2. Toolkit API Script Examples

Example 13. Making the Toolkit Visible in System Console

Use the `toolkit_set_property` command to modify the `visible` property of the root toolkit. Use the word `self` if a property is applied to the entire toolkit. In other cases, refer to the root toolkit using `all`.

```
toolkit_set_property self visible true
```

Example 14. Adding Widgets

Use the `toolkit_add` command to add widgets.

```
toolkit_add my_button button all
```

The following commands add a label widget `my_label` to the root toolkit. In the GUI, the label appears as **Widget Label**.

```
set name "my_label"
set content "Widget Label"
toolkit_add $name label all
toolkit_set_property $name text $content
```

In the GUI, the displayed text changes to the new value. Add one more label:

```
toolkit_add my_label_2 label all
toolkit_set_property my_label_2 text "Another label"
```

The new label appears to the right of the first label.

To place the new label under the first, use the following command:

```
toolkit_set_property self itemsPerRow 1
```

Example 15. Gathering Input

To incorporate user input into your Toolkit API,

1. Create a text field using the following commands:

```
set name "my_text_field"
set widget_type "textField"
set parent "all"
toolkit_add $name $widget_type $parent
```

2. The widget size is very small. To make the widget fill the horizontal space, use the following command:

```
toolkit_set_property my_text_field expandableX true
```

3. Now, the text field is fully visible. You can type text into the field, on clicking. To retrieve the contents of the field, use the following command:

```
set content [toolkit_get_property my_text_field text]
puts $content
```

This command prints the contents into the console.

Example 16. Updating Widgets Upon User Events

When you use callbacks, the Toolkit API can also perform actions without interactive typing:

1. Start by defining a procedure that updates the first label with the text field contents:

```
proc update_my_label_with_my_text_field{
    set content [toolkit_get_property my_text_field text]
    toolkit_set_property my_label text $content
}
```

2. Run the `update_my_label_with_my_text_field` command in the Tcl Console. The first label now matches the text field contents.
3. Use the `update_my_label_with_my_text_field` command whenever the text field changes:

```
toolkit_set_property my_text_field onChange
update_my_label_with_my_text_field
```

The Toolkit executes the `onChange` property each time the text field changes. The execution of this property changes the first field to match what you type.

Example 17. Buttons

Use buttons to trigger actions.

1. To create a button that changes the second label:

```
proc append_to_my_label_2 {suffix} {
                        set old_text [toolkit_get_property my_label_2 text]
                        set new_text "${old_text}${suffix}"
                        toolkit_set_property my_label_2 text $new_text
                        }
```

```
                        set text_to_append ", and more"
                        toolkit_add my_button button all
                        toolkit_set_property my_button onClick
[append_to_my_label_2 $text_to_append]
```

2. Click the button to append some text to the second label.

Example 18. Groups

The property `itemsPerRow` dictates the laying out of widgets in a group. For more complicated layouts where the number of widgets per row is different, use nested groups. To add a new group with more widgets per row:

```
toolkit_add my_inner_group group all
toolkit_set_property my_inner_group itemsPerRow 2
toolkit_add inner_button_1 button my_inner_group
toolkit_add inner_button_2 button my_inner_group
```

These commands create a row with a group of two buttons. To make the nested group more seamless, remove the border with the group name using the following commands:

```
toolkit_set_property my_inner_group title ""
```

You can set the `title` property to any other string to ensure the display of the border and title text.

Example 19. Tabs

Use tabs to manage widget visibility:

```
toolkit_add my_tabs tabbedGroup all
toolkit_set_property my_tabs expandableX true
toolkit_add my_tab_1 group my_tabs
toolkit_add my_tab_2 group my_tabs
toolkit_add tabbed_label_1 label my_tab_1
toolkit_add tabbed_label_2 label my_tab_2
toolkit_set_property tabbed_label_1 text "in the first tab"
toolkit_set_property tabbed_label_2 text "in the second tab"
```

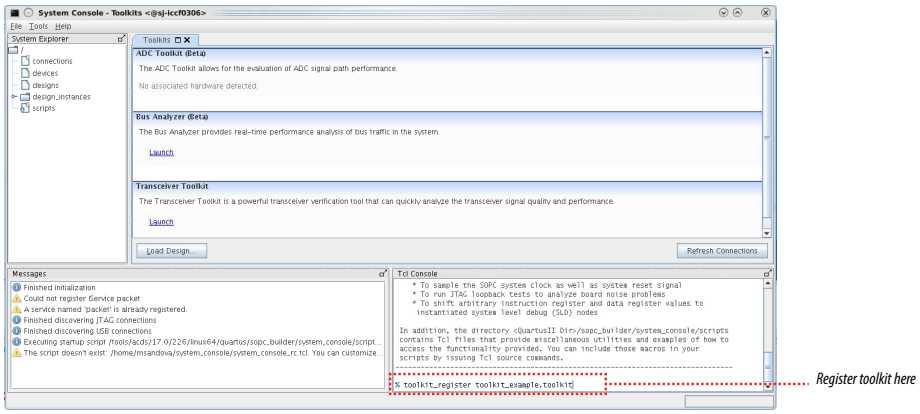
These commands add a set of two tabs, each with a group containing a label. Clicking on the tabs changes the displayed group/label.

2.9.6.3. Toolkit API GUI Example

This example shows how to register and launch a toolkit containing an interactive GUI window:

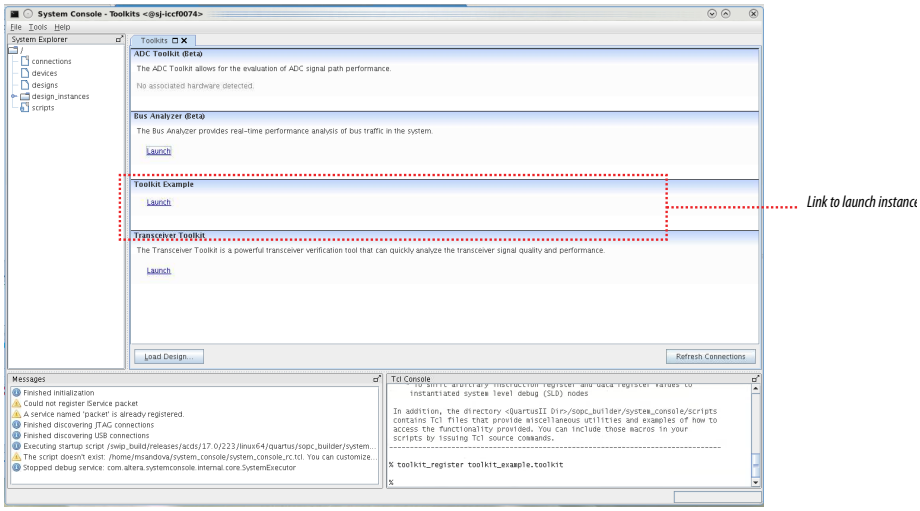
1. Write a toolkit description file. For a working example, refer to *Creating a Toolkit Description File*.
2. Generate a `.tcl` file using the text on *Toolkit API GUI Example .tcl File*.
3. Open the System Console.
4. Register your toolkit in the **Tcl Console** pane. Include the relative path to your file's location.

Figure 7. Registering Your Toolkit



The Toolkit appears in the **Toolkits** tab

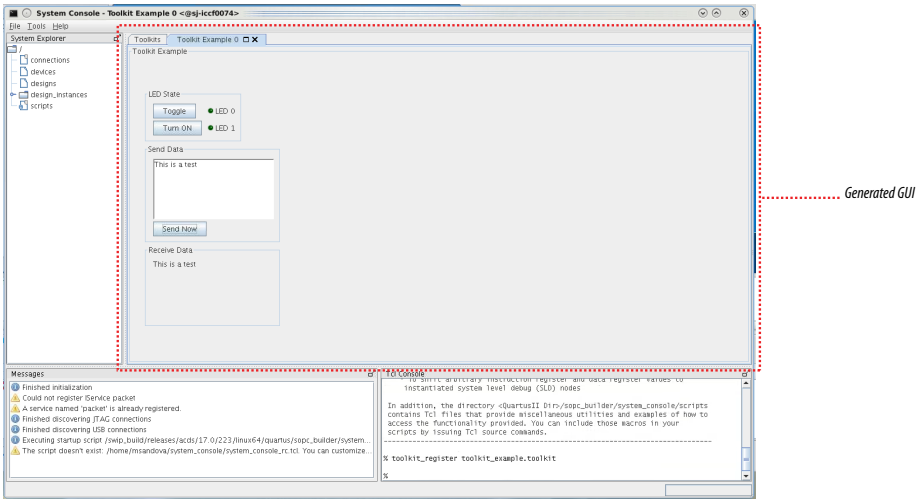
Figure 8. Toolkits Tab After Toolkit Example Registration



5. Click the Launch link.

   A new tab appears, containing the widgets you specified in the TCL file.

Figure 9. Toolkit Example GUI



When you insert text in the **Send Data** field and click **Launch**, the text appears in the **Receive Data** field.

Related Information

Creating a Toolkit Description File on page 36

2.9.6.3.1. Toolkit API GUI Example .tcl File

The following Toolkit API .tcl file creates a GUI window that provides debug interaction with your design.

```
namespace eval Test {

    variable ledValue 0
    variable dashboardActive 0
    variable Switch_off 1

    proc toggle { position } {
        set ::Test::ledValue ${position}
        ::Test::updateDashboard
        }

    proc sendText {} {
        set sendText [toolkit_get_property sendTextText text]
        toolkit_set_property receiveTextText text $sendText
        }

    proc dashBoard {} {

        if { ${::Test::dashboardActive} == 1 } {
            return -code ok "dashboard already active"
        }

        set ::Test::dashboardActive 1
        #
        # top group widget
        #
        toolkit_add  topGroup group self
        toolkit_set_property  topGroup expandableX false
        toolkit_set_property  topGroup expandableY false
        toolkit_set_property  topGroup itemsPerRow 1
        toolkit_set_property  topGroup title ""
```

```
        #
        # leds group widget
        #
        toolkit_add  ledsGroup group topGroup
        toolkit_set_property  ledsGroup expandableX false
        toolkit_set_property  ledsGroup expandableY false
        toolkit_set_property  ledsGroup itemsPerRow 2
        toolkit_set_property  ledsGroup title "LED State"

        #
        # leds widgets
        #
        toolkit_add  led0Button button ledsGroup
        toolkit_set_property  led0Button enabled true
        toolkit_set_property  led0Button expandableX false
        toolkit_set_property  led0Button expandableY false
        toolkit_set_property  led0Button text "Toggle"
        toolkit_set_property  led0Button onClick {::Test::toggle 1}

        toolkit_add  led0LED led ledsGroup
        toolkit_set_property  led0LED expandableX false
        toolkit_set_property  led0LED expandableY false
        toolkit_set_property  led0LED text "LED 0"
        toolkit_set_property  led0LED color "green_off"

        toolkit_add  led1Button button ledsGroup
        toolkit_set_property  led1Button enabled true
        toolkit_set_property  led1Button expandableX false
        toolkit_set_property  led1Button expandableY false
        toolkit_set_property  led1Button text "Turn ON"
        toolkit_set_property  led1Button onClick {::Test::toggle 2}

        toolkit_add  led1LED led ledsGroup
        toolkit_set_property  led1LED expandableX false
        toolkit_set_property  led1LED expandableY false
        toolkit_set_property  led1LED text "LED 1"
        toolkit_set_property  led1LED color "green_off"


        #
        # sendText widgets
        #
        toolkit_add  sendTextGroup group topGroup
        toolkit_set_property  sendTextGroup expandableX false
        toolkit_set_property  sendTextGroup expandableY false
        toolkit_set_property  sendTextGroup itemsPerRow 1
        toolkit_set_property  sendTextGroup title "Send Data"

        toolkit_add  sendTextText text sendTextGroup
        toolkit_set_property  sendTextText expandableX false
        toolkit_set_property  sendTextText expandableY false
        toolkit_set_property  sendTextText preferredWidth 200
        toolkit_set_property  sendTextText preferredHeight 100
        toolkit_set_property  sendTextText editable true
        toolkit_set_property  sendTextText htmlCapable false
        toolkit_set_property  sendTextText text ""

        toolkit_add  sendTextButton button sendTextGroup
        toolkit_set_property  sendTextButton enabled true
        toolkit_set_property  sendTextButton expandableX false
        toolkit_set_property  sendTextButton expandableY false
        toolkit_set_property  sendTextButton text "Send Now"
        toolkit_set_property  sendTextButton onClick {::Test::sendText}

        #
        # receiveText widgets
        #
        toolkit_add  receiveTextGroup group topGroup
        toolkit_set_property  receiveTextGroup expandableX false
        toolkit_set_property  receiveTextGroup expandableY false
        toolkit_set_property  receiveTextGroup itemsPerRow 1
```

```
        toolkit_set_property  receiveTextGroup title "Receive Data"

        toolkit_add  receiveTextText text receiveTextGroup
        toolkit_set_property  receiveTextText expandableX false
        toolkit_set_property  receiveTextText expandableY false
        toolkit_set_property  receiveTextText preferredWidth 200
        toolkit_set_property  receiveTextText preferredHeight 100
        toolkit_set_property  receiveTextText editable false
        toolkit_set_property  receiveTextText htmlCapable false
        toolkit_set_property  receiveTextText text ""

        return -code ok
    }

    proc updateDashboard {} {

        if { ${::Test::dashboardActive} > 0 } {

                toolkit_set_property  ledsGroup title "LED State"
                if { [ expr ${::Test::ledValue} & 0x01 &  \
                            ${::Test::Switch_off} ] } {
                    toolkit_set_property  led0LED color "green"
                    set ::Test::Switch_off  0
                } else {
                    toolkit_set_property  led0LED color "green_off"
                    set ::Test::Switch_off  1
                }
                if { [ expr ${::Test::ledValue} & 0x02 ] } {
                    toolkit_set_property  led1LED color "green"
                } else {
                    toolkit_set_property  led1LED color "green_off"
                }
        }
    }
}
::Test::dashBoard
```

2.9.6.4. Toolkit API Commands

Toolkit API commands run in the context of a unique toolkit instance.

toolkit_register on page 46
toolkit_open on page 47
get_quartus_ini on page 48
toolkit_get_context on page 49
toolkit_get_types on page 50
toolkit_get_properties on page 51
toolkit_add on page 52
toolkit_get_property on page 53
toolkit_set_property on page 54
toolkit_remove on page 55
toolkit_get_widget_dimensions on page 56

2.9.6.4.1. toolkit_register

Description

Point to the XML file that describes the plugin (`.toolkit` file) .

Usage

`toolkit_register` <*toolkit_file*>

Returns

No return value.

Arguments

<*toolkit_file*> Path to the toolkit definition file.

Example

```
toolkit_register /path/to/toolkit_example.toolkit
```

2.9.6.4.2. toolkit_open

Description

Opens an instance of a toolkit in System Console.

Usage

`toolkit_open` *<toolkit_id> [<context>]*

Returns

No return value.

Arguments

<toolkit_id> Name of the toolkit type to open.

<context> An optional context, such as a service path for a hardware resource that is associated with the toolkit that opens.

Example

```
toolkit_open my_toolkit_id
```

2.9.6.4.3. get_quartus_ini

Description

Returns the value of an `ini` setting from the Intel Quartus Prime software `.ini` file.

Usage

`get_quartus_ini` <*ini*> <*type*>

Returns

Value of `ini` setting.

Arguments

<*ini*> Name of the Intel Quartus Prime software `.ini` setting.

<*type*> (Optional) Type of `.ini` setting. The known types are `string` and `enabled`. If the type is `enabled`, the value of the `.ini` setting returns `1`, or `0` if not enabled.

Example

```
set my_ini_enabled [get_quartus_ini my_ini enabled]
```

```
set my_ini_raw_value [get_quartus_ini my_ini]
```

2.9.6.4.4. toolkit_get_context

Description

Returns the context that was specified when the toolkit was opened. If no context was specified, returns an empty string.

Usage

```
toolkit_get_context
```

Returns

Context.

Arguments

No arguments.

Example

```
set context [toolkit_get_context]
```

2.9.6.4.5. toolkit_get_types

Description

Returns a list of widget types.

Usage

```
toolkit_get_types
```

Returns

List of widget types.

Arguments

No arguments.

Example

```
set widget_names [toolkit_get_types]
```

2.9.6.4.6. toolkit_get_properties

Description

Returns a list of toolkit properties for a type of widget.

Usage

`toolkit_get_properties` <*widgetType*>

Returns

List of toolkit properties.

Arguments

<*widgetType*> Type of widget.

Example

```
set widget_properties [toolkit_get_properties xyChart]
```

2.9.6.4.7. toolkit_add

Description

Adds a widget to the current toolkit.

Usage

`toolkit_add` *<id>* *<type>*<*groupid*>

Returns

No return value.

Arguments

<id> A unique ID for the widget being added.

<type> The type of widget that is being added.

<groupid> The ID for the parent group that contains the new widget. Use `self` for the toolkit base group.

Example

```
toolkit_add my_button button parentGroup
```

2.9.6.4.8. toolkit_get_property

Description

Returns the property value for a specific widget.

Usage

`toolkit_get_property` <*id*> <*propertyName*>

Returns

The property value.

Arguments

<*id*> A unique ID for the widget being queried.

<*propertyName*> The name of the widget property.

Example

```
set enabled [toolkit_get_property my_button enabled]
```

2.9.6.4.9. toolkit_set_property

Description

Sets the property value for a specific widget.

Usage

`toolkit_set_property` <id><*propertyName*> <*value*>

Returns

No return value.

Arguments

<*id*> A unique ID for the widget being modified.

<*propertyName*> The name of the widget property being set.

<*value*> The new value for the widget property.

Example

```
toolkit_set_property my_button enabled 0
```

2.9.6.4.10. toolkit_remove

Description

Removes a widget from the specified toolkit.

Usage

`toolkit_remove <id>`

Returns

No return value.

Arguments

<id> A unique ID for the widget being removed.

Example

```
toolkit_remove my_button
```

2.9.6.4.11. toolkit_get_widget_dimensions

Description

Returns the width and height of the specified widget.

Usage

`toolkit_get_widget_dimensions <id>`

Returns

Width and height of specified widget.

Arguments

<id> A unique ID for the widget being added.

Example

```
set dimensions [toolkit_get_widget_dimensions my_button]
```

2.9.6.5. Toolkit API Properties

The following are the Toolkit API widget properties:

Widget Types and Properties on page 58
barChart Properties on page 59
button Properties on page 60
checkBox Properties on page 61
comboBox Properties on page 62
dial Properties on page 63
fileChooserButton Properties on page 64
group Properties on page 65
label Properties on page 66
led Properties on page 67
lineChart Properties on page 68
list Properties on page 69
pieChart Properties on page 70
table Properties on page 71
text Properties on page 72
textField Properties on page 73
timeChart Properties on page 74
xyChart Properties on page 75

2.9.6.5.1. Widget Types and Properties

Table 15. Toolkit API Widget Types and Properties

Name	Description
`enabled`	Enables or disables the widget.
`expandable`	Controls whether the widget is expandable.
`expandableX`	Allows the widget to resize horizontally if there is space available in the cell where it resides.
`expandableY`	Allows the widget to resize vertically if there is space available in the cell where it resides.
`foregroundColor`	Sets the foreground color.
`maxHeight`	If the widget's `expandableY` is set, this is the maximum height in pixels that the widget can take.
`minHeight`	If the widget's `expandableY` is set, this is the minimum height in pixels that the widget can take.
`maxWidth`	If the widget's `expandableX` is set, this is the maximum width in pixels that the widget can take.
`minWidth`	If the widget's `expandableX` is set, this is the minimum width in pixels that the widget can take.
`preferredHeight`	The height of the widget if `expandableY` is not set.
`preferredWidth`	The width of the widget if `expandableX` is not set.
`toolTip`	Implements a mouse-over tooltip.
`visible`	Displays the widget.

2.9.6.5.2. barChart Properties

Table 16. Toolkit API barChart Properties

Name	Description
`title`	Chart title.
`labelX`	X-axis label text.
`label`	X-axis label text.
`range`	Y-axis value range. By default, it is auto range. Specify the range using a Tcl list, for example: `[list lower_numerical_value upper_numerical_value]`.
`itemValue`	Specify the value using a Tcl list, for example: `[list bar_category_str numerical_value]`.

2.9.6.5.3. button Properties

Table 17. Toolkit API button Properties

Name	Description
`onClick`	Specifies the Tcl command to run every time you click the button. Usually the command is a `proc`.
`text`	The text on the button.

2.9.6.5.4. checkBox Properties

Table 18. Toolkit API checkBox Properties

Name	Description
`checked`	Specifies the state of the checkbox.
`onClick`	Specifies the Tcl command to run every time you click the checkbox. The command is usually a `proc`.
`text`	The text on the checkbox.

2.9.6.5.5. comboBox Properties

Table 19. Toolkit API comboBox Properties

Name	Description
`onChange`	A Tcl callback to run when the value of the combo box changes.
`options`	A list of items to display in the combo box.
`selectedItem`	The selected item in the combo box.

2.9.6.5.6. dial Properties

Table 20. Toolkit API dial Properties

Name	Description
`max`	The maximum value that the dial can show.
`min`	The minimum value that the dial can show.
`ticksize`	The space between the different tick marks of the dial.
`title`	The title of the dial.
`value`	The value that the dial's needle marks. It must be between min and max.

2.9.6.5.7. fileChooserButton Properties

Table 21. Toolkit API fileChooserButton Properties

Name	Description
`text`	The text on the button.
`onChoose`	A Tcl command that runs every time you click the button. The command is usually a `proc`.
`title`	The title of the dialog box.
`chooserButtonText`	The text of the dialog box approval button. Default value is `Open`.
`filter`	The file filter, based on extension. The filter supports only one extension. By default, the filter allows all file names. Specify the filter using the syntax `[list filter_description file_extension]`, for example: `[list "Text Document (.txt)" "txt"]`.
`mode`	Specifies what kind of files or directories you can select. The default is `files_only`. Possible options are `files_only` and `directories_only`.
`multiSelectionEnabled`	Controls whether you can select multiple files. Default value is `false`.
`paths`	This property is read-only. Returns a list of file paths selected in the file chooser dialog box. The property is most useful when you use it within the `onClick` script, or inside a procedure that updates the result after the dialog box closes.

2.9.6.5.8. group Properties

Table 22. Toolkit API group Properties

Name	Description
`itemsPerRow`	The number of widgets the group can position in one row, from left to right, before moving to the next row.
`title`	The title of the group. Groups with a title can have a border around them, and setting an empty title removes the border.

2.9.6.5.9. label Properties

Table 23. Toolkit API label Properties

Name	Description
`text`	The text to show in the label.

2.9.6.5.10. led Properties

Table 24. Toolkit API led Properties

Name	Description
`color`	The color of the LED. The options are: `red_off`, `red`, `yellow_off`, `yellow`, `green_off`, `green`, `blue_off`, `blue`, and `black`.
`text`	The text to show next to the LED.

2.9.6.5.11. lineChart Properties

Table 25. Toolkit API lineChart Properties

Name	Description
`title`	Chart title.
`labelX`	X-axis label text.
`labelY`	Y-axis label text.
`range`	Y-axis value range. By default, it is auto range. Specify the range using a Tcl list, for example: `[list lower_numerical_value upper_numerical_value]`.
`itemValue`	Item value. Specify the value using a Tcl list, for example: `[list bar_category_str numerical_value]`.

2.9.6.5.12. list Properties

Table 26. Toolkit API list Properties

Name	Description
`selected`	Index of the selected item in the combo box.
`options`	List of options to display.
`onChange`	A Tcl callback to run when the selected item in the list changes.

2.9.6.5.13. pieChart Properties

Table 27. Toolkit API pieChart Properties

Name	Description
`title`	Chart title.
`itemValue`	Item value. Specified using a Tcl list, for example: `[list bar_category_str numerical_value]`.

2.9.6.5.14. table Properties

Table 28. Toolkit API table Properties

Name	Description
columnCount	The number of columns (Mandatory) (0, by default).
rowCount	The number of rows (Mandatory) (0, by default).
headerReorderingAllowed	Controls whether you can drag the columns (false, by default).
headerResizingAllowed	Controls whether you can resize all column widths. (false, by default). *Note:* You can resize each column individually with the columnWidthResizable property.
rowSorterEnabled	Controls whether you can sort the cell values in a column (false, by default).
showGrid	Controls whether to draw both horizontal and vertical lines (true, by default).
showHorizontalLines	Controls whether to draw horizontal line (true, by default).
rowIndex	Current row index. Zero-based. This value affects some properties below (0, by default).
columnIndex	Current column index. Zero-based. This value affects all column specific properties below (0, by default).
cellText	Specifies the text inside the cell given by the current rowIndex and columnIndex (Empty, by default).
selectedRows	Control or retrieve row selection.
columnHeader	The text in the column header.
columnHeaders	A list of names to define the columns for the table.
columnHorizontalAlignment	The cell text alignment in the specified column. Supported types are leading (default), left, center, right, trailing.
columnRowSorterType	The type of sorting method. This is applicable only if rowSorterEnabled is true. Each column has its own sorting type. Possible types are string (default), int, and float.
columnWidth	The number of pixels in the column width.
columnWidthResizable	Controls whether the column width is resizable by you (false, by default).
contents	The contents of the table as a list. For a table with columns A, B, and C, the format of the list is {A1 B1 C1 A2 B2 C2 etc}.

2.9.6.5.15. text Properties

Table 29. Toolkit API text Properties

Name	Description
`editable`	Controls whether the text box is editable.
`htmlCapable`	Controls whether the text box can format HTML.
`text`	The text to show in the text box.

2.9.6.5.16. textField Properties

Table 30. Toolkit API textField Properties

Name	Description
`editable`	Controls whether the text box is editable.
`onChange`	A Tcl callback to run when you change the content of the text box.
`text`	The text in the text box.

2.9.6.5.17. timeChart Properties

Table 31. Toolkit API timeChart Properties

Name	Description
`labelX`	The label for the X-axis.
`labelY`	The label for the Y-axis.
`latest`	The latest value in the series.
`maximumItemCount`	The number of sample points to display in the historic record.
`title`	The title of the chart.
`range`	Sets the range for the chart. The range has the form `{low, high}`. The low/high values are doubles.
`showLegend`	Specifies whether a legend for the series is shown in the graph.

2.9.6.5.18. xyChart Properties

Table 32. Toolkit API xyChart Properties

Name	Properties
`title`	Chart title.
`labelX`	X-Axis label text.
`labelY`	Y-Axis label text.
`range`	Sets the range for the chart. The range is of the form `{low, high}`. The low/high values are doubles.
`maximumItemCount`	Specifies the maximum number of data values to keep in a data series. This setting only affects new data in the chart. If you add more data values than the `maximumItemCount`, only the last `maximumItemCount` number of entries are kept.
`series`	Adds a series of data to the chart. The first value in the spec is the identifier for the series. If the same identifier is set twice, the Toolkit API selects the most recent series. If the identifier does not contain series data, that series is removed from the chart. Specify the series in a Tcl list: `{identifier, x-1 y-1, x-2 y-2}`.
`showLegend`	Sets whether a legend for the series appears in the graph.

2.10. ADC Toolkit

The ADC Toolkit is designed to work with Intel MAX 10 devices and helps you understand the performance of the analog signal chain as seen by the on-board ADC hardware. The GUI displays the performance of the ADC using industry standard metrics. You can export the collected data to a `.csv` file and process this raw data yourself. The ADC Toolkit is built on the System Console framework and can only be operated using the GUI. There is no Tcl support for the tool.

Prerequisites for Using the ADC Toolkit

- Altera Modular ADC IP core
  - **External Reference Voltage** if you select **External** in the Altera Modular ADC IP parameters
- Reference signal

The ADC Toolkit needs a sine wave signal to be fed to the analog inputs. You need the capability to precisely set the level and frequency of the reference signal. A high-precision sine wave is needed for accurate test results; however, there are useful things that can be read in **Scope** mode with any input signal.

To achieve the best testing results, ensure that the reference signal has less distortion than the device ADC is able to resolve. Otherwise, you are adding distortions from the source into the resulting ADC distortion measurements. The limiting factor is based on hardware precision.

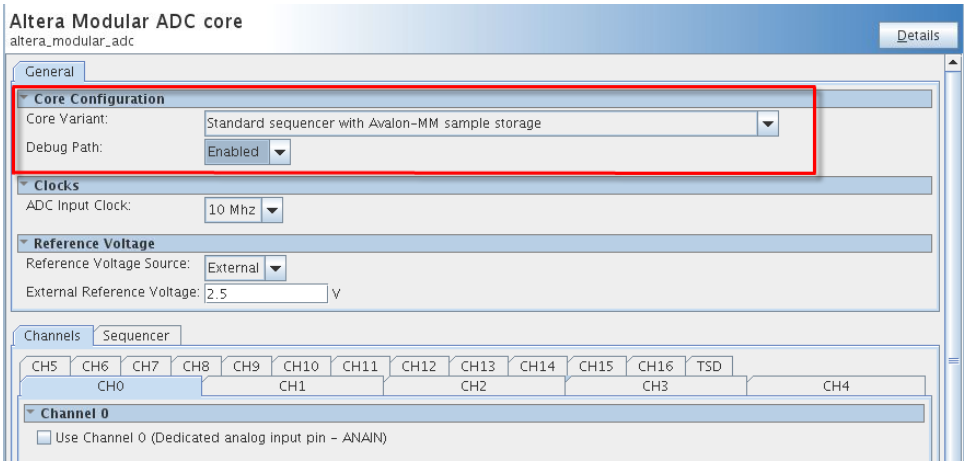
Note: When applying a sine wave, the ADC should sample at 2x the fundamental sine wave frequency. There should be a low-pass filter, 3dB point set to the fundamental frequency.

Configuring the Altera Modular ADC IP Core

The Altera Modular ADC IP core needs to be included in the design. You can instantiate this IP core from the **IP Catalog**. When you configure this IP core in the **Parameter Editor**, you need to enable the **Debug Path** option located under **Core Configuration**.

There are two limitations in the Intel Quartus Prime software v14.1 for the Altera Modular ADC IP core. The ADC Toolkit does not support the **ADC control core only** option under **Core Configuration**. You must select a core variant that uses the standard sequencer in order for the Altera Modular ADC IP core to work with ADC Toolkit. Also, if an Avalon Master is not connected to the sequencer, you must manually start the sequencer before the ADC Toolkit.

Figure 10. Altera Modular ADC Core



Starting the ADC Toolkit

You can launch the ADC Toolkit from System Console. Before starting the ADC toolkit, you need to verify that the board is programmed. You can then load the `.sof` by clicking **File ➤ Load Design**. If System Console was started with an active project, the design is auto-loaded when you start System Console.

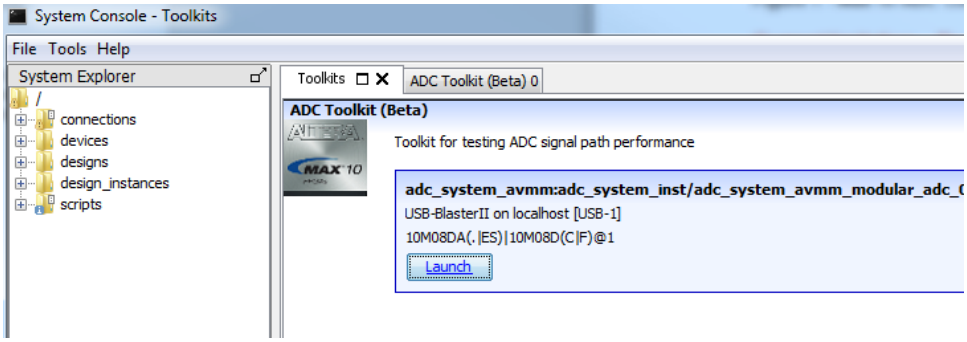
There are two methods to start the ADC Toolkit. Both methods require you to have a Intel MAX 10 device connected, programmed with a project, and linked to this project. However, the **Launch** command only shows up if these requirements are met. You can always start the ADC Toolkit from the **Tools** menu, but a successful connection still depends on meeting the above requirements.

- Click **Tools ➤ ADC Toolkit**
- Alternatively, click **Launch** from the **Toolkits** tab. The path for the device is displayed above the **Launch** button.

Note: Only one ADC Toolkit enabled device can be connected at a time.

Upon starting the ADC Toolkit, an identifier path on the ADC Toolkit tab shows you which ADC on the device is being used for this instance of the ADC Toolkit.

Figure 11. Launching ADC Toolkit

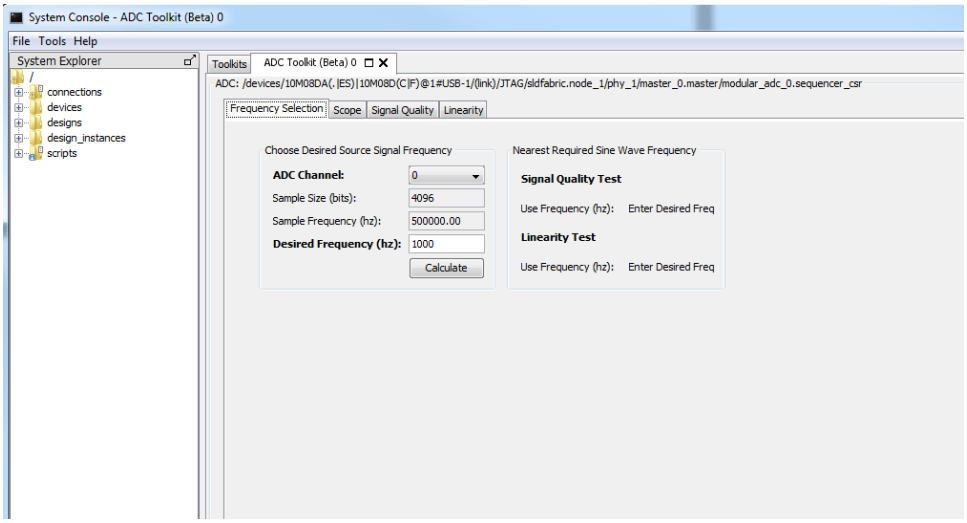


ADC Toolkit Flow

The ADC Toolkit GUI consists of four panels: **Frequency Selection**, **Scope**, **Signal Quality**, and **Linearity**.

1. Use the **Frequency Selection** panel to calculate the required sine wave frequency for proper signal quality testing. The ADC Toolkit provides the nearest ideal frequency based on the desired reference signal frequency.
2. Use the **Scope** panel to tune the signal generator or inspect input signal characteristics.
3. Use the **Signal Quality** panel to test the performance of the ADC using industry standard metrics.
4. Use the **Linearity** panel to test the linearity performance of the ADC and display differential and integral non-linearity results.

Figure 12. ADC Toolkit GUI



Related Information

- Using the ADC Toolkit in Intel MAX 10 Devices online training
- Intel MAX 10 FPGA Device Overview
- Intel MAX 10 FPGA Device Datasheet

- Intel MAX 10 FPGA Design Guidelines
- Intel MAX 10 Analog to Digital Converter User Guide
- Additional information about sampling frequency
  Nyquist sampling theorem and how it relates to the nominal sampling interval required to avoid aliasing.

2.10.1. ADC Toolkit Terms

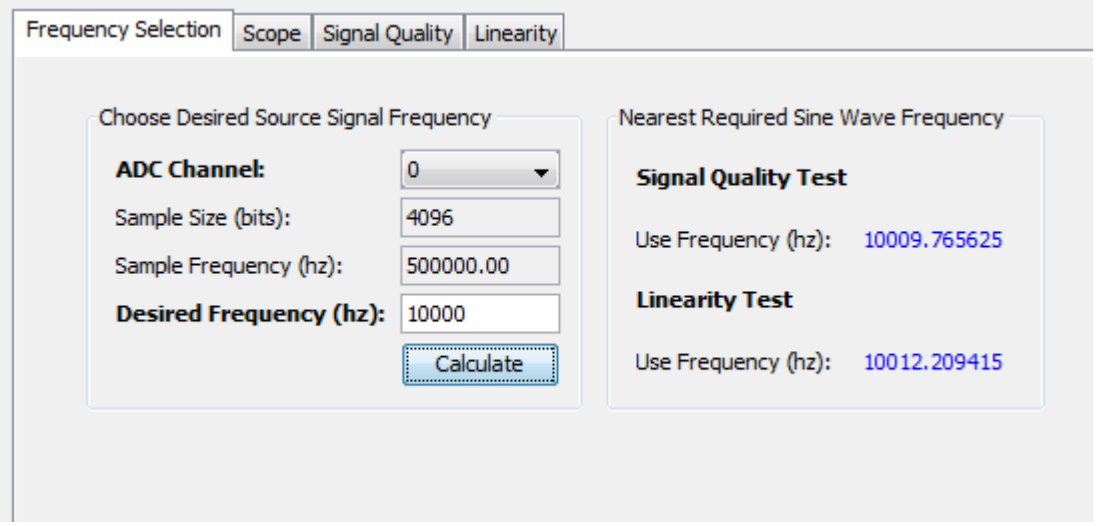
Table 33. ADC Toolkit Terms

Term	Description
SNR	The ratio of the output signal voltage level to the output noise level.
THD	The ratio of the sum of powers of the harmonic frequency components to the power of the fundamental/original frequency component.
SFDR	Characterizes the ratio between the fundamental signal and the highest spurious in the spectrum.
SINAD	The ratio of the RMS value of the signal amplitude to the RMS value of all other spectral components, including harmonics, but excluding DC.
ENOB	The number of bits with which the ADC behaves.
DNL	The maximum and minimum difference in the step width between actual transfer function and the perfect transfer function
INL	The maximum vertical difference between the actual and the ideal curve. It indicates the amount of deviation of the actual curve from the ideal transfer curve.

2.10.2. Setting the Frequency of the Reference Signal

The **Frequency Selection** panel allows you to compute the reference signal frequency that ADC performance tests require. This frequency is critical and affects the validity of your test results.The computed frequency varies depending on the type of test you want to do with the ADC Toolkit.

Figure 13. Frequency Selection Panel



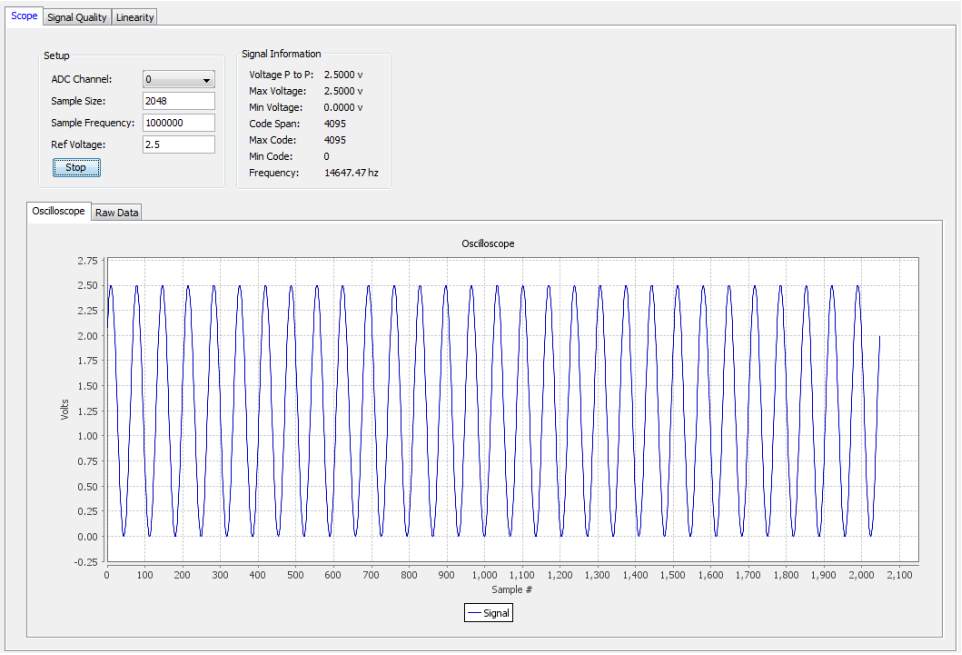
To set the frequency of the reference signal:

1. On **ADC Channel**, select the ADC channel that you plan to test.
   The tool populates the **Sample Size** and **Sample Frequency** fields.
2. In **Desired Frequency**, enter the target frequency for testing.
3. Click **Calculate**.
   - The closest frequency for valid testing near your desired frequency appears under both **Signal Quality Test** and **Linearity Test**.
   - The nearest required sine wave frequencies are different for the signal quality test and linearity test.
4. Set your signal generator to the precise frequency given by the tool based on the type of test you want to run.

2.10.3. Tuning the Signal Generator

The **Scope** panel allows you to tune the signal generator in order to achieve the best possible performance from the ADC.

Figure 14. Scope Mode Panel



To tune the signal generator:

1. On **ADC Channel**, select the ADC channel that you plan to test.
2. Enter the reference **Sample Frequency** (unless the tool can extract this value from the IP).
3. Enter the **Ref Voltage** (unless the tool can extract this value from the IP).
4. Click **Run**.
   The tool repeatedly captures a buffer worth of data and displays the data as a waveform, besides additional information under **Signal Information**.
5. Tune the signal generator to use the maximum dynamic range of the ADC without clipping.

   Note: Avoid hitting 0 or 4095 because of signal clipping.
6. Ensure that the sine wave under **Oscilloscope** shows evenly balanced top and bottom peaks. This indicates optimum value.
   - For Intel MAX 10 devices, you want to get as close to **Min Code = 0** and **Max Code = 4095** without actually hitting those values.
   - To observe coherent sampling in the test window, you must set the frequency precisely to the value needed for testing, Before moving forward, follow the suggested value for signal quality testing or linearity testing that appears next to the detected frequency.
7. From the **Raw Data** tab, export the data as a `.csv` file.

Related Information

Additional information about coherent sampling vs window sampling

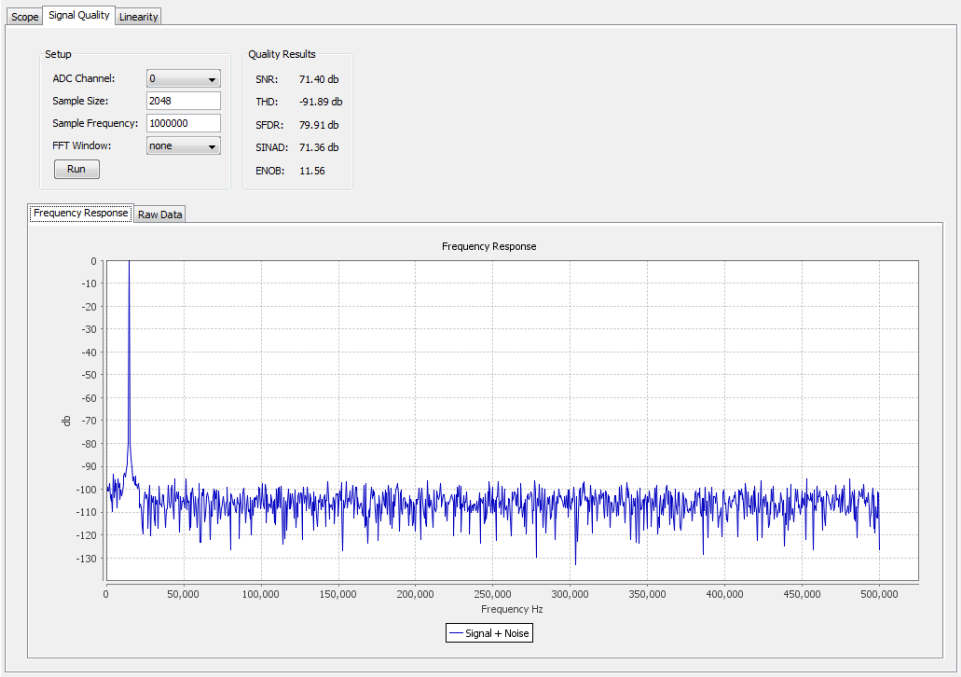
2.10.4. Running a Signal Quality Test

The available performance metrics in signal quality test mode are the following: signal to noise ratio (SNR), total harmonic distortion (THD), spurious free dynamic range (SFDR), signal to noise and distortion ratio (SINAD), effective number of bits (ENOB), and a frequency response graph.The frequency response graph shows the signal, noise floor, and any spurs or harmonics.

The signal quality parameters are measurements relative to the carrier signal and not the full scale of the ADC.

Before running a signal quality test, ensure that you have set up the frequency of the reference signal using **Scope** mode.

Figure 15. Signal Quality Panel



To run a signal quality test:

1. On **ADC Channel**, select the ADC channel that you plan to test.
2. Click **Run**.

   From the **Raw Data** tab, you can export your data as a `.csv` file.

For signal quality tests, the signal must be coherently sampled. Based on the sampling rate and number of samples to test, specific input frequencies are required for coherent sampling.The sample frequency for each channel is calculated based on the ADC sequencer configuration.

Related Information

Additional information about dynamic parameters such as SNR, THD, etc

2.10.5. Running a Linearity Test

The linearity test determines the linearity of the step sizes of each ADC code. It uses a histogram testing method which requires sinusoidal inputs which are easier to source from signal generators and DACs than other test methods.

When using **Linearity** test mode, the reference signal must meet specific requirements:

- The signal source covers the full code range of the ADC. Results improve if the time spent at code end is equivalent, by tuning the reference signal in **Scope** mode.
- If you use code ends, ensure that you are not clipping the signal. Look at the signal in **Scope** mode to see that it does not look flat at the top or bottom. A good practice is to back away from code ends and test a smaller range within the desired operating range of the ADC input signal.
- Choosing a frequency that is not an integer multiple of the sample rate and buffer size helps to ensure all code bins are filled relatively evenly to the probability density function of a sine wave. If an integer multiple is selected, some bins may be skipped entirely while others are over populated. This makes the tests results invalid. Use the frequency calculator feature to determine a good signal frequency near your desired frequency.

To run a linearity test:

1. On **ADC Channel**, select the ADC channel that you plan to test.
2. Enter the test sample size in **Burst Size**. Larger samples increase the confidence in the test results.
3. Click **Run**.
    - You can stop the test at anytime, as well as click **Run** again to continue adding to the aggregate data. To start fresh, click **Reset** after you stop a test. Anytime you change the input signal or channel, you should click **Reset** so your results are correct for a particular input.
    - There are three graphical views of the data: **Histogram** view, **DNL** view, and **INL** view.
    - From the **Raw Data** tab, you can export your data as a `.csv` file.

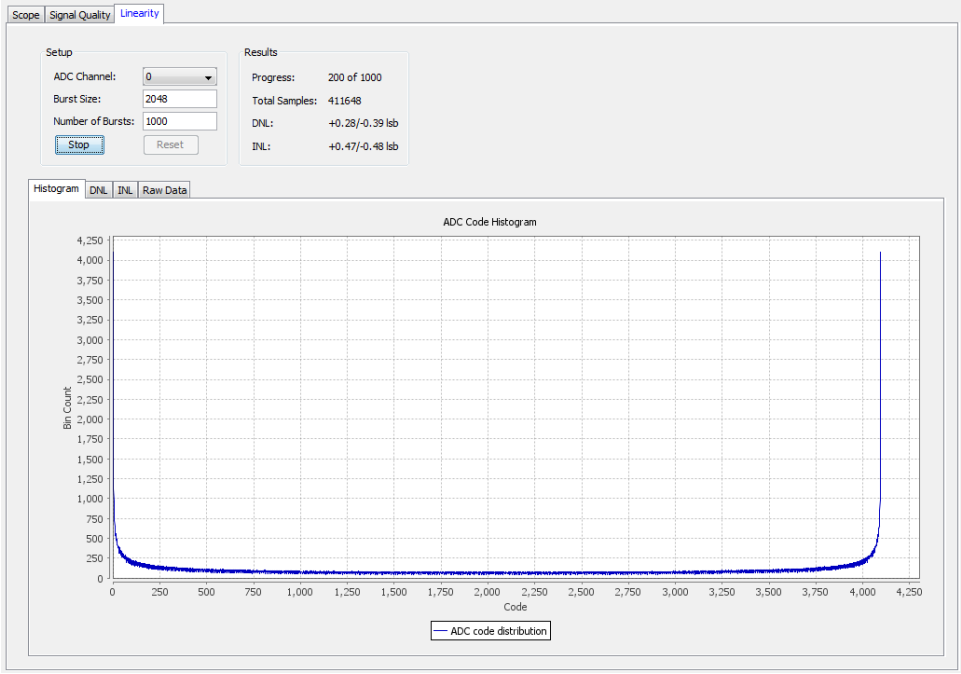
2.10.6. ADC Toolkit Data Views

Histogram View

The **Histogram** view shows how often each code appears. The graph updates every few seconds as it collects data. You can use the **Histogram** view to quickly check if your test signal is set up appropriately.

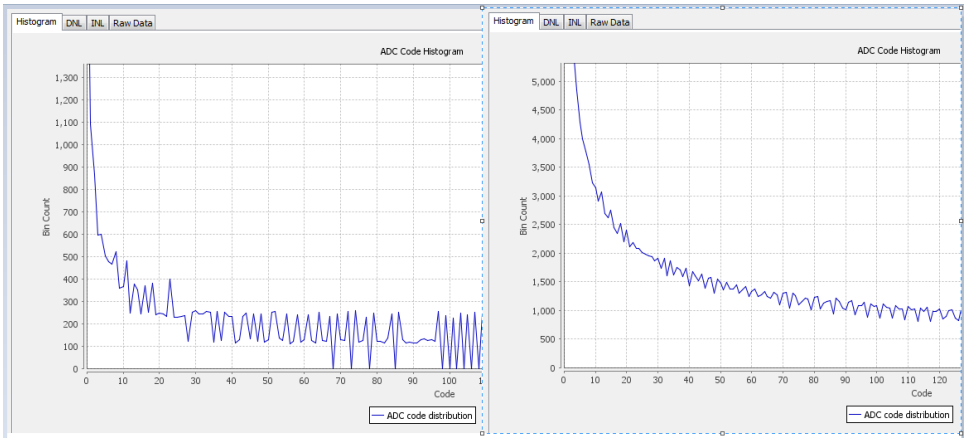
Figure 16. Example of Pure Sine Wave Histogram

The figure below shows the shape of a pure sine wave signal. Your reference signal should look similar.



If your reference signal is not a relatively smooth line, but has jagged edges with some bins having a value of 0, and adjacent bins with a much higher value, then the test signal frequency is not adequate. Use **Scope** mode to help choose a good frequency for linearity testing.

Figure 17. Examples of (Left) Poor Frequency Choice vs (Right) Good Frequency Choice

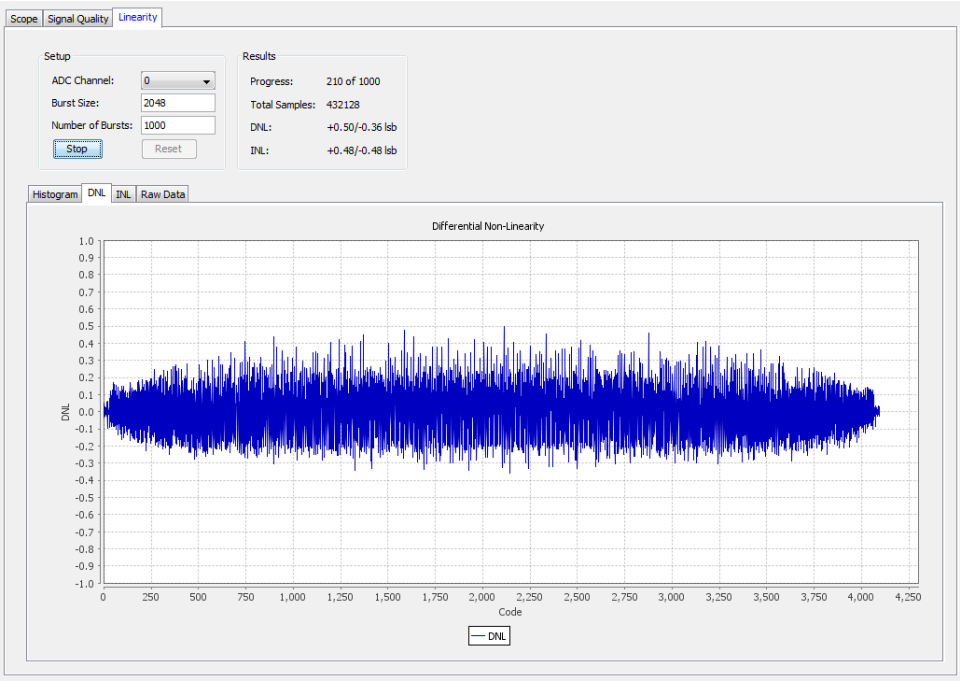


Differential Non-linearity View

Figure 18. Example of Good Differential Non-linearity

The **DNL** view shows the currently collected data. Ideally, you want your data to look like a straight line through the 0 on the x-axis. When there are not enough samples of data, the line appears rough. The line improves as more data is collected and averaged.

Each point in the graph represents how many LSB values a particular code differs from the ideal step size of 1 LSB. The **Results** box shows the highest positive and negative DNL values.

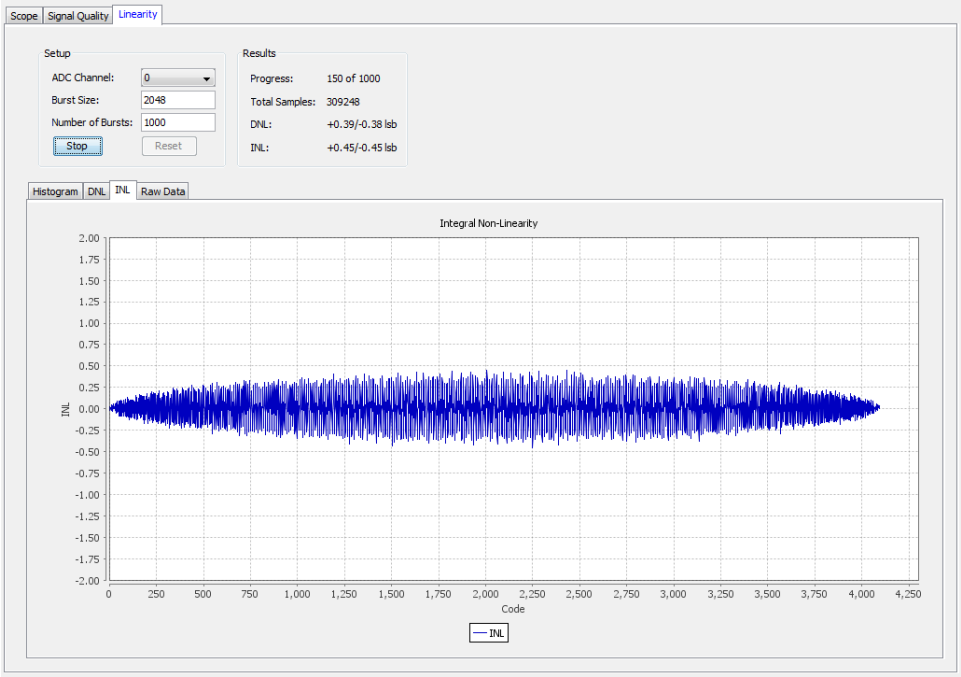


Integral Non-linearity View

Figure 19. Example of Good Integral Non-linearity

The **INL** view shows currently collected data. Ideally, with a perfect ADC and enough samples, the graph appears as a straight line through 0 on the x-axis.

Each point in the graph represents how many LSB values a particular code differs from its expected point in the voltage slope. The **Results** box shows the highest positive and negative INL values.



2.11. System Console Examples and Tutorials

Intel provides examples for performing board bring-up, creating a simple dashboard, 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 (Standard).

Related Information

On-Chip Debugging Design Examples Website
Contains the design files for the example designs that you can download.

2.11.1. 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 Altera 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

2.11.1.1. Processor Commands

Table 34. Processor Commands

Command [1]	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 85

2.12. 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.

Not all Stratix® V boards support the On-Board Intel FPGA Download Cable II. For example, the transceiver signal integrity board does not support the On-Board Intel FPGA Download Cable II.

2.13. 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

(1) 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.

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 20. Hardware in the Loop Host-Target Setup

Related Information

White Paper 01208: Hardware in the Loop from the MATLAB/Simulink Environment

2.13.1. Supported MATLAB API Commands

You can perform the work from the MATLAB environment, and read and write to masters and slaves through the System Console. The supported MATLAB API commands spare you from launching the System Console software. 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 20. 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
```

2.13.2. 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-MM components in the design (DSP Builder for Intel FPGAs can port non-Avalon-MM 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 (Standard)
7. Program the Intel FPGA
8. Interact with the Intel FPGA through the supported MATLAB API commands.

2.14. Deprecated Commands

The table lists commands that have been deprecated. These commands are currently supported, but are targeted for removal from System Console.

Note: All `dashboard_<name>` commands are deprecated and replaced with `toolkit_<name>` commands for Intel Quartus Prime software15.1, and later.

Table 35. Deprecated Commands

Command	Arguments	Function
`open_service`	*<service_type>* *<service_path>*	Opens the specified service type at the specified path. Calls to `open_service` may be replaced with calls to `claim_service` providing that the return value from `claim_service` is stored and used to access and close the service.

2.15. 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
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
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.
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.
November 2012	12.1.0	Re-organization of content.
August 2012	12.0.1	Moved Transceiver Toolkit commands to Transceiver Toolkit chapter.

continued...

Document Version	Intel Quartus Prime Version	Changes
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.

3. Debugging Transceiver Links

The Transceiver Toolkit helps you 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.

The Transceiver Toolkit allows you to:

- Control the transmitter or receiver channels to optimize transceiver settings and hardware features.
- Test bit-error rate (BER) while running multiple links at the target data rate.
- Control internal pattern generators and checkers, as well as enabling loopback modes.
- Run auto sweep tests to identify the best physical media attachment (PMA) settings for each link.
- For Stratix V devices, view the receiver horizontal and vertical eye margin during testing.
- Test multiple devices across multiple boards simultaneously.

Note: The Transceiver Toolkit runs from the System Console framework.

To launch the toolkit, click **Tools ➤ System Debugging Tools ➤ Transceiver Toolkit**. Alternatively, you can run Tcl scripts from the command-line:

```
system-console --script=<name of script>
```

For an online demonstration using the Transceiver Toolkit to run a high-speed link test with one of the design examples, refer to the Transceiver Toolkit Online Demo on the Altera website.

Related Information

- On-Chip Debugging Design Examples
- Transceiver Toolkit Online Demo
- Transceiver Toolkit for Intel Arria® 10 Devices (OTCVRKITA10)
  26 Minutes Online Course
- Transceiver Toolkit for 28-nm Devices (OTCVR1100)
  39 Minutes Online Course

3.1. Channel Manager

The Channel Manager is the graphical component of the Transceiver Toolkit. The Channel Manager allows you to configure and control transceiver channels and links, and adjust programmable analog settings to improve the signal integrity of the link. The Channel Manager is in the Workspace area of the System Console.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

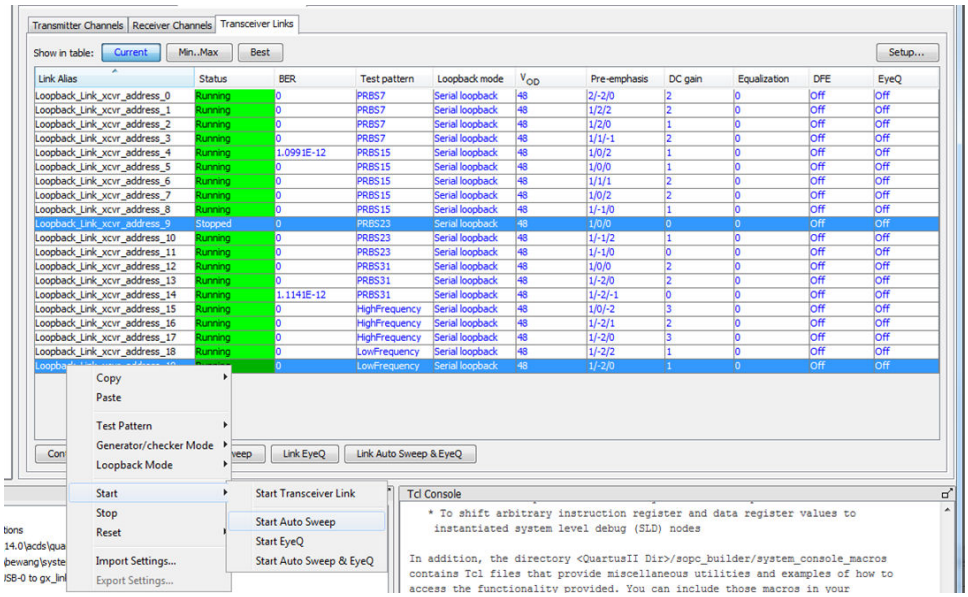
ISO 9001:2015 Registered

The Channel Manager consists of three tabs that display components in a spreadsheet format:

- **Transmitter Channels**
- **Receiver Channels**
- **Transceiver Links**

The columns on each tab depend on the parameters of each device.

Figure 21. Example: Transceiver Links Tab of the Channel Manager



Channel Manager Functions

The Channel Manager simplifies actions such as:

- Copying and pasting settings—Copy, paste, import, and export PMA settings to and from channels.
- Importing and exporting settings— To export PMA settings to a text file, select a row in the Channel Manager. To apply the PMA settings from a text file, select one or more rows in the Channel Manager. The PMA settings in the text file apply to a single channel. When you import the PMA settings from a text file, you are duplicating one set of PMA settings for all the channels you select.
- Starting and stopping tests—The Channel Manager allows you to start and stop tests by right-clicking the channels. You can select two or more rows in the Channel Manager to start or stop test for multiple channels.

Related Information

- System Explorer Pane on page 22
- System Console GUI on page 21
- User Interface Settings Reference on page 119

3.1.1. Channel Display Modes

The three channel display modes are:

- **Current** (default)—shows the current values from the device. Blue text indicates that the settings are live.
- **Min/Max**—shows the minimum and maximum values to be used in the auto sweep.
- **Best**—shows the best tested values from the last completed auto sweep run.

Note: The **Transmitter Channels** tab only shows the **Current** display mode. The Transceiver Toolkit requires a receiver channel to perform auto sweep tests.

3.2. Transceiver Debugging Flow Walkthrough

These steps describe the high-level process of debugging transceivers with the Transceiver Toolkit.

1. Modify the design to enable transceiver debug.
2. Load the modified design to the FPGA.
3. Load the design to the Transceiver Toolkit.
4. Link hardware resources.
5. Verify hardware connections.
6. Identify transceiver channels.
7. Run link tests or control PMA analog settings.

3.3. Modifying the Design to Enable Transceiver Debug

The configuration of the debugging system varies by device family.

3.3.1. Adapting an Intel FPGA Design Example

Design examples allow you to quickly test the functionality of the receiver and transmitter channels in your design. You can modify and customize the design examples to match your intended transceiver design and signal integrity development board.

1. Download a design example from the On-Chip Debugging Design Examples page of the Intel FPGA website.
2. Open the Intel Quartus Prime and click **Project ➤ Restore Archived Project** to restore the design example project archive.
3. Compare the development board and device specified in the `readme.txt` file with your board and device:

Option	Description
Same development board and same device	Directly program the device with the programming file included in the example.
Same board, different device	Choose the appropriate device and recompile the design.

Option	Description
Different board	Edit the necessary pin assignments and recompile the design example.

4. To recompile the design, you must make your modifications to the system configuration in Platform Designer (Standard), regenerate in Platform Designer (Standard), and recompile the design in the Intel Quartus Prime software to generate a new programming file.

Once you recompile your design, you can:

- Change the transceiver settings in the design examples and observe the effects on transceiver link performance
- Isolate and verify the high-speed serial links without debugging other logic in your design.

Refer to the `readme.txt` of each design example for more information.

3.3.1.1. Modifying Stratix V Design Examples

You can adapt Intel FPGA design examples to experiment with configurations that match your own design. For example, you can change data rate, number of lanes, PCS-PMA width, FPGA-fabric interface width, or input reference clock frequency. To modify the design examples, change the IP core parameters and regenerate the system in Platform Designer (Standard). Next, update the top-level design file, and re-assign device I/O pins as necessary.

To modify a Stratix V design example PHY block to match your design, follow these steps:

1. Determine the number of channels your design requires.
2. Open the <*project name*>`.qpf` for the design example in the Intel Quartus Prime software.
3. Click **Tools ➤ Platform Designer (Standard)**.
4. On the **System Contents** tab, right-click the PHY block and click **Edit**. Specify options for the PHY block to match your design requirement for number of lanes, data rate, PCS-PMA width, FPGA-fabric interface width, and input reference clock frequency.
5. Specify a multiple of the FPGA-fabric interface data width for **Avalon Data Symbol Size**. The available values are **8** or **10**. Click **Finish**.
6. Delete any timing adapter from the design. The timing adapters are not required.
7. From the IP Catalog, add one **Data Pattern Generator** and **Data Pattern Checker** for each transmitter and receiver lane.
8. Right-click **Data Pattern Generator** and click **Edit**. Specify a value for **ST_DATA_W** that matches the FPGA-fabric interface width.
9. Right-click **Data Pattern Checker** and click **Edit**. Specify a value for **ST_DATA_W** that matches the FPGA-fabric interface width.
10. From the IP Catalog, add a **Transceiver Reconfiguration Controller**.
11. Right-click **Transceiver Reconfiguration Controller** and click **Edit**. Specify 2* number of lanes for the number of reconfigurations interfaces. Click **finish**.

12. Create connections for the data pattern generator and data pattern checker components. Right-click the net name in the **System Contents** tab and specify the following connections.

From		To	
Block Name	**Net Name**	**Block Name**	**Net Name**
clk_100	clk	data_pattern_generator	csr_clk
clk_100	clk_reset	data_pattern_generator	csr_clk_reset
master_0	master	data_pattern_generator	csr_slave
xcvr_*_phy_0	tx_clk_out0	data_pattern_generator	pattern_out_clk
xcvr_*_phy_0	tx_parallel_data0	data_pattern_generator	pattern_out
clk_100	clk	data_pattern_checker	csr_clk
clk_100	clk_reset	data_pattern_checker	csr_clk_reset
master_0	master	data_pattern_checker	csr_slave
xcvr_*_phy_0	rx_clk_out0	data_pattern_checker	pattern_in_clk
xcvr_*_phy_0	rx_parallel_data0	data_pattern_checker	pattern_in

13. Click **System ➤ Assign Base Addresses**.
14. Connect the reset port of timing adapters to `clk_reset` of `clk_100`.
15. To implement the changes to the system, click **Generate ➤ Generate HDL**.
16. If you modify the number of lanes in the PHY, you must update the top-level file accordingly. The following example shows Verilog HDL code for a two-channel design that declares input and output ports in the top-level design. The example design includes the low latency PHY IP core. If you modify the PHY parameters, you must modify the top-level design with the correct port names. Platform Designer (Standard) displays an example of the PHY, click **Generate ➤ HDL Example**.

```
module low_latency_10g_1ch DUT (
                        input  wire GXB_RXL11,
                        input  wire GXB_RXL12,
                        output wire GXB_TXL11,
                        output wire GXB_TX12
                        );
                        .....
                        low_latency_10g_1ch DUT (
                        .....
                        .xcvr_low_latency_phy_0_tx_serial_data_export
({GXB_TXL11,     GXB_TXL12}),
                        .xcvr_low_latency_phy_0_rx_serial_data_export
({GXB_RXL11,         GXB_TXL12}),
                        .....
                        );
```

17. From the Intel Quartus Prime software, click **Assignments ➤ Pin Planner** and update pin assignments to match your board.
18. Edit the design's Synopsys Design Constraints (`.sdc`) to reflect the reference clock change. Ignore the reset warning messages.
19. Click **Start ➤ Start Compilation** to recompile the design.

3.3.1.1.1. Generating reconfig_clk from an Internal PLL

You can use an internal PLL to generate the `reconfig_clk`, by changing the Platform Designer (Standard) connections to delay offset cancellation until the generated clock is stable.

- If there is no free running clock within the required frequency range of the reconfiguration clock, add a PLL to the top-level of the design example. The frequency range varies depending on the device family. Refer to the device family data sheet for your device.
- When using an internal PLL, hold off offset cancellation until the generated clock is stable. You do this by connecting the `pll_locked` signal of the internal PLL to the `.clk_clk_in_reset_n` port of the Platform Designer (Standard) system, instead of the `system_reset` signal.
- Implement the filter logic, inverter, and synchronization to the `reconfig_clk` outside of the Platform Designer (Standard) system with your own logic.

You can find the support solution in the Intel FPGA Knowledge Base. The solution applies to only Arria® V, Cyclone® V, Stratix IV GX/GT, and Stratix V devices.

3.3.2. Stratix V Debug System Configuration

For Stratix V designs, the Transceiver Toolkit configuration requires instantiation of the JTAG to Avalon Bridge and Reconfiguration Controller IP cores. Click **Tools ➤ IP Catalog** to parameterize, generate, and instantiate the following debugging components for Stratix V designs.

Table 36. Stratix V / 28nm Transceiver Toolkit IP Core Configuration

Component	Debugging Functions	Parameterization Notes	Connect To
Transceiver Native PHY	Supports all debugging functions	• If **Enable 10G PCS** is enabled, **10G PCS protocol mode** must be set to **basic** on the **10G PCS** tab.	• Avalon-ST Data Pattern Checker • Avalon-ST Data Pattern Generator • JTAG to Avalon Master Bridge • Reconfiguration controller
Custom PHY	Test all possible transceiver parallel data widths	• Set lanes, group size, serialization factor, data rate, and input clock frequency to match your application. • Turn on **Avalon data interfaces**. • Disable **8B/10B**. • Set **Word alignment mode** to **manual**. • Disable **rate match FIFO**. • Disable **byte ordering block**.	• Avalon-ST Data Pattern Checker • Avalon-ST Data Pattern Generator • JTAG to Avalon Master Bridge • Reconfiguration controller
Low Latency PHY	Test at more than 8.5 Gbps in GT devices or use of PMA direct mode (such as when using six channels in one quad)	• Set **Phase compensation FIFO mode** to **EMBEDDED** above certain data rates and set to **NONE** for PMA direct mode. • Turn on **Avalon data interfaces**. • Set serial loopback mode to enable serial loopback controls in the toolkit.	• Avalon-ST Data Pattern Checker • Avalon-ST Data Pattern Generator • JTAG to Avalon Master Bridge • Reconfiguration controller

continued...

Component	Debugging Functions	Parameterization Notes	Connect To
Intel-Avalon Data Pattern Generator	Generates standard data test patterns at Avalon-ST source ports	• Select **PRBS7**, **PRBS15**, **PRBS23**, **PRBS31**, **high frequency**, or **low frequency** patterns. • Turn on **Enable Bypass interface** for connection to design logic.	• PHY input port • JTAG to Avalon Master Bridge • Your design logic
Intel-Avalon Data Pattern Checker	Validates incoming data stream against test patterns accepted on Avalon streaming sink ports	Specify a value for ST_DATA_W that matches the FPGA-fabric interface width.	• PHY output port • JTAG to Avalon Master Bridge
Reconfiguration Controller	Supports PMA control and other transceiver settings	• Connect the reconfiguration controller to • Connect `reconfig_from_xcvr` to `reconfig_to_xcvr`. • Enable Analog controls. • Turn on **Enable Eye Viewerblock** to enable signal eye analysis (Stratix V only) • Turn on **Enable Bit Error Rate Block** for BER testing • Turn on **Enable decision feedback equalizer (DFE) block** for link optimization • Enable DFE block	• PHY input port • JTAG to Avalon Master Bridge
JTAG to Avalon Master Bridge	Accepts encoded streams of bytes with transaction data and initiates Avalon-MM transactions	N/A	• PHY input port • Avalon-ST Data Pattern Checker • Avalon-ST Data Pattern Generator • Reconfiguration Controller

3.3.2.1. Bit Error Rate Test Configuration (Stratix V)

Use the following configuration to perform bit rate error testing in Stratix V designs.

Figure 22. Bit Error Rate Test Configuration (Stratix V)

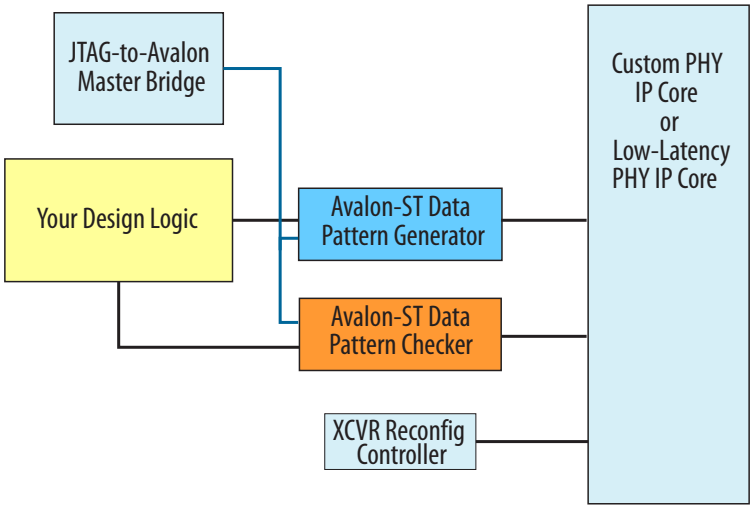


Table 37. System Connections: Bit Error Rate Tests

From	To
Your Design Logic	Data Pattern Generator bypass port
Data Pattern Generator	PHY input port
JTAG to Avalon Master Bridge	Intel FPGA Avalon Data Pattern Generator
JTAG to Avalon Master Bridge	Intel FPGA Avalon Data Pattern Checker
JTAG to Avalon Master Bridge	PHY input port
Data Pattern Checker	PHY output port
Transceiver Reconfiguration Controller	PHY input port

Related Information

Running BER Tests on page 111

3.3.2.2. PRBS Signal Eye Test Configuration (Stratix V)

Use the following configuration to perform PRBS signal eye testing in Stratix V designs.

Figure 23. PRBS Signal Eye Test Configuration (Stratix V)

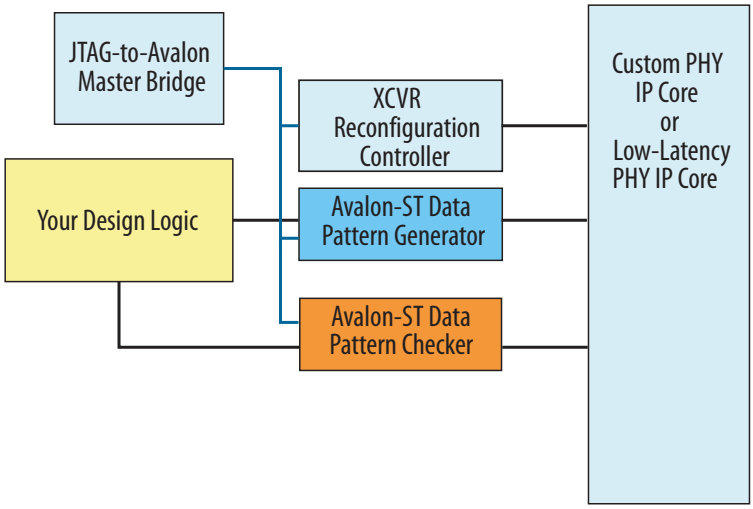


Table 38. System Connections: PRBS Signal Eye Tests (Stratix V)

From	To
Your Design Logic	Data Pattern Generator bypass port
Data Pattern Generator	PHY input port
JTAG to Avalon Master Bridge	Intel Avalon Data Pattern Generator
JTAG to Avalon Master Bridge	Intel Avalon Data Pattern Checker
Data Pattern Checker	PHY output port

continued...

From	To
JTAG to Avalon Master Bridge	Transceiver Reconfiguration Controller
JTAG to Avalon Master Bridge	PHY input port
Transceiver Reconfiguration Controller	PHY input port

Related Information

Running PRBS Signal Eye Tests (Stratix V only) on page 112

3.3.2.2.1. Enabling Serial Bit Comparator Mode (Stratix V)

Serial bit comparator mode allows you to run Eye Viewer diagnostic features with any PRBS patterns or user-design data, without disrupting the data path. For Stratix V devices, you must enable **Serial bit comparator** mode.

To enable this mode for Stratix V devices, you must enable the following debugging component options when configuring the debugging system:

Table 39. Component Settings for Serial Bit Comparator Mode

Debugging Component	Setting for Serial Bit Mode(2)
Transceiver Reconfiguration Controller	Turn on **Enable Eye Viewer block** and **Enable Bit Error Rate Block**
Data Pattern Generator(3)	Turn on **Enable Bypass interface**

Serial bit comparator mode is less accurate than **Data pattern checker** mode for single bit error checking. Do not use **Serial bit comparator** mode if you require an exact error rate. Use the **Serial bit comparator** mode for checking a large window of error. The toolkit does not read the bit error counter in real-time because it reads through the memory-mapped interface. Serial bit comparator mode has the following hardware limitations for Stratix V devices:

- Toolkit uses serial bit checker only on a single channel per reconfiguration controller at a time.
- When the serial bit checker is running on channel *n*, you can change only the V_{OD}, pre-emphasis, DC gain, and Eye Viewer settings on that channel. Changing or enabling DFE or CTLE can cause corruption of the serial bit checker results.
- When the serial bit checker is running on a channel, you cannot change settings on any other channel on the same reconfiguration controller.
- When the serial bit checker is running on a channel, you cannot open any other channel in the Transceiver Toolkit.
- When the serial bit checker is running on a channel, you cannot copy PMA settings from any channel on the same reconfiguration controller.

3.3.2.3. Custom Traffic Signal Eye Test Configuration (Stratix V)

Use the following configuration to perform custom traffic signal eye testing in Stratix V designs.

(2) Settings in Table 39 on page 100 are supported in Stratix V devices only.

(3) Limited support for Data Pattern Generator or data pattern in Serial Bit Mode.

Figure 24. System Configuration: Custom Traffic Signal Eye Tests (Stratix V)

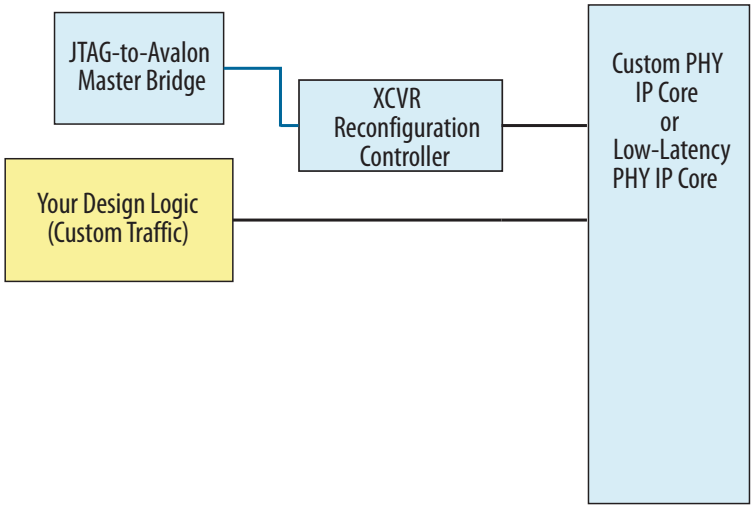


Table 40. System Connections: Custom Traffic Signal Eye Tests (Stratix V)

From	To
Your design logic with custom traffic	PHY input port
JTAG to Avalon Master Bridge	Transceiver Reconfiguration Controller
JTAG to Avalon Master Bridge	PHY input port
Transceiver Reconfiguration Controller	PHY input port

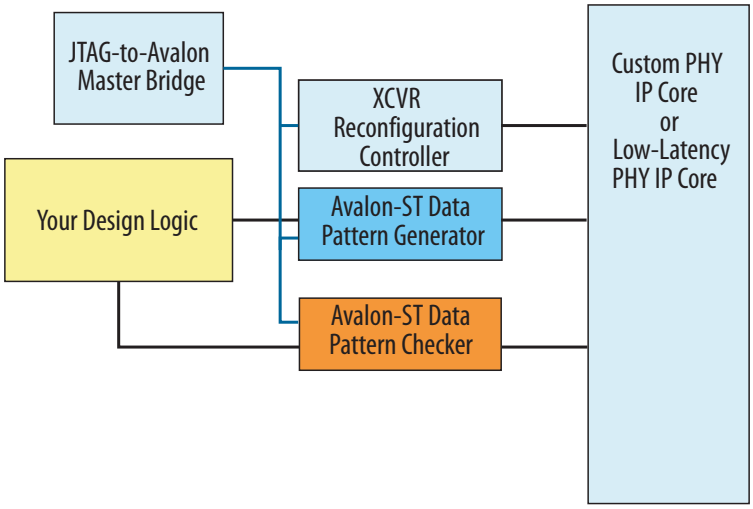
Related Information

Running Custom Traffic Tests (Stratix V only) on page 113

3.3.2.4. Link Optimization Test Configuration (Stratix V)

Use the following configuration for link optimization tests in Stratix V devices.

Figure 25. System Configuration: Link Optimization Tests (Stratix V)



From	To
Your Design Logic	Data Pattern Generator bypass port
Data Pattern Generator	PHY input port
JTAG to Avalon Master Bridge	Altera Avalon Data Pattern Generator
JTAG to Avalon Master Bridge	Altera Avalon Data Pattern Checker
Data Pattern Checker	PHY output port
JTAG to Avalon Master Bridge	Transceiver Reconfiguration Controller
JTAG to Avalon Master Bridge	PHY input port
Transceiver Reconfiguration Controller	PHY input port

Related Information

Running the Auto Sweep Test on page 114

3.3.2.5. PMA Analog Setting Control Configuration (Stratix V)

Use the following configuration to control PMA Analog settings in Stratix V designs.

Figure 26. System Configuration: PMA Analog Setting Control (Stratix V)

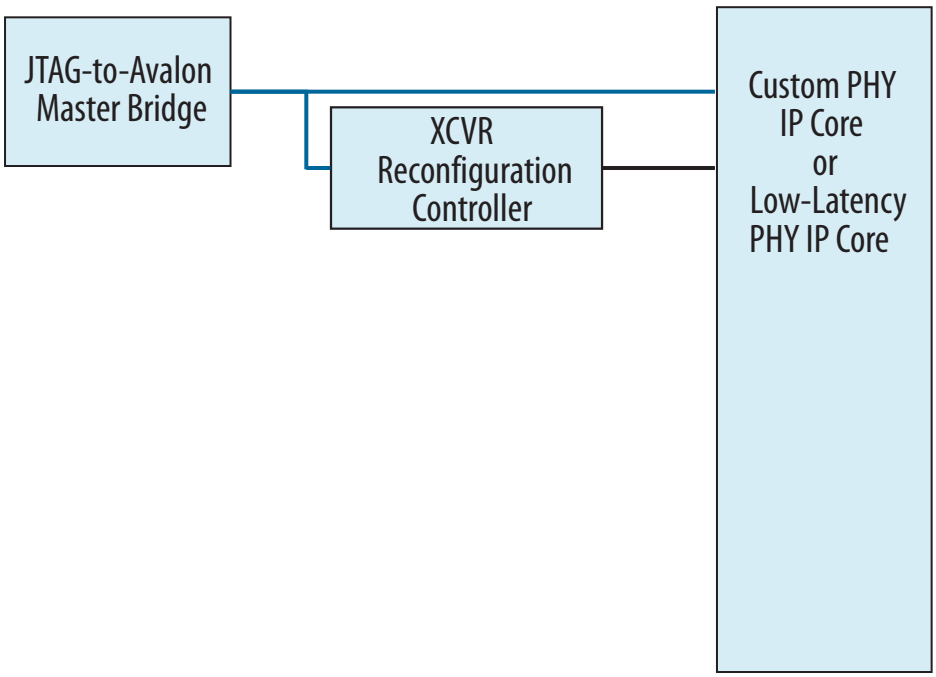


Table 41. System Connections: PMA Analog Setting Control (Stratix V)

From	To
JTAG to Avalon Master Bridge	Transceiver Reconfiguration Controller
JTAG to Avalon Master Bridge	PHY input port
Transceiver Reconfiguration Controller	PHY input port

Related Information

Controlling PMA Analog Settings on page 115

3.3.3. Instantiating and Parameterizing Intel Arria 10 Debug IP cores

To debug Intel Arria 10 designs with the Transceiver Toolkit, you must enable debugging settings in Transceiver Intel FPGA IP cores. You can either activate these settings when you first instantiate these components, or modify your instance after preliminary compilation.

The IP cores that you modify are:

- Transceiver Native PHY
- Transceiver ATX PLL
- CMU PLL
- fPLL

The parameters that you enable in the debug IP cores are:

Table 42. IP Cores and Debug Settings

For more information about these parameters, refer to *Debug Settings for Transceiver IP Cores*.

IP Core	Enable dynamic reconfiguration	Enable Altera Debug Master Endpoint	Enable capability registers	Enable control and status registers	Enable PRBS Soft accumulators
Transceiver Native PHY	Yes	Yes	Yes	Yes	Yes
Transceiver ATX PLL	Yes	Yes			
CMU PLL	Yes	Yes			
fPLL	Yes	Yes			

For each transceiver IP core:

1. In the **IP Components** tab of the Project Navigator, right-click the IP instance, and click **Edit in Parameter Editor**.
2. Turn on debug settings as they appear in the *IP Cores and Debug Settings* table above.

Figure 27. Intel Arria 10 Transceiver Native PHY IP Core in the Parameter Editor

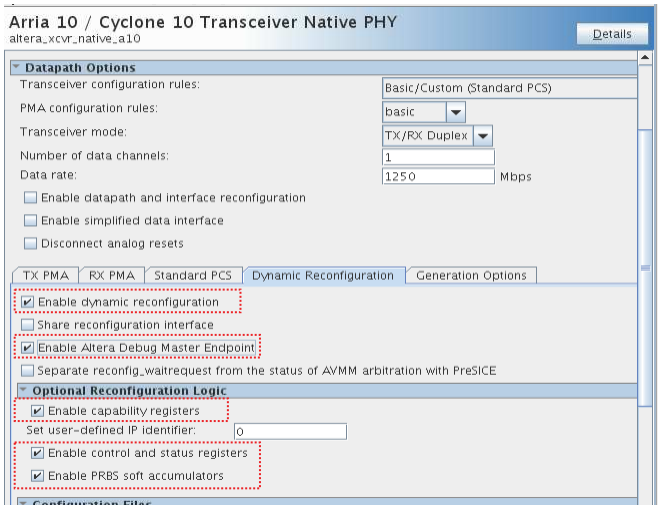
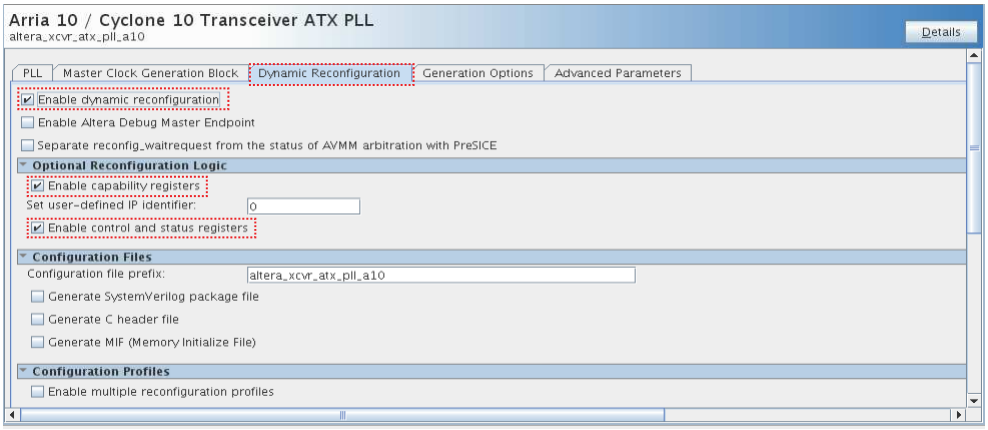


Figure 28. Intel Arria 10 Transceiver ATX PLL Core in the Parameter Editor



3. Click **Generate HDL**.

After enabling parameters for all IPs in the design, recompile your project.

3.3.3.1. Debug Settings for Transceiver IP Cores

The table describes the settings that you turn on when preparing your transceiver for debug:

Table 43. Intel FPGA IP Settings for Transceiver Debug

Setting	Description
Enable Dynamic Reconfiguration	Allows you to change the behavior of the transceiver channels and PLLs without powering down the device
Enable Altera Debug Master Endpoint	Allows you to access the transceiver and PLL registers through System Console. When you recompile your design, Intel Quartus Prime software inserts the ADME, debug fabric, and embedded logic during synthesis.
Enable capability registers	Capability registers provide high level information about the configuration of the transceiver channel
Enable control and status registers	Enables soft registers to read status signals and write control signals on the PHY interface through the embedded debug.
Enable PRBS Soft Accumulators	Enables soft logic for performing PRBS bit and error accumulation when you use the hard PRBS generator and checker.

For more information about dynamic reconfiguration parameters on Intel Arria 10 devices, refer to the *Intel Arria 10 Transceiver PHY User Guide*.

Related Information

Dynamic Reconfiguration Parameters

3.4. Programming the Design into an Intel FPGA

After you include debug components in the design, compile, and generate programming files, you can program the design in the Intel FPGA.

Related Information

nocp-doc-link/683528

3.5. Loading the Design in the Transceiver Toolkit

If the FPGA is already programmed with the project when loading, the Transceiver Toolkit automatically links the design to the target hardware in the toolkit. The toolkit automatically discovers links between transmitter and receiver of the same channel.

Before loading the device, ensure that you connect the hardware. The device and JTAG connections appear in the **Device** and **Connections** folders of the **System Explorer** pane.

To load the design into the Transceiver Toolkit:

1. In the System Console, click **File ➤ Load Design**.
2. Select the `.sof` programming file for the transceiver design.

After loading the project, the **designs** and **design instances** folders in the **System Explorer** pane display information about the design, such as the design name and the blocks in the design that can communicate to the System Console.

Related Information

System Explorer Pane on page 22

3.6. Linking Hardware Resources

Linking the hardware resources maps the project you load to the target FPGA. When you load multiple design projects for multiple FPGAs, linking indicates which of the projects is in each of the FPGAs. The toolkit automatically discovers hardware and designs that you connect. You can also manually link a design to connected hardware resources in the **System Explorer**.

If you are using more than one Intel FPGA board, you can set up a test with multiple devices linked to the same design. This setup is useful if you want to perform a link test between a transmitter and receiver on two separate devices. You can also load multiple Intel Quartus Prime projects and link between different systems. You can perform tests on separate and unrelated systems in a single Intel Quartus Prime instance.

Figure 29. One Channel Loopback Mode for Stratix V (28nm)

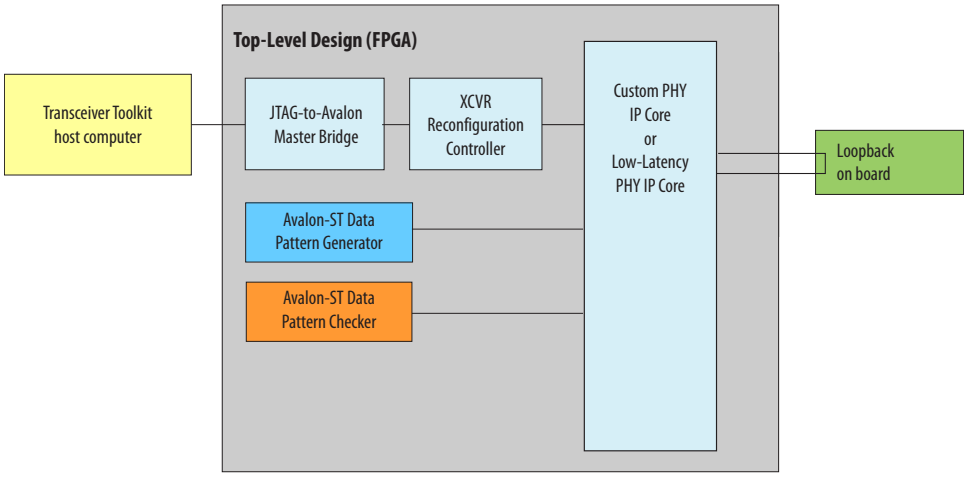


Figure 30. One Channel Loopback Mode for Intel Arria 10devices

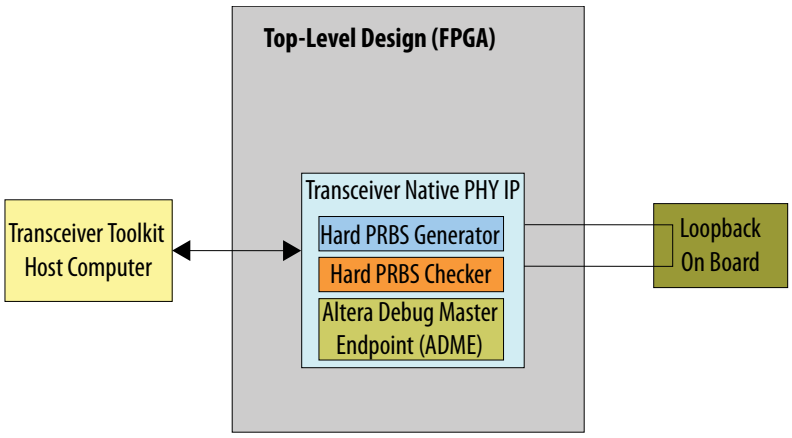


Figure 31. Four Channel Loopback Mode for Stratix V / 28nm

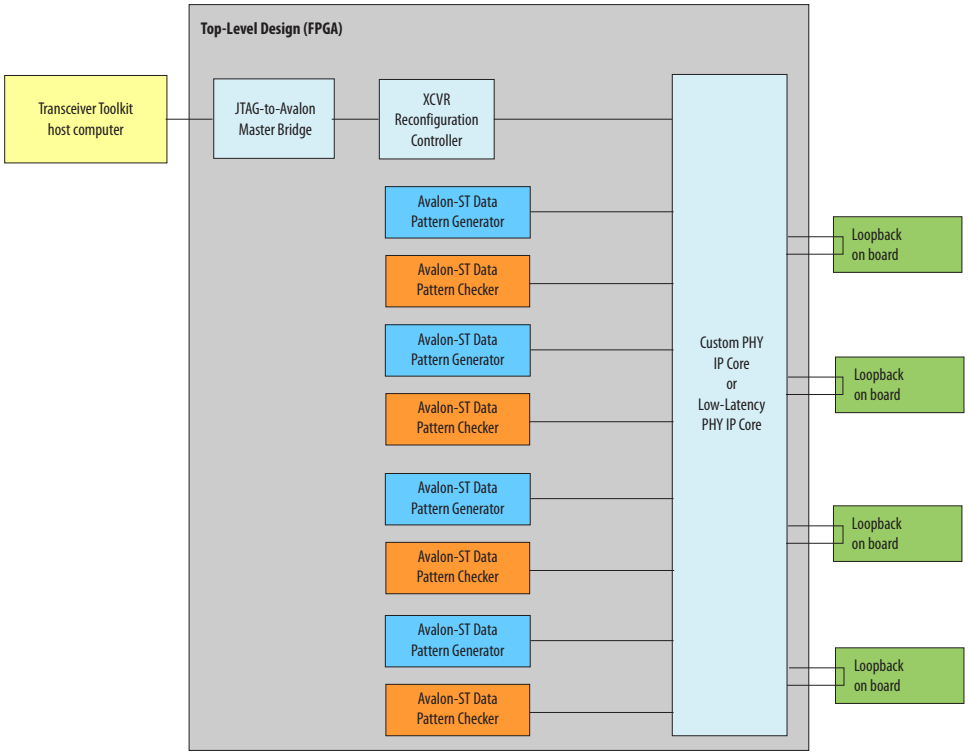
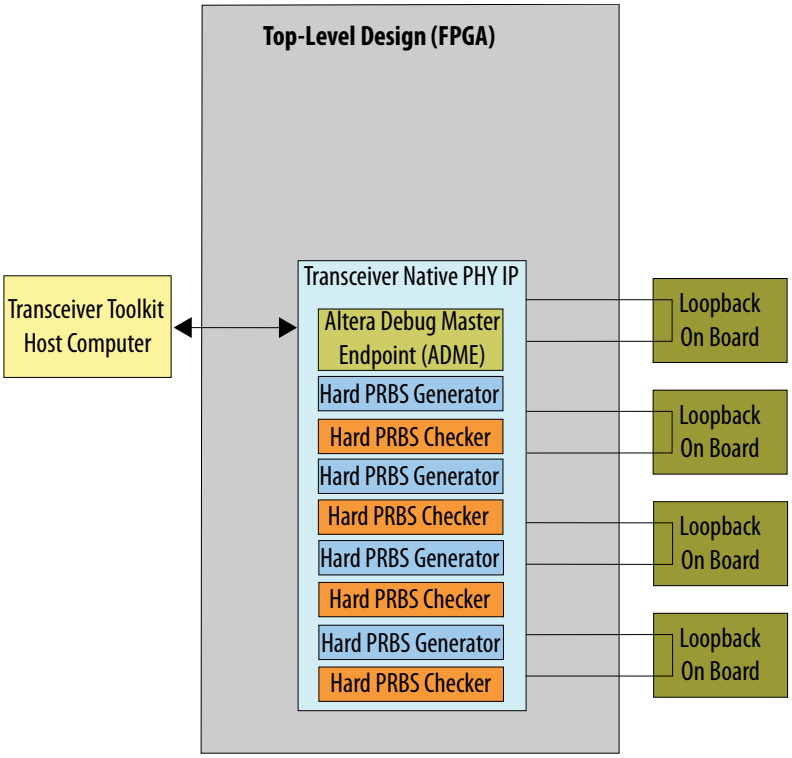


Figure 32. Four Channel Loopback Mode for Intel Arria 10devices



3.6.1. Linking One Design to One Device

To link one design to one device by one Intel FPGA Download Cable:

1. Load the design for your Intel Quartus Prime project.
2. If the design is not auto-linked, link each device to an appropriate design.
3. Create the link between channels on the device to test.

3.6.2. Linking Two Designs to Two Devices

To link two designs to two separate devices on the same board, connected by one Intel FPGA Download Cable download cable:

1. Load the design for all the Intel Quartus Prime project files you need.
2. If the design does not auto-link, link each device to an appropriate design
3. Open the project for the second device.
4. Link the second device on the JTAG chain to the second design (unless the design auto-links).
5. Create a link between the channels on the devices you want to test.

3.6.3. Linking One Design on Two Devices

To link the same design on two separate devices, follow these steps:

1. In the Transceiver Toolkit, open the `.sof` you are using on both devices.
2. Link the first device to this design instance.
3. Link the second device to the design.
4. Create a link between the channels on the devices you want to test.

3.6.4. Linking Designs and Devices on Separate Boards

To link two designs to two separate devices on separate boards that connect to separate Intel FPGA Download Cables:

1. Load the design for all the Intel Quartus Prime project files you need.
2. If the design does not auto-link, link each device to an appropriate design.
3. Create the link between channels on the device to test.
4. Link the device connected to the second Intel FPGA Download Cable to the second design.
5. Create a link between the channels on the devices you want to test.

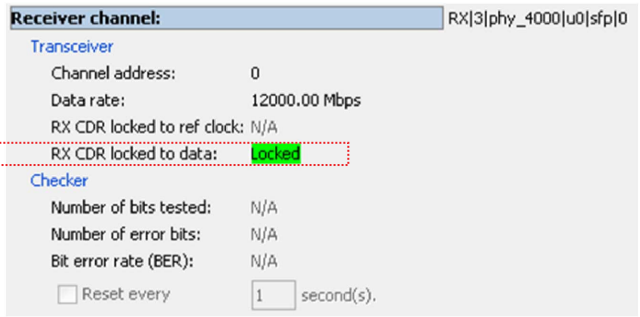
3.6.5. Verifying Hardware Connections

After creating links, verify that the channels connect correctly and loop back properly on the hardware.This precaution saves time in the workflow.

Use the toolkit to send data patterns and receive them correctly:

1. In the **Receiver** tab, verify that **RX CDR locked to Data** is set to **Locked**.

Figure 33. RX CDR Locked to Data



2. Start the generator on the Transmitter Channel.
3. Start the checker on the Receiver Channel.
4. Verify you have Lock to Data, and the Bit Error Rate between the two is very low or zero.

After you verify communication between transmitter and receiver, you can create a link between the two transceivers and perform Auto Sweep and Eye Viewer[4] tests with the pair.

[4] Eye Viewer available only for Stratix V devices.

3.7. Identifying Transceiver Channels

Verify whether the Transceiver Toolkit detects the channels correctly. If a receiver and transmitter share a transceiver channel, the toolkit identifies the channel.

The Transceiver Toolkit identifies and displays transmitter and receiver channels on the **Transmitter Channels** and **Receiver Channels** tabs of the Channel Manager. You can also manually identify the transmitter and receiver in a transceiver channel, and then create a link between the two for testing.

3.7.1. Controlling Transceiver Channels

To adjust or monitor transmitter or receiver settings while the channels are running:

- In the **Transmitter Channels** tab, click **Control Transmitter Channel**
- In the **Receiver Channels** tab, click **Control Receiver Channel**.
- In the **Transceiver Links** tab, click **Control Receiver Channel**.

For example, you can transmit a data pattern across the transceiver link, and then report the signal quality of the data you receive.

3.8. Creating Transceiver Links

Creating a link designates which Transmitter and Receiver channels connect physically. The toolkit automatically creates links when a receiver and transmitter share a transceiver channel. You can also manually create and delete links between transmitter and receiver channels.

To create a transceiver link:

1. In the Channel Manager, click **Setup**.
2. Select the generator and checker you want to control.
3. Select the transmitter and receiver pair you want to control.
4. Click **Create Transceiver Link**.
5. Click **Close**.

The Transceiver Toolkit generates an automatic name for the link, but you can use a shorter, more meaningful name by typing in the **Link Alias** cell.

3.9. Running Link Tests

Once you identify the transceiver channels for debugging, you can run link tests. Use the **Transceiver Links** tab to control link tests.

When you run link tests, channel color highlights indicate the test status:

Table 44. Channel Color Highlights

Color	Transmitter Channel	Receiver Channel
Red	Channel is closed or generator clock is not running.	Channel is closed or checker clock is not running.
Green	Generator is sending a pattern.	Checker is checking and data pattern is locked.
Neutral (same color as background)	Channel is open, generator clock is running, and generator is not sending a pattern.	Channel is open, checker clock is running, and checker is not checking.
Yellow	N/A	Checker is checking and data pattern is not locked.

3.9.1. Running BER Tests

BER tests help you assess signal integrity. Follow these steps to run BER tests across a transceiver link:

1. In the Channel Manager, click **Control Transceiver Link**.
2. Specify a PRBS **Test pattern**
3. If your device supports setting a **Checker mode**, set to **Data pattern checker**.
4. Try different values of **Reconfiguration**, **Generator**, or **Checker** settings, if available.
5. Click **Start** to run the pattern with your settings.
6. If your device supports error injection, you can click **Inject Error** to inject error bits.
7. You can also **Reset** the counter, or **Stop** the test.

Note: Intel Arria 10 devices do not support **Inject Error** if you use the hard PRBS Pattern Generator and Checker in the system configuration.

Related Information

- Bit Error Rate Test Configuration (Stratix V) on page 98
- User Interface Settings Reference on page 119

3.9.2. Signal Eye Margin Testing (Stratix V only)

Stratix V includes Eye Viewer circuitry, that allows visualization of the horizontal and vertical eye margin at the receiver. For supported devices, use signal eye tests to tune the PMA settings of your transceiver. This results in the best eye margin and BER at high data rates. The toolkit disables signal eye testing for unsupported devices.

The Eye Viewer graph can display a bathtub curve, eye diagram representing eye margin, or heat map display. The run list displays the statistics of each Eye Viewer test. When PMA settings are suitable, the bathtub curve is wide, with sharp slopes near the edges. The curve is up to 30 units wide. If the bathtub is narrow, then the signal quality is poor. The wider the bathtub curve, the wider the eye. The smaller the bathtub curve, the smaller the eye. The eye contour shows the estimated horizontal and vertical eye opening at the receiver.

You can right-click any of the test runs in the list, and then click **Apply Settings to Device** to quickly apply that PMA setting to your device. You can also click **Export**, **Import**, or **Create Report**.

Figure 34. Eye Viewer Settings and Status Showing Results of Two Test Runs

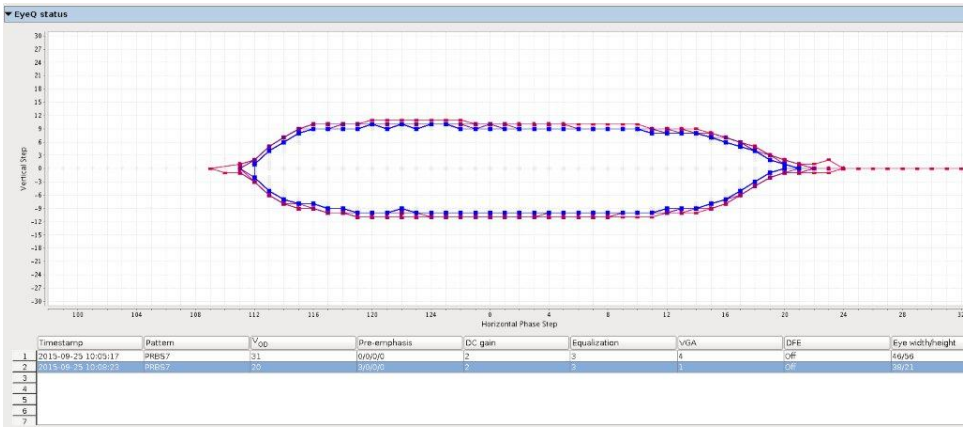
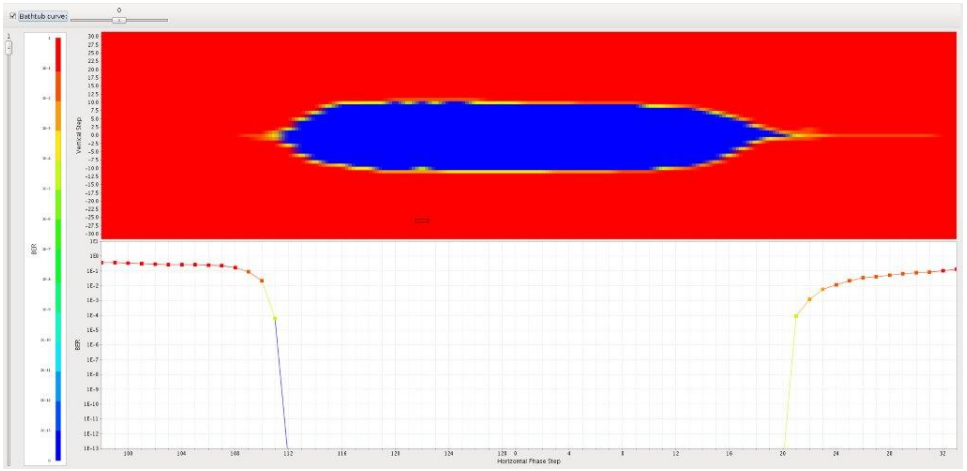


Figure 35. Heat Map Display and Bathtub Curve Through Eye



3.9.2.1. Running PRBS Signal Eye Tests (Stratix V only)

Run PRBS signal eye tests to visualize the estimated horizontal and vertical eye opening at the receiver. After programming the FPGA with your debugging design, loading the design in the toolkit, and linking hardware, follow these steps to run PRBS signal eye tests:

1. Click **Setup**.
   a. Select the generator and checker you want to control.
   b. Select the transmitter and receiver pair you want to control.

c. Click **Create Transceiver Link** and click **Close**.

2. Click **Link Eye Viewer**, and select **Eye Viewer** as the **Test mode**. The **Eye Viewer** mode displays test results as a bathtub curve, heat map, or eye contour representing bit error and phase offset data.
3. Specify the PRBS **Test pattern** and the **Checker mode**. Use **Serial bit comparator** checker mode only for checking a large window of error with custom traffic.

   The checker mode option is only available after you turn on **Enable Eye Viewer block** and **Enable Bit Error Rate Block** in the Reconfiguration Controller component. (Stratix V designs only)
4. Specify **Run length** and **Eye Viewer settings** to control the test coverage and type of Eye Viewer results displayed, respectively.
5. Click **Start** to run the pattern with your settings. Eye Viewer uses the current channel settings to start a phase sweep of the channel. The phase sweep runs 32 iterations. As the run progresses, view the status under **Eye Viewer status**. Use this diagram to compare PMA settings for the same channel and to choose the best combination of PMA settings for a particular channel.
6. When the run completes, the chart displays the characteristics of each run. Click **Stop** to halt the test, change the PMA settings, and re-start the test. Click **Create Report** to export data to a table format for further viewing.

Related Information

PRBS Signal Eye Test Configuration (Stratix V) on page 99

3.9.3. Running Custom Traffic Tests (Stratix V only)

After programming the FPGA with your debugging design, loading the design in the toolkit, and linking hardware, follow these steps to run custom traffic tests:

1. In the Channel Manager, click **Setup**.
2. Select the associated reconfiguration controller.
3. Click **Create Transceiver Link** and click **Close**.
4. Click the **Receiver Eye Viewer** tab.
5. Select **Eye Viewer** as the **Test mode**. The **Eye Viewer** mode displays test results as a bathtub curve, heat map, or eye contour representing bit error and phase offset data.
6. Specify the PRBS **Test pattern**.
7. For **Checker mode**, select **Serial bit comparator**.

   The checker mode option is only available after you turn on **Enable Eye Viewer block** and **Enable Bit Error Rate Block** for the Reconfiguration Controller component.

8. Specify **Run length** and **Eye Viewer settings** to control the test coverage and type of Eye Viewer results displayed, respectively.
9. Click **Start** to run the pattern with your settings. Eye Viewer uses the current channel settings to start a phase sweep of the channel. The phase sweep runs 32 iterations. As the run progresses, view the status under **Eye Viewer status**.
10. When the run completes, the chart displays the characteristics of each run. Click **Stop** to halt the test, change the PMA settings, and re-start the test. Click **Create Report** to export data to a table format for further viewing.

Related Information

Custom Traffic Signal Eye Test Configuration (Stratix V) on page 100

3.9.4. Link Optimization Tests

The Transceiver Toolkit auto sweep test automatically sweeps PMA ranges to determine the transceiver settings that provide the best signal integrity. The toolkit allows you to store a history of the test runs, and keep a record of the best PMA settings.

3.9.4.1. Running the Auto Sweep Test

to run link optimization tests:

1. In the **Transceiver Links** tab, select the channel you want to control.
2. Click **Link Auto Sweep**.
   The **Advanced** tab appears with **Auto sweep** as **Test mode**.
3. Specify the PRBS **Test pattern**.
4. Specify **Run length** experiment with the **Transmitter settings**, and **Receiver settings** to control the test coverage and PMA settings, respectively.
5. Click **Start** to run all combinations of tests meeting the PMA parameter limits. When the run completes the chart is displayed and the characteristics of each run are listed in the run list.
6. You can click **Stop** to halt the test, change the PMA settings, and re-start the test. Click **Create Report** to export data to a table format for further viewing.
7. If you want to determine the best tap settings using decision feedback equalization (DFE):
   a. Set the **DFE mode** to **Off**.
   b. Use Auto Sweep to find optimal PMA settings.
   c. If BER = 0, use the best PMA settings achieved.
   d. If BER > 0, use this PMA setting, and set the minimum and maximum values obtained from Auto Sweep to match this setting. Set the maximum DFE range to limits for each of the three DFE settings.
   e. Run **Create Report** to view the results and determine which DFE setting has the best BER. Use these settings in conjunction with the PMA settings for the best results.

Related Information

- Link Optimization Test Configuration (Stratix V) on page 101

- Instantiating and Parameterizing Intel Arria 10 Debug IP cores on page 103

3.9.4.2. Determining the Best Tap Settings

To determine the best tap settings using decision feedback equalization (DFE):

1. Use Auto Sweep to find optimal PMA settings, while leaving the **DFE mode** set to **Off**.

Option	Description
If BER = 0	Use the best PMA settings achieved.
If BER > 0	Use this PMA setting, and set the minimum and maximum values auto sweep reports to match this setting. Set the maximum DFE range to limits for each of the three DFE settings.

2. Click **Create Report** to view the results and determine which DFE setting has the best BER. Use these settings in conjunction with the PMA settings for the best results.

3.10. Controlling PMA Analog Settings

The Transceiver Toolkit allows you to directly control PMA analog settings while the link is running. For a detailed description of each parameter, refer to the PHY user guide of the corresponding device.

To control PMA analog settings, follow these steps:

1. In the Channel Manager, click **Setup**.
2. In the **Transmitter Channels** tab, define a transmitter without a generator, and click **Create Transmitter Channel**.
3. In the **Receiver Channels** tab, define a receiver without a generator, and click **Create Receiver Channel**.
4. In the **Transceiver Links** tab, select the transmitter and receivers you want to control, and click **Create Transceiver Link**.
5. Click **Close**.
6. Click **Control Receiver Channel**, **Control Transmitter Channel**, or **Control Transceiver Link** to directly control the PMA settings while running.

3.10.1. Intel Arria 10 and Intel Cyclone 10 GX PMA Settings

The following figures show the PMA analog settings.

Figure 36. Transmitter Channel PMA Settings

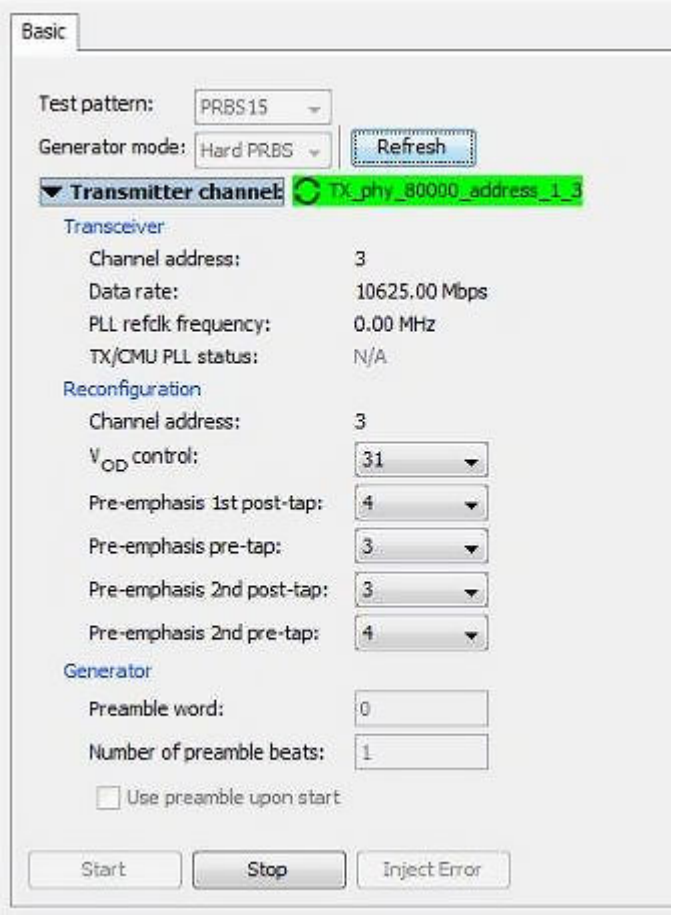


Figure 37. Receiver Channel PMA Settings

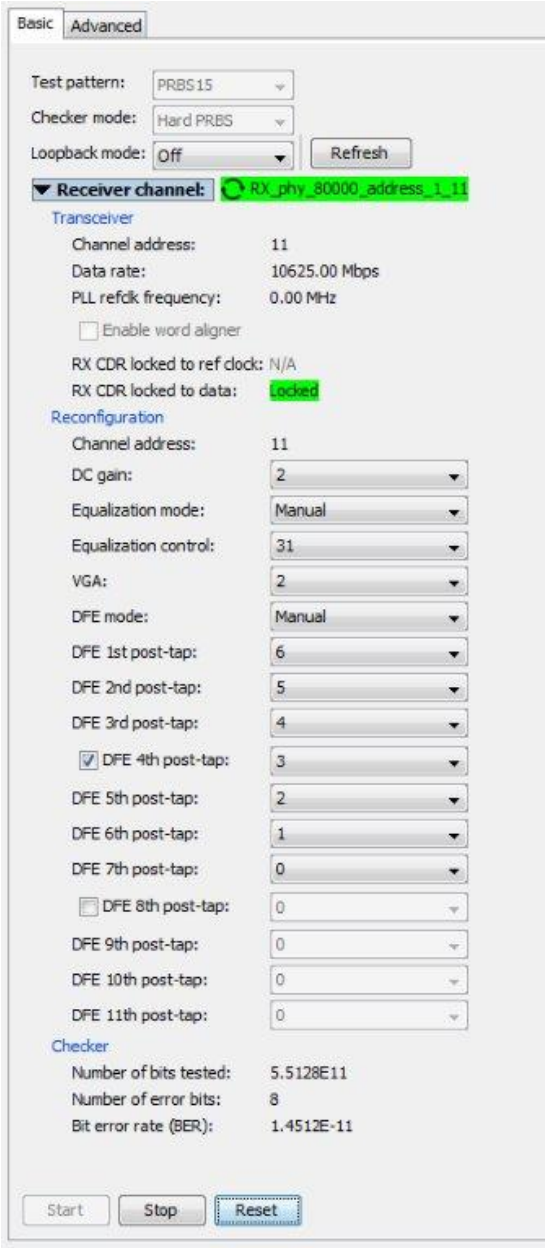
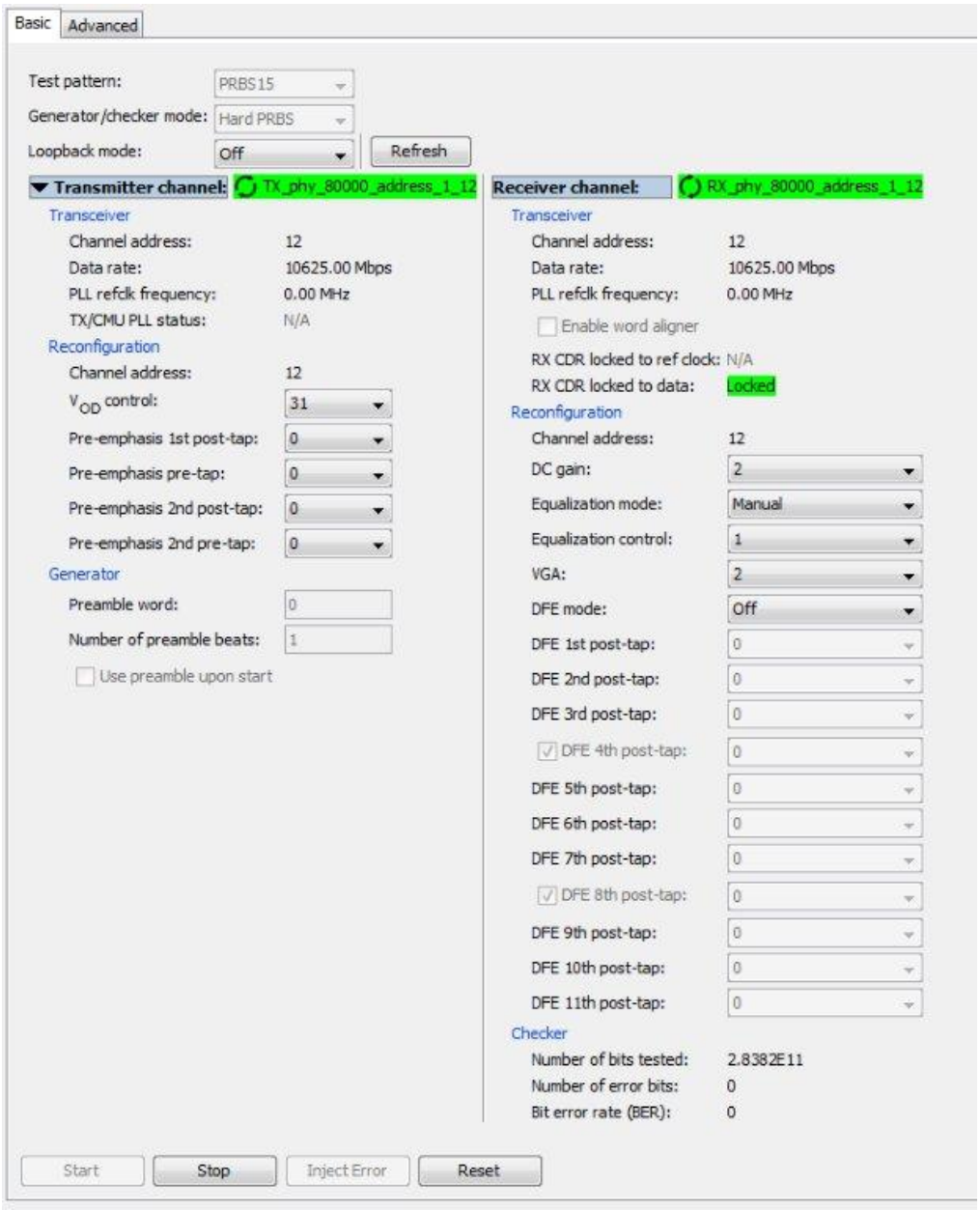


Figure 38. Transceiver Link PMA Settings



Related Information

- Instantiating and Parameterizing Intel Arria 10 Debug IP cores on page 103
- PMA Analog Setting Control Configuration (Stratix V) on page 102
- PMA Parameters

3.11. User Interface Settings Reference

The Transceiver Toolkit user interface contains the following settings:

Table 45. Transceiver Toolkit Control Pane Settings

Settings in alphabetical order. All the settings appear in the **Transceiver Link** control pane.

Setting	Description	Device Families	Control Pane
Alias	Name you choose for the channel.	All supported device families	Transmitter pane Receiver pane
Auto Sweep status	Reports the current and best tested bits, errors, bit error rate, and case count for the current Auto Sweep test.	All supported device families	Receiver pane
Bit error rate (BER)	Reports the number of errors divided by bits tested since the last reset of the checker.	All supported device families	Receiver pane
Channel address	Logical address number of the transceiver channel.	All supported device families	Transmitter pane Receiver pane
Data rate	Data rate of the channel that appears in the project file, or data rate the frequency detector measures. To use the frequency detector, turn on **Enable Frequency Counter** in the Data Pattern Checker IP core or Data Pattern Generator IP core, regenerate the IP cores, and recompile the design. The measured data rate depends on the Avalon management clock frequency that appears in the project file. If you make changes to your settings and want to sample the data rate again, click the **Refresh** button next to the **Data rate**	All supported device families	Transmitter pane Receiver pane
DC gain	Provides an equal boost to the incoming signal across the frequency spectrum.	All supported device families	Receiver pane
DFE mode	Decision feedback equalization (DFE) for improving signal quality. **Device** / **Value**: Stratix V: 1-5; Intel Arria 10: 1-11 In Stratix V devices DFE modes are **Off**, **Manual**, **One-time adaptive mode** and **Adaptation Enabled**. **Adaptation Enabled** mode DFE automatically tries to find the best tap values. In Intel Arria 10 devices, DFE modes are **Off**, **Manual** and **Adaptation Enabled**. DFE in **Adaptation Enabled** mode automatically tries to find the best tap values.	Stratix V Intel Arria 10	Receiver pane
Enable word aligner	Forces the transceiver channel to align to the word you specify.	Stratix V	Receiver pane

continued...

Setting	Description	Device Families	Control Pane
Equalization control	Boosts the high-frequency gain of the incoming signal to compensate for the low-pass filter effects of the physical medium. When you use this option with DFE, use DFE in **Manual** or **Adaptation Enabled** mode. In Stratix V devices, auto sweep supports AEQ one-time adaptation.	All supported device families	Receiver pane
Equalization mode	For Intel Arria 10 devices, you can set **Equalization Mode** to **Manual** or **Triggered**. In Stratix V devices, Adaptive equalization (AEQ) automatically evaluates and selects the best combination of equalizer settings, and turns off **Equalization Control**. The one-time selection determines the best setting and stops searching. You can use AEQ for multiple, independently controlled receiver channels.	All supported device families	Receiver pane
Error rate limit	Turns on or off error rate limits. **Start checking after** specifies the number of bits the toolkit waits before looking at the bit error rate (BER) for the next two checks. **Bit error rate achieves below** sets upper bit error rate limits. If the error rate is better than the set error rate, the test ends. **Bit error rate exceeds** sets lower bit error rate limits. If the error rate is worse than the set error rate, the test ends.	All supported device families	Receiver pane
Generator/Checker mode	Specifies **Data pattern checker** or **Serial bit comparator** for BER tests. If you enable **Serial bit comparator** the Data Pattern Generator sends the PRBS pattern, but the serial bit comparator checks the pattern. In **Bypass mode**, clicking **Start** begins counting on the Serial bit comparator. For BER testing: • Intel Arria 10 devices support the Data Pattern Checker and the Hard PRBS. • Stratix V devices support the Data Pattern Checker and the Serial Bit Checker.	All supported device families	Transmitter pane Receiver pane
Horizontal phase step interval	Specifies the number of horizontal steps to increment when performing a sweep. Increasing the value increases the speed of the test but at a lower resolution. This option only applies to eye contour.	Stratix V	Transmitter pane Receiver pane
Increase test range	For the selected set of controls, increases the span of tests by one unit down for the minimum, and one unit up for the maximum. You can span either PMA Analog controls (non-DFE controls), or the DFE controls. You can quickly set up a test to check if any PMA setting combinations near your current best yields better results. To use, right-click the **Advanced** panel	All supported device families	Receiver pane
Inject Error	Flips one bit to the output of the data pattern generator to introduce an artificial error.	Stratix V	Transmitter pane
Maximum tested bits	Sets the maximum number of bits tested for each test iteration.	All supported device families	Receiver pane
Number of bits tested	Specifies the number of bits tested since the last reset of the checker.	All supported device families	Receiver pane

continued...

Setting	Description	Device Families	Control Pane
Number of error bits	Specifies the number of error bits encountered since the last reset of the checker.	All supported device families	Receiver pane
Number of preamble beats	Number of clock cycles to which the preamble word is sent before the test pattern begins.	Stratix V	Transmitter pane
PLL refclk freq	Channel reference clock frequency that appears in the project file, or reference clock frequency calculated from the measured data rate.	All supported device families	Transmitter pane Receiver pane
Populate with	Right-click the **Advanced** panel to load current values on the device as a starting point, or initially load the best settings auto sweep determines. The Intel Quartus Prime software automatically applies the values you specify in the drop-down lists for the Transmitter settings and Receiver settings.	All supported device families	Receiver pane
Preamble word	Word to send out if you use the preamble mode (only if you use soft PRBS Data Pattern Generator and Checker).	All supported device families	Transmitter pane
Pre-emphasis	This programmable module boosts high frequencies in the transmit data for each transmit buffer signal. This action counteracts possible attenuation in the transmission media. (Stratix V only) Using pre-emphasis can maximize the data eye opening at the far-end receiver.	All supported device families	Transmitter pane
Receiver channel	Specifies the name of the selected receiver channel.	All supported device families	Receiver pane
Refresh Button	After loading the `.pof` file, loads fresh settings from the registers after running dynamic reconfiguration.	All supported device families	Transmitter pane Receiver pane
Reset	Resets the current test.	All supported device families	Receiver pane
Rules Based Configuration (RBC) validity checking	Displays in red any invalid combination of settings for each list under **Transmitter settings** and **Receiver settings**, based on previous settings. When you enable this option, the settings appear in red to indicate the current combination is invalid. This action avoids manually testing invalid settings that you cannot compile for your design, and prevents setting the device into an invalid mode for extended periods of time and potentially damaging the circuits.	All supported device families	Receiver pane
Run length	Sets coverage parameters for test runs.	All supported device families	Transmitter pane Receiver pane
RX CDR PLL status(5)	Shows the receiver in lock-to-reference (LTR) mode. When in auto-mode, if data cannot be locked, this signal alternates in LTD mode if the CDR is locked to data.	All supported device families	Receiver pane

continued...

(5) For Stratix V devices, the Phase Frequency Detector (PFD) is inactive in LTD mode. The `rx_is_lockedtoref` status signal turns on and off randomly, and is not significant in LTD mode.

Setting	Description	Device Families	Control Pane
RX CDR data status	Shows the receiver in lock-to-data (LTD) mode. When in auto-mode, if data cannot be locked, the signal stays high when locked to data and never switches.	All supported device families	Receiver pane
Serial loopback enabled	Inserts a serial loopback before the buffers, allowing you to form a link on a transmitter and receiver pair on the same physical channel of the device.	All supported device families	Transmitter pane Receiver pane
Start	Starts the pattern generator or checker on the channel to verify incoming data.	All supported device families	Transmitter pane Receiver pane
Stop	Stops generating patterns and testing the channel.	All supported device families	Transmitter pane Receiver pane
Target bit error rate	Finds the contour edge of the bit error rate that you select. This option only applies to eye contour mode.	Stratix V	Transmitter pane Receiver pane
Test pattern	Test pattern sent by the transmitter channel. The Data Pattern Checker self-aligns both high and low frequency patterns. Use **Bypass mode** to send user-design data. **Device Family** / **Test Patterns Available** Stratix V: **PRBS7**, **PRBS15**, **PRBS23**, **PRBS31**, **LowFrequency**, **HighFrequency**, and **Bypass mode**. Intel Arria 10: **PRBS9**, **PRBS15**, **PRBS23**, and **PRBS31**.	All supported device families	Transmitter pane Receiver pane
Time limit	Specifies the time limit unit and value to have a maximum bounds time limit for each test iteration.	All supported device families	Receiver
Transmitter channel	Specifies the name of the selected transmitter channel.	All supported device families	Transmitter pane
TX/CMU PLL status	Specifies whether the transmitter channel PLL is locked to the reference clock.	All supported device families	Transmitter pane
Use preamble upon start	If turned on, sends the preamble word before the test pattern. If turned off, starts sending the test pattern immediately.	All supported device families	Transmitter pane
Vertical phase step interval	Specify the number of vertical steps to increment when performing a sweep. Increasing the value increases the speed of the test but at a lower resolution. This option only applies to the eye contour.	Stratix V	Transmitter pane Receiver pane
V_{OD} control	Programmable transmitter differential output voltage.	All supported device families	Transmitter pane

Related Information

Channel Manager on page 92

3.12. Troubleshooting Common Errors

Missing high-speed link pin connections

Check the pin connections to identify high-speed links (tx_p/n and rx_p/n) are missing. When porting an older design to the latest version of the Intel Quartus Prime software, make sure that these connections exist after porting.

Reset Issues:

Ensure that the reset input to the Transceiver Native PHY, Transceiver Reset Controller, and ATX PLL Intel FPGA IPs is not held active (`1'b1`). The Transceiver Toolkit highlights in red all the Transceiver Native PHY channels that you are setting up.

Unconnected `reconfig_clk`

You must connect and drive the `reconfig_clk` input to the Transceiver Native PHY and ATX PLL Intel FPGA IPs. Otherwise, the toolkit does not display the transceiver link channel.

3.13. Scripting API Reference

The Intel Quartus Prime software provides an API to access Transceiver Toolkit functions using Tcl commands, and script tasks such as linking device resources and identifying high-speed serial links.

To save the project setup in a Tcl script for use in subsequent testing sessions:

1. Set up and define links that describe the entire physical system.
2. Click **Save Tcl Script** to save the setup for future use.

You can also build a custom test routine script.

To run the scripts, double-click the script name in the System Explorer scripts folder.

To view a list of the available Tcl command descriptions from the Tcl Console window, including example usage:

1. In the Tcl console, type `help help`. The Console displays all Transceiver Toolkit Tcl commands.
2. Type `help` <*command name*>. The Console displays the command description.

3.13.1. Transceiver Toolkit Commands

The following tables list the available Transceiver Toolkit scripting commands.

Table 46. Transceiver Toolkit channel_rx Commands

Command	Arguments	Function
`transceiver_channel_rx_get_data`	*<service-path>*	Returns a list of the current checker data. The results are in the order of number of bits, number of errors, and bit error rate.
`transceiver_channel_rx_get_dcgain`	*<service-path>*	Gets the DC gain value on the receiver channel.
`transceiver_channel_rx_get_dfe_tap_value`	*<service-path> <tap position>*	Gets the current tap value of the channel you specify at the tap position you specify.
`transceiver_channel_rx_get_eqctrl`	*<service-path>*	Gets the equalization control value on the receiver channel.
`transceiver_channel_rx_get_pattern`	*<service-path>*	Returns the current data checker pattern by name.
`transceiver_channel_rx_has_dfe`	*<service-path>*	Reports whether the channel you specify has the DFE feature available.
`transceiver_channel_rx_has_eye_viewer`	*<service-path>*	(Stratix V only) Reports whether the Eye Viewer feature is available for the channel you specify.
`transceiver_channel_rx_is_checking`	*<service-path>*	Returns non-zero if the checker is running.
`transceiver_channel_rx_is_dfe_enabled`	*<service-path>*	Reports whether the DFE feature is enabled on the channel you specify.
`transceiver_channel_rx_is_locked`	*<service-path>*	Returns non-zero if the checker is locked onto the incoming data.
`transceiver_channel_rx_reset_counters`	*<service-path>*	Resets the bit and error counters inside the checker.
`transceiver_channel_rx_reset`	*<service-path>*	Resets the channel you specify.
`transceiver_channel_rx_set_dcgain`	*<service-path> <value>*	Sets the DC gain value on the receiver channel.
`transceiver_channel_rx_set_dfe_enabled`	*<service-path> <disable(0)/ enable(1)>*	Enables or disables the DFE feature on the channel you specify.
`transceiver_channel_rx_set_dfe_tap_value`	*<service-path> <tap position> <tap value>*	Sets the current tap value of the channel you specify at the tap position you specify to the value you specify.
`transceiver_channel_rx_set_dfe_adaptive`	*<service-path> <adaptive-mode>*	Sets DFE adaptation mode of the channel you specify. **Value** / **Description** 0 — off 1 — adaptive 2 — one-time adaptive

continued...

Command	Arguments	Function
`transceiver_channel_rx_set_eqctrl`	*<service-path> <value>*	Sets the equalization control value on the receiver channel.
`transceiver_channel_rx_start_checking`	*<service-path>*	Starts the checker.
`transceiver_channel_rx_stop_checking`	*<service-path>*	Stops the checker.
`transceiver_channel_rx_get_eye_viewer_phase_step`	*<service-path>*	(Stratix V only) Gets the current phase step of the channel you specify.
`transceiver_channel_rx_set_pattern`	*<service-path> <pattern-name>*	Sets the expected pattern to the one specified by the pattern name.
`transceiver_channel_rx_is_eye_viewer_enabled`	*<service-path>*	(Stratix V only) Reports whether the Eye Viewer feature is enabled on the channel you specify.
`transceiver_channel_rx_set_eye_viewer_enabled`	*<service-path> <disable(0)/ enable(1)>*	(Stratix V only) Enables or disables the Eye Viewer feature on the channel you specify.
`transceiver_channel_rx_set_eye_viewer_phase_step`	*<service-path> <phase step>*	(Stratix V only) Sets the phase step of the channel you specify.
`transceiver_channel_rx_set_word_aligner_enabled`	*<service-path> <disable(0)/ enable(1)>*	Enables or disables the word aligner of the channel you specify.
`transceiver_channel_rx_is_word_aligner_enabled`	*<service-path> <disable(0)/ enable(1)>*	Reports whether the word aligner feature is enabled on the channel you specify.
`transceiver_channel_rx_is_locked`	*<service-path>*	Returns non-zero if the checker is locked onto the incoming signal.
`transceiver_channel_rx_is_rx_locked_to_data`	*<service-path>*	Returns `1` if transceiver is in lock to data (LTD) mode. Otherwise `0`.
`transceiver_channel_rx_is_rx_locked_to_ref`	*<service-path>*	Returns `1` if transceiver is in lock to reference (LTR) mode. Otherwise `0`.
`transceiver_channel_rx_has_eye_viewer_1d`	*<service-path>*	(Stratix V only) Detects whether the eye viewer in *<service-path>* supports 1D-Eye Viewer mode.
`transceiver_channel_rx_set_1deye_mode`	*<service-path> <disable(0)/ enable(1)>*	(Stratix V only) Enables or disables 1D-Eye Viewer mode.
`transceiver_channel_rx_get_1deye_mode`	*<service-path>*	(Stratix V only) Returns whether 1D-Eye Viewer mode is on or off.

Table 47. Transceiver Toolkit channel_tx Commands

Command	Arguments	Function
`transceiver_channel_tx_disable_preamble`	*<service-path>*	Disables the preamble mode at the beginning of generation.
`transceiver_channel_tx_enable_preamble`	*<service-path>*	Enables the preamble mode at the beginning of generation.
`transceiver_channel_tx_get_number_of_preamble_beats`	*<service-path>*	Returns the number of beats to send out the preamble word.
`transceiver_channel_tx_get_pattern`	*<service-path>*	Returns the pattern.

continued...

Command	Arguments	Function
`transceiver_channel_tx_get_preamble_word`	*<service-path>*	Returns the preamble word.
`transceiver_channel_tx_get_preemph0t`	*<service-path>*	Gets the pre-emphasis first pre-tap value on the transmitter channel.
`transceiver_channel_tx_get_preemph1t`	*<service-path>*	Gets the pre-emphasis first post-tap value on the transmitter channel.
`transceiver_channel_tx_get_preemph2t`	*<service-path>*	Gets the pre-emphasis second post-tap value on the transmitter channel.
`transceiver_channel_tx_get_preemph3t`	*<service-path>*	Gets the pre-emphasis second pre-tap value on the transmitter channel.
`transceiver_channel_tx_get_vodctrl`	*<service-path>*	Gets the V_{OD} control value on the transmitter channel.
`transceiver_channel_tx_inject_error`	*<service-path>*	Injects a 1-bit error into the generator's output.
`transceiver_channel_tx_is_generating`	*<service-path>*	Returns non-zero if the generator is running.
`transceiver_channel_tx_is_preamble_enabled`	*<service-path>*	Returns non-zero if preamble mode is enabled.
`transceiver_channel_tx_set_number_of_preamble_beats`	*<service-path> <number-of-preamble-beats>*	Sets the number of beats to send out the preamble word.
`transceiver_channel_tx_set_pattern`	*<service-path> <pattern-name>*	Sets the output pattern to the one specified by the pattern name.
`transceiver_channel_tx_set_preamble_word`	*<service-path> <preamble-word>*	Sets the preamble word to be sent out.
`transceiver_channel_tx_set_preemph0t`	*<service-path> <value>*	Sets the pre-emphasis first pre-tap value on the transmitter channel.
`transceiver_channel_tx_set_preemph1t`	*<service-path> <value>*	Sets the pre-emphasis first post-tap value on the transmitter channel.
`transceiver_channel_tx_set_preemph2t`	*<service-path> <value>*	Sets the pre-emphasis second post-tap value on the transmitter channel.
`transceiver_channel_tx_set_preemph3t`	*<service-path> <value>*	Sets the pre-emphasis second pre-tap value on the transmitter channel.
`transceiver_channel_tx_set_vodctrl`	*<service-path> <vodctrl value>*	Sets the V_{OD} control value on the transmitter channel.
`transceiver_channel_tx_start_generation`	*<service-path>*	Starts the generator.
`transceiver_channel_tx_stop_generation`	*<service-path>*	Stops the generator.

Table 48. Transceiver Toolkit Transceiver Toolkit debug_link Commands

Command	Arguments	Function
`transceiver_debug_link_get_pattern`	*<service-path>*	Gets the pattern the link uses to run the test.
`transceiver_debug_link_is_running`	*<service-path>*	Returns non-zero if the test is running on the link.
`transceiver_debug_link_set_pattern`	*<service-path> <data pattern>*	Sets the pattern the link uses to run the test.
`transceiver_debug_link_start_running`	*<service-path>*	Starts running a test with the currently selected test pattern.
`transceiver_debug_link_stop_running`	*<service-path>*	Stops running the test.

Table 49. Transceiver Toolkit reconfig_analog Commands

Command	Arguments	Function
`transceiver_reconfig_analog_get_logical_channel_address`	*<service-path>*	Gets the transceiver logic channel address currently set.
`transceiver_reconfig_analog_get_rx_dc gain`	*<service-path>*	Gets the DC gain value on the receiver channel specified by the current logic channel address.
`transceiver_reconfig_analog_get_rx_eq ctrl`	*<service-path>*	Gets the equalization control value on the receiver channel specified by the current logic channel address.
`transceiver_reconfig_analog_get_tx_pr eemph0t`	*<service-path>*	Gets the pre-emphasis first pre-tap value on the transmitter channel specified by the current logic channel address.
`transceiver_reconfig_analog_get_tx_pr eemph1t`	*<service-path>*	Gets the pre-emphasis first post-tap value on the transmitter channel specified by the current logic channel address.
`transceiver_reconfig_analog_get_tx_pr eemph2t`	*<service-path>*	Gets the pre-emphasis second post-tap value on the transmitter channel specified by the current logic channel address.
`transceiver_reconfig_analog_get_tx_vo dctrl`	*<service-path>*	Gets the V_{OD} control value on the transmitter channel specified by the current logic channel address.
`transceiver_reconfig_analog_set_logical_channel_address`	*<service-path> <logic channel address>*	Sets the transceiver logic channel address.
`transceiver_reconfig_analog_set_rx_dc gain`	*<service-path> <dc_gain value>*	Sets the DC gain value on the receiver channel specified by the current logic channel address
`transceiver_reconfig_analog_set_rx_eq ctrl`	*<service-path> <eqctrl value>*	Sets the equalization control value on the receiver channel specified by the current logic channel address.
`transceiver_reconfig_analog_set_tx_pr eemph0t`	*<service-path> <value>*	Sets the pre-emphasis first pre-tap value on the transmitter channel specified by the current logic channel address.

continued...

Command	Arguments	Function
`transceiver_reconfig_analog_set_tx_preemph1t`	*<service-path> < value>*	Sets the pre-emphasis first post-tap value on the transmitter channel specified by the current logic channel address.
`transceiver_reconfig_analog_set_tx_preemph2t`	*<service-path> <value>*	Sets the pre-emphasis second post-tap value on the transmitter channel specified by the current logic channel address.
`transceiver_reconfig_analog_set_tx_vodctrl`	*<service-path> <vodctrl value>*	Sets the V_{OD} control value on the transmitter channel specified by the current logic channel address.

Table 50. Transceiver Toolkit Decision Feedback Equalization (DFE) Commands

Command	Arguments	Function
`alt_xcvr_reconfig_dfe_get_logical_channel_address`	*<service-path>*	Gets the logic channel address that other `alt_xcvr_reconfig_dfe` commands use to apply.
`alt_xcvr_reconfig_dfe_is_enabled`	*<service-path>*	Reports whether the DFE feature is enabled on the previously channel you specify.
`alt_xcvr_reconfig_dfe_set_enabled`	*<service-path> <disable(0)/ enable(1)>*	Enables or disables the DFE feature on the previously channel you specify.
`alt_xcvr_reconfig_dfe_set_logical_channel_address`	*<service-path> <logic channel address>*	(Stratix V only) Sets the logic channel address that other `alt_xcvr_reconfig_eye_viewer` commands use.
`alt_xcvr_reconfig_dfe_set_tap_value`	*<service-path> <tap position> <tap value>*	Sets the tap value at the previously channel you specify at specified tap position and value.

Table 51. Transceiver Toolkit Eye Monitor Commands (Stratix V only)

Command	Arguments	Function
`alt_xcvr_custom_is_word_aligner_enabled`	*<service-path> <disable(0)/ enable(1)>*	Reports whether the word aligner feature is enabled on the previously channel you specify.
`alt_xcvr_custom_set_word_aligner_enabled`	*<service-path> <disable(0)/ enable(1)>*	Enables or disables the word aligner of the previously channel you specify.
`alt_xcvr_custom_is_rx_locked_to_data`	*<service-path>*	Returns whether the receiver CDR is locked to data.
`alt_xcvr_custom_is_rx_locked_to_ref`	*<service-path>*	Returns whether the receiver CDR PLL is locked to the reference clock.
`alt_xcvr_custom_is_serial_loopback_enabled`	*<service-path>*	Returns whether the serial loopback mode of the previously channel you specify is enabled.
`alt_xcvr_custom_set_serial_loopback_enabled`	*<service-path> <disable(0)/ enable(1)>*	Enables or disables the serial loopback mode of the previously channel you specify.

continued...

Command	Arguments	Function
`alt_xcvr_custom_is_tx_pll_locked`	*<service-path>*	Returns whether the transmitter PLL is locked to the reference clock.
`alt_xcvr_reconfig_eye_viewer_get_logical_channel_address`	*<service-path>*	Gets the logic channel address on which other `alt_reconfig_eye_viewer` commands use.
`alt_xcvr_reconfig_eye_viewer_get_phase_step`	*<service-path>*	Gets the current phase step of the previously channel you specify.
`alt_xcvr_reconfig_eye_viewer_is_enabled`	*<service-path>*	Reports whether the Eye Viewer feature is enabled on the previously channel you specify.
`alt_xcvr_reconfig_eye_viewer_set_enabled`	*<service-path> <disable(0)/ enable(1)>*	Enables or disables the Eye Viewer feature on the previously channel you specify. Setting a value of 2 enables both Eye Viewer and the Serial Bit Comparator.
`alt_xcvr_reconfig_eye_viewer_set_logical_channel_address`	*<service-path> <logic channel address>*	Sets the logic channel address that other `alt_reconfig_eye_viewer` commands use.
`alt_xcvr_reconfig_eye_viewer_set_phase_step`	*<service-path> <phase step>*	Sets the phase step of the previously channel you specify.
`alt_xcvr_reconfig_eye_viewer_has_ber_checker`	*<service-path>*	Detects whether the eye viewer pointed to by *<service-path>* supports the Serial Bit Comparator.
`alt_xcvr_reconfig_eye_viewer_ber_checker_is_enabled`	*<service-path>*	Detects whether the Serial Bit Comparator is enabled.
`alt_xcvr_reconfig_eye_viewer_ber_checker_start`	*<service-path>*	Starts the Serial Bit Comparator counters.
`alt_xcvr_reconfig_eye_viewer_ber_checker_stop`	*<service-path>*	Stops the Serial Bit Comparator counters.
`alt_xcvr_reconfig_eye_viewer_ber_checker_reset_counters`	*<service-path>*	Resets the Serial Bit Comparator counters.
`alt_xcvr_reconfig_eye_viewer_ber_checker_is_running`	*<service-path>*	Reports whether the Serial Bit Comparator counters are currently running or not.
`alt_xcvr_reconfig_eye_viewer_ber_checker_get_data`	*<service-path>*	Gets the current total bit, error bit, and exception counts for the Serial Bit Comparator.
`alt_xcvr_reconfig_eye_viewer_has_1deye`	*<service-path>*	Detects whether the eye viewer pointed to by *<service-path>* supports 1D-Eye Viewer mode.
`alt_xcvr_reconfig_eye_viewer_set_1deye_mode`	*<service-path> <disable(0)/ enable(1)*	Enables or disables 1D-Eye Viewer mode.
`alt_xcvr_reconfig_eye_viewer_get_1deye_mode`	*<service-path>*	Gets the enable or disabled state of 1D-Eye Viewer mode.

Table 52. Channel Type Commands

Command	Arguments	Function
`get_channel_type`	*<service-path> <logical-channel-num>*	Reports the detected type (GX/GT) of channel *<logical-channel-num>* for the reconfiguration block located at *<service-path>*.
`set_channel_type`	*<service-path> <logical-channel-num> <channel-type>*	Overrides the detected channel type of channel *<logical-channel-num>* for the reconfiguration block located at *<service-path>* to the type specified (0:GX, 1:GT).

Table 53. Loopback Commands

Command	Arguments	Function
`loopback_get`	*<service-path>*	Returns the value of a setting or result on the loopback channel. Available results include: • Status—running or stopped. • Bytes—number of bytes sent through the loopback channel. • Errors—number of errors reported by the loopback channel. • Seconds—number of seconds since the loopback channel was started.
`loopback_set`	*<service-path>*	Sets the value of a setting controlling the loopback channel. Some settings are only supported by particular channel types. Available settings include: • Timer—number of seconds for the test run. • Size—size of the test data. • Mode—mode of the test.
`loopback_start`	*<service-path>*	Starts sending data through the loopback channel.
`loopback_stop`	*<service-path>*	Stops sending data through the loopback channel.

3.13.2. Data Pattern Generator Commands

You can use Data Pattern Generator commands to control data patterns for debugging transceiver channels. You must instantiate the Data Pattern Generator component to support these commands.

Table 54. Soft Data Pattern Generator Commands

Command	Arguments	Function
`data_pattern_generator_start`	*<service-path>*	Starts the data pattern generator.
`data_pattern_generator_stop`	*<service-path>*	Stops the data pattern generator.
`data_pattern_generator_is_generating`	*<service-path>*	Returns non-zero if the generator is running.
`data_pattern_generator_inject_error`	*<service-path>*	Injects a 1-bit error into the generator output.
`data_pattern_generator_set_pattern`	*<service-path> <pattern-name>*	Sets the output pattern that *<pattern-name>* specifies. **Value:** • PRBS7 • PRBS15 • PRBS23 • PRBS31 **Description:** Pseudo-random binary sequences. PRBS files are clear text, and you can modify the PRBS files.

continued...

Command	Arguments	Function
		<table><tr><th>Value</th><th>Description</th></tr><tr><td>HF</td><td>Outputs high frequency, constant pattern of alternating 0s and 1s</td></tr><tr><td>LF</td><td>Outputs low frequency, constant pattern of 10b'1111100000 for 10-bit symbols and 8b'11110000 for 8-bit symbols</td></tr></table>
`data_pattern_generator_get_pattern`	*<service-path>*	Returns currently selected output pattern.
`data_pattern_generator_get_available_patterns`	*<service-path>*	Returns a list of available data patterns by name.
`data_pattern_generator_enable_preamble`	*<service-path>*	Enables the preamble mode at the beginning of generation.
`data_pattern_generator_disable_preamble`	*<service-path>*	Disables the preamble mode at the beginning of generation.
`data_pattern_generator_is_preamble_enabled`	*<service-path>*	Returns a non-zero value if preamble mode is enabled.
`data_pattern_generator_set_preamble_word`	*<preamble-word>*	Sets the preamble word (could be 32-bit or 40-bit).
`data_pattern_generator_get_preamble_word`	*<service-path>*	Gets the preamble word.
`data_pattern_generator_set_preamble_beats`	*<service-path><number-of-preamble-beats>*	Sets the number of beats to send out in the preamble word.
`data_pattern_generator_get_preamble_beats`	*<service-path>*	Returns the currently set number of beats to send out in the preamble word.
`data_pattern_generator_fcnter_start`	*<service-path><max-cycles>*	Sets the max cycle count and starts the frequency counter.
`data_pattern_generator_check_status`	*<service-path>*	Queries the data pattern generator for current status. Returns a bitmap indicating the status, with bits defined as follows:<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Enabled</td></tr><tr><td>1</td><td>Bypass enabled</td></tr><tr><td>2</td><td>Avalon</td></tr><tr><td>3</td><td>Sink ready</td></tr><tr><td>4</td><td>Source valid</td></tr></table>

continued...

Command	Arguments	Function
		<table><tr><th>Value</th><th>Description</th></tr><tr><td>5</td><td>Frequency counter enabled</td></tr></table>
data_pattern_generator_fcnter_report	<*service-path*><*force-stop*>	Reports the current measured clock ratio, stopping the counting first depending on <*force-stop*>.

Table 55. Hard Data Pattern Generator Commands

Command	Arguments	Function
hard_prbs_generator_start	<*service-path*>	Starts the generator that you specify.
hard_prbs_generator_stop	<*service-path*>	Stops the generator that you specify.
hard_prbs_generator_is_generating	<*service-path*>	Checks the generation status. Returns: <table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Generating</td></tr><tr><td>1</td><td>Otherwise</td></tr></table>
hard_prbs_generator_set_pattern	<*service-path*> <*pattern*>	Sets the pattern of the hard PRBS generator you specify to `pattern`.
hard_prbs_generator_get_pattern	<*service-path*>	Returns the current pattern for a given hard PRBS generator.
hard_prbs_generator_get_available_patterns	<*service-path*>	Returns the available patterns for a given hard PRBS generator.

3.13.3. Data Pattern Checker Commands

You can use Data Pattern Checker commands to verify your generated data patterns. You must instantiate the Data Pattern Checker component to support these commands.

Table 56. Soft Data Pattern Checker Commands

Command	Arguments	Function
data_pattern_checker_start	<*service-path*>	Starts the data pattern checker.
data_pattern_checker_stop	<*service-path*>	Stops the data pattern checker.
data_pattern_checker_is_checking	<*service-path*>	Returns a non-zero value if the checker is running.
data_pattern_checker_is_locked	<*service-path*>	Returns non-zero if the checker is locked onto the incoming data.
data_pattern_checker_set_pattern	<*service-path*> <*pattern-name*>	Sets the expected pattern to <*pattern-name*>.
data_pattern_checker_get_pattern	<*service-path*>	Returns the currently selected expected pattern by name.
data_pattern_checker_get_available_patterns	<*service-path*>	Returns a list of available data patterns by name.

continued...

Command	Arguments	Function
data_pattern_checker_get_data	<service-path>	Returns a list of the current checker data. The results are in the following order: number of bits, number of errors, and bit error rate.
data_pattern_checker_reset_counters	<service-path>	Resets the bit and error counters inside the checker.
data_pattern_checker_fcnter_start	<service-path><max-cycles>	Sets the max cycle count and starts the frequency counter.
data_pattern_checker_check_status	<service-path> <service-path>	Queries the data pattern checker for current status. Returns a bitmap indicating status: Value: 0 — Status: Enabled Value: 1 — Status: Locked Value: 2 — Status: Bypass enabled Value: 3 — Status: Avalon Value: 4 — Status: Sink ready Value: 5 — Status: Source valid Value: 6 — Status: Frequency counter enabled
data_pattern_checker_fcnter_report	<service-path><force-stop>	Reports the current measured clock ratio, stopping the counting first depending on <force-stop>.

Table 57. Hard Data Pattern Checker Commands

Command	Arguments	Function
hard_prbs_checker_start	<service-path>	Starts the specified hard PRBS checker.
hard_prbs_checker_stop	<service-path>	Stops the specified hard PRBS checker.
hard_prbs_checker_is_checking	<service-path>	Checks the running status of the specified hard PRBS checker. Returns a non-zero value if the checker is running.
hard_prbs_checker_set_pattern	<service-path> <pattern>	Sets the pattern of the specified hard PRBS checker to parameter <pattern>.
hard_prbs_checker_get_pattern	<service-path>	Returns the current pattern for a given hard PRBS checker.
hard_prbs_checker_get_available_patterns	<service-path>	Returns the available patterns for a given hard PRBS checker.
hard_prbs_checker_get_data	<service-path>	Returns the current bit and error count data from the specified hard PRBS checker.
hard_prbs_checker_reset_counters	<service-path>	Resets the bit and error counts of the specified hard PRBS checker.

3.14. Debugging Transceiver Links Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2018.07.03	18.0.0	• Added Device Family column to table: *Transceiver Toolkit Control Pane Settings*
2017.11.06	17.1.0	• Renamed *EyeQ* to *Eye Viewer*. • Updated topic "Transceiver Debugging Flow" and renamed to "Transceiver Debugging Flow Walkthrough". • Updated instructions for instantiating and parameterizing Debug IP cores. — Removed figure: "Altera Debug Master Endpoint Block Diagram". • Added step on programming designs as a part of the debugging flow. • Updated information about debugging transceiver links for Intel Arria 10 devices.
2016.10.31	16.1.0	• Removed EyeQ support for Intel Arria 10. • Renamed *"Continuous Adaptation"* to *"Adaptation Enabled"*.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*. • Added description of new Refresh button. • Added description of VGA dialog box. • Added two tables in Transceiver Toolkit Commands section. — Hard Data Pattern Generator Commands — Hard Data Pattern Checker Commands • Separated Arria 10 and Stratix V system configuration steps.
May 2015	15.0.0	• Added section about Implementation Differences Between Stratix V and Arria 10. • Added section about Recommended Flow for Arria 10 Transceiver Toolkit Design with the Quartus II Software. • Added section about Transceiver Toolkit Troubleshooting • Updated the following sections with information about using the Transceiver Toolkit with Arria 10 devices: — Serial Bit Comparator Mode — Arria 10 Support and Limitations — Configuring BER Tests — Configuring PRBS Signal Eye Tests — Adapting Altera Design Examples — Modifying Design Examples — Configuring Custom Traffic Signal Eye Tests — Configuring Link Optimization Tests — Configuring PMA Analog Setting Control — Running BER Tests — Toolkit GUI Setting Reference • Reworked Table: Transceiver Toolkit IP Core Configuration • Replaced Figure: EyeQ Settings and Status Showing Results of Two Test Runs with Figure: EyeQ Settings and Status Showing Results of Three Test Runs. • Added Figure: Arria 10 Altera Debug Master Endpoint Block Diagram. • Added Figure: BER Test Configuration (Arria10/ Gen 10/ 20nm) Block Diagram. • Added Figure: PRBS Signal Test Configuration (Arria 10/ 20nm) Block Diagram. • Added Figure: Custom Traffic Signal Eye Test Configuration (Arria 10/ Gen 10/ 20nm) Block Diagram.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• Added Figure: PMA Analog Setting Control Configuration (Arria 10/ Gen 10/ 20nm) Block Diagram. • Added Figure: One Channel Loopback Mode (Arria 10/ 20nm) Block Diagram. • Added Figure: Four Channel Loopback Mode (Arria 10/ Gen 10/ 20nm) Block Diagram. Software Version 15.0 Limitations • Transceiver Toolkit supports EyeQ for Arria 10 designs. • Supports optional hard acceleration for EyeQ. This allows for much faster EyeQ data collection. Enable this in the Arria 10 Transceiver Native PHY IP core under the **Dynamic Configuration** tab. Turn on **Enable ODI acceleration logic**.
December, 2014	14.1.0	• Added section about Arria 10 support and limitations.
June, 2014	14.0.0	• Updated GUI changes for Channel Manager with popup menus, IP Catalog, Quartus II, and Qsys. • Added ADME and JTAG debug link info for Arria 10. • Added instructions to run Tcl script from command line. • Added heat map display option. • Added procedure to use internal PLL to generate reconfig_clk. • Added note stating RX CDR PLL status can toggle in LTD mode.
November, 2013	13.1.0	• Reorganization and conversion to DITA.
May, 2013	13.0.0	• Added Conduit Mode Support, Serial Bit Comparator, Required Files and Tcl command tables.
November, 2012	12.1.0	• Minor editorial updates. Added Tcl help information and removed Tcl command tables. Added 28-Gbps Transceiver support section.
August, 2012	12.0.1	• General reorganization and revised steps in modifying Altera example designs.
June, 2012	12.0.0	• Maintenance release for update of Transceiver Toolkit features.
November, 2011	11.1.0	• Maintenance release for update of Transceiver Toolkit features.
May, 2011	11.0.0	• Added new Tcl scenario.
December, 2010	10.1.0	• Changed to new document template. Added new 10.1 release features.
August, 2010	10.0.1	• Corrected links.
July 2010	10.0.0	• Initial release.

4. Quick Design Debugging Using Signal Probe

The Signal Probe incremental routing feature helps reduce the hardware verification process and time-to-market for system-on-a-programmable-chip (SOPC) designs. Easy access to internal device signals is important in the design or debugging process. The Signal Probe feature makes design verification more efficient by routing internal signals to I/O pins quickly without affecting the design. When you start with a fully routed design, you can select and route signals for debugging to either previously reserved or currently unused I/O pins.

The Signal Probe feature supports the Arria series, Cyclone series, MAX II, and Stratix series device families.

Related Information

System Debugging Tools Overview on page 7

4.1. Design Flow Using Signal Probe

The Signal Probe feature allows you to reserve available pins and route internal signals to those reserved pins, while preserving the behavior of your design. Signal Probe is an effective debugging tool that provides visibility into your FPGA.

You can reserve pins for Signal Probe and assign I/O standards after a full compilation. Each Signal Probe-source to Signal Probe-pin connection is implemented as an engineering change order (ECO) that is applied to your netlist after a full compilation.

To route the internal signals to the device's reserved pins for Signal Probe, perform the following tasks:

1. Perform a full compilation.
2. Reserve Signal Probe Pins.
3. Assign Signal Probe sources.
4. Add registers between pipeline paths and Signal Probe pins.
5. Perform a Signal Probe compilation.
6. Analyze the results of a Signal Probe compilation.

4.1.1. Perform a Full Compilation

You must complete a full compilation to generate an internal netlist containing a list of internal nodes to probe.

To perform a full compilation, on the Processing menu, click **Start Compilation**.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

4.1.2. Reserve Signal Probe Pins

Signal Probe pins can only be reserved after a full compilation. You can also probe any unused I/Os of the device. Assigning sources is a simple process after reserving Signal Probe pins. The sources for Signal Probe pins are the internal nodes and registers in the post-compilation netlist that you want to probe.

Note: Although you can reserve Signal Probe pins using many features within the Intel Quartus Prime software, including the Pin Planner and the Tcl interface, you should use the **Signal Probe Pins** dialog box to create and edit your Signal Probe pins.

4.1.3. Assign Signal Probe Sources

A Signal Probe source can be any combinational node, register, or pin in your post-compilation netlist. To find a Signal Probe source, in the Node Finder, use the Signal Probe filter to remove all sources that cannot be probed. You might not be able to find a particular internal node because the node can be optimized away during synthesis, or the node cannot be routed to the Signal Probe pin. For example, you cannot probe nodes and registers within Gigabit transceivers in Stratix IV devices because there are no physical routes available to the pins.

Note: To probe virtual I/O pins generated in low-level partitions in an incremental compilation flow, select the source of the logic that feeds the virtual pin as your Signal Probe source pin.

Because Signal Probe pins are implemented and routed as ECOs, turning the **Signal Probe enable** option on or off is the same as selecting **Apply Selected Change** or **Restore Selected Change** in the Change Manager window. If the Change Manager window is not visible at the bottom of your screen, on the View menu, point to **Utility Windows** and click **Change Manager**.

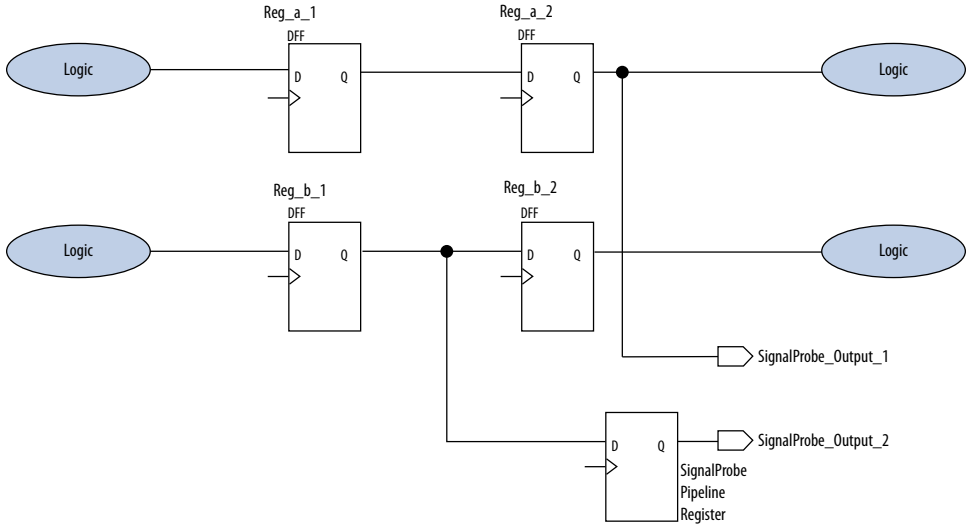
4.1.4. Add Registers Between Pipeline Paths and Signal Probe Pins

You can specify the number of registers placed between a Signal Probe source and a Signal Probe pin. The registers synchronize data to a clock and control the latency of the Signal Probe outputs. The Signal Probe feature automatically inserts the number of registers specified into the Signal Probe path.

The figure shows a single register between the Signal Probe source `Reg_b_1` and Signal Probe `Signal Probe_Output_2` output pin added to synchronize the data between the two Signal Probe output pins.

Note: When you add a register to a Signal Probe pin, the Signal Probe compilation attempts to place the register to best meet timing requirements. You can place Signal Probe registers either near the Signal Probe source to meet f_{MAX} requirements, or near the I/O to meet t_{CO} requirements.

Figure 39. Synchronizing Signal Probe Outputs with a Signal Probe Register



In addition to clock input for pipeline registers, you can also specify a reset signal pin for pipeline registers. To specify a reset pin for pipeline registers, use the Tcl command `make_sp`.

4.1.5. Perform a Signal Probe Compilation

Perform a Signal Probe compilation to route your Signal Probe pins. A Signal Probe compilation saves and checks all netlist changes without recompiling the other parts of the design. A Signal Probe compilation takes a fraction of the time of a full compilation to finish. The design's current placement and routing are preserved.

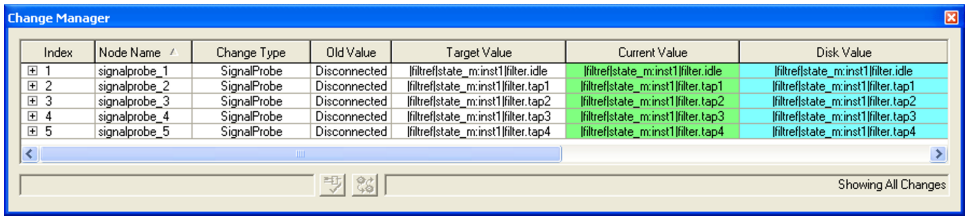
To perform a Signal Probe compilation, on the Processing menu, point to **Start** and click **Start Signal Probe Compilation**.

4.1.6. Analyze the Results of a Signal Probe Compilation

After a Signal Probe compilation, the results are available in the compilation report file. Each Signal Probe pin is displayed in the **Signal Probe Fitting Result** page in the **Fitter** section of the Compilation Report. To view the status of each Signal Probe pin in the **Signal Probe Pins** dialog box, on the Tools menu, click **Signal Probe Pins**.

The status of each Signal Probe pin appears in the Change Manager window. If the Change Manager window is not visible at the bottom of your GUI, from the View menu, point to **Utility Windows** and click **Change Manager**.

Figure 40. Change Manager Window with Signal Probe Pins



To view the timing results of each successfully routed Signal Probe pin, on the Processing menu, point to **Start** and click **Start Timing Analysis**.

Related Information

Engineering Change Management with the Chip Planner

4.1.7. What a Signal Probe Compilation Does

After a full compilation, you can start a Signal Probe compilation either manually or automatically. A Signal Probe compilation performs the following functions:

- Validates Signal Probe pins
- Validates your specified Signal Probe sources
- Adds registers into Signal Probe paths, if applicable
- Attempts to route from Signal Probe sources through registers to Signal Probe pins

To run the Signal Probe compilation immediately after a full compilation, on the Tools menu, click **Signal Probe Pins**. In the **Signal Probe Pins** dialog box, click **Start Check & Save All Netlist Changes**.

To run a Signal Probe compilation manually after a full compilation, on the Processing menu, point to **Start** and click **Start Signal Probe Compilation**.

Note: You must run the Fitter before a Signal Probe compilation. The Fitter generates a list of all internal nodes that can serve as Signal Probe sources.

Turn the **Signal Probe enable** option on or off in the **Signal Probe Pins** dialog box to enable or disable each Signal Probe pin.

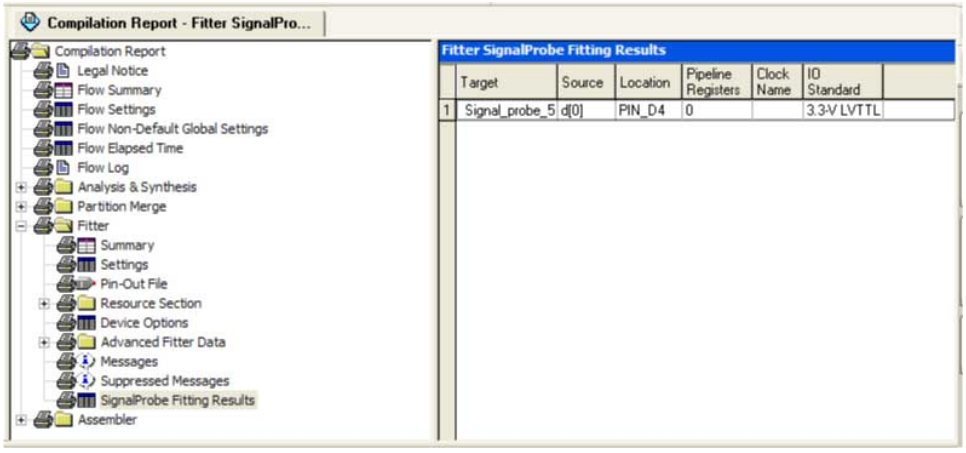
4.1.8. Understanding the Results of a Signal Probe Compilation

After a Signal Probe compilation, the results appear in two sections of the compilation report file. The fitting results and status of each Signal Probe pin appears in the **Signal Probe Fitting Result** screen in the Fitter section of the Compilation Report.

Table 58. Status Values

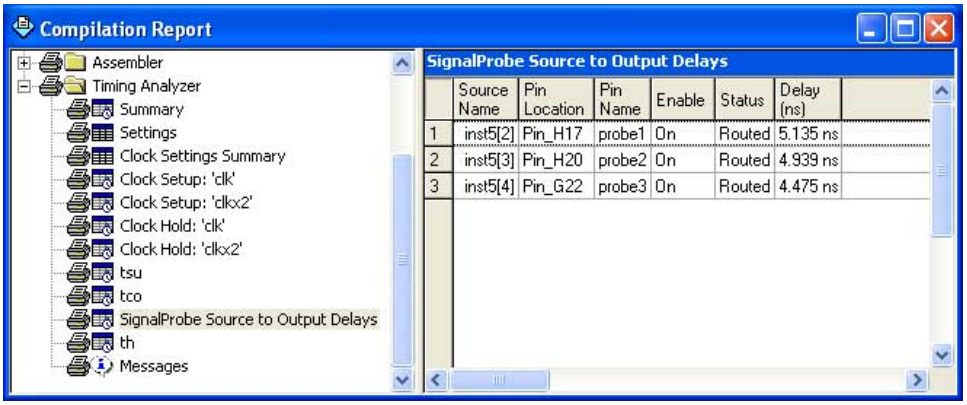
Status	Description
Routed	Connected and routed successfully
Not Routed	Not enabled
Failed to Route	Failed routing during last Signal Probe compilation
Need to Compile	Assignment changed since last Signal Probe compilation

Figure 41. Signal Probe Fitting Results Page in the Compilation Report Window



The **Signal Probe source to output delays** screen in the Timing Analysis section of the Compilation Report displays the timing results of each successfully routed Signal Probe pin.

Figure 42. Signal Probe Source to Output Delays Page in the Compilation Report Window



Note: After a Signal Probe compilation, the processing screen of the Messages window also provides the results for each Signal Probe pin and displays slack information for each successfully routed Signal Probe pin.

4.1.8.1. Analyzing Signal Probe Routing Failures

A Signal Probe compilation can fail for any of the following reasons:

- **Route unavailable**—the Signal Probe compilation failed to find a route from the Signal Probe source to the Signal Probe pin because of routing congestion.
- **Invalid or nonexistent Signal Probe source**—you entered a Signal Probe source that does not exist or is invalid.
- **Unusable output pin**—the output pin selected is found to be unusable.

Routing failures can occur if the Signal Probe pin's I/O standard conflicts with other I/O standards in the same I/O bank.

If routing congestion prevents a successful Signal Probe compilation, you can allow the compiler to modify routing to the specified Signal Probe source. On the Tools menu, click **Signal Probe Pins** and turn on **Modify latest fitting results during Signal Probe compilation**. This setting allows the Fitter to modify existing routing channels used by your design.

Note: Turning on **Modify latest fitting results during Signal Probe compilation** can change the performance of your design.

4.2. Scripting Support

You can also run some 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:

```
quartus_sh --qhelp
```

Note: The Tcl commands in this section are part of the `::quartus::chip_planner` Intel Quartus Prime Tcl API. Source or include the `::quartus::chip_planner` Tcl package in your scripts to make these commands available.

Related Information

- Tcl Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*
- Command Line Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*

4.2.1. Making a Signal Probe Pin

To make a Signal Probe pin, type the following command:

```
make_sp [-h | -help] [-long_help] [-clk <clk>] [-io_std <io_std>] \
-loc <loc> -pin_name <pin name> [-regs <regs>] [-reset <reset>] \
-src_name <source name>
```

4.2.2. Deleting a Signal Probe Pin

To delete a Signal Probe pin, type the following Tcl command:

```
delete_sp [-h | -help] [-long_help] -pin_name <pin name>
```

4.2.3. Enabling a Signal Probe Pin

To enable a Signal Probe pin, type the following Tcl command:

```
enable_sp [-h | -help] [-long_help] -pin_name <pin name>
```

4.2.4. Disabling a Signal Probe Pin

To disable a Signal Probe pin, type the following Tcl command:

```
disable_sp [-h | -help] [-long_help] -pin_name <pin name>
```

4.2.5. Performing a Signal Probe Compilation

To perform a Signal Probe compilation, type the following command:

```
quartus_sh --flow signalprobe <project name>
```

4.2.5.1. Script Example

The example shows a script that creates a Signal Probe pin called `sp1` and connects the `sp1` pin to source node `reg1` in a project that was already compiled.

Creating a Signal Probe Pin Called sp1

```
package require ::quartus::chip_planner
project_open project
read_netlist
make_sp -pin_name sp1 -src_name reg1
check_netlist_and_save
project_close
```

4.2.6. Reserving Signal Probe Pins

To reserve a Signal Probe pin, add the commands shown in the example to the Intel Quartus Prime Settings File (`.qsf`) for your project.

Reserving a Signal Probe Pin

```
set_location_assignment <location> -to <Signal Probe pin name>
set_instance_assignment -name RESERVE_PIN \
"AS SIGNALPROBE OUTPUT" -to <Signal Probe pin name>
```

Valid locations are pin location names, such as `Pin_A3`.

4.2.6.1. Common Problems When Reserving a Signal Probe Pin

If you cannot reserve a Signal Probe pin in the Intel Quartus Prime software, it is likely that one of the following is true:

- You have selected multiple pins.
- A compilation is running in the background. Wait until the compilation is complete before reserving the pin.
- You have the Intel Quartus Prime Lite Edition software, in which the Signal Probe feature is not enabled by default.
- You have not set the pin reserve type to **As Signal Probe Output**. To reserve a pin, on the Assignments menu, in the **Assign Pins** dialog box, select **As Signal Probe Output**.
- The pin is reserved from a previous compilation. During a compilation, the Intel Quartus Prime software reserves each pin on the targeted device. If you end the Intel Quartus Prime process during a compilation, for example, with the **Windows Task Manager End Process** command or the UNIX `kill` command, perform a full recompilation before reserving pins as Signal Probe outputs.
- The pin does not support the Signal Probe feature. Select another pin.
- The current device family does not support the Signal Probe feature.

4.2.7. Adding Signal Probe Sources

- To assign the node name to a Signal Probe pin, type the following Tcl command:

```
set_instance_assignment -name SIGNALPROBE_SOURCE <node name> \
                        -to <signalprobe pin name>
```

- To turn off individual Signal Probe pins, specify OFF instead of ON with the following command:

```
set_instance_assignment -name SIGNALPROBE_ENABLE ON \
                -to <Signal Probe pin name>
```

4.2.8. Assigning I/O Standards

- To assign an I/O standard to a pin, type the following Tcl command:

```
set_instance_assignment -name IO_STANDARD <I/O standard> -to <Signal Probe
pin name>
```

Related Information

I/O Standards Definition

4.2.9. Adding Registers for Pipelining

To add registers for pipelining, type the following Tcl command:

```
set_instance_assignment -name SIGNALPROBE_CLOCK <clock name> \
-to <Signal Probe pin name>

set_instance_assignment -name SIGNALPROBE_NUM_REGISTERS <number of registers> \
-to <Signal Probe pin name>
```

4.2.10. Running Signal Probe Immediately After a Full Compilation

To run Signal Probe immediately after a full compilation, type the following Tcl command:

```
set_global_assignment -name SIGNALPROBE_DURING_NORMAL_COMPILATION ON
```

4.2.11. Running Signal Probe Manually

To run Signal Probe as part of a scripted flow using Tcl, use the following in your script:

```
execute_flow -signalprobe
```

To perform a Signal Probe compilation interactively at a command prompt, type the following command:

```
quartus_sh_fit --flow signalprobe <project name>
```

4.2.12. Enabling or Disabling All Signal Probe Routing

Use the Tcl command in the example to turn on or turn off Signal Probe routing. When using this command, to turn Signal Probe routing on, specify ON. To turn Signal Probe routing off, specify OFF.

Turning Signal Probe On or Off with Tcl Commands

```
set spe [get_all_assignments -name SIGNALPROBE_ENABLE] \
foreach_in_collection asgn $spe {
   set signalprobe_pin_name [lindex $asgn 2]
   set_instance_assignment -name SIGNALPROBE_ENABLE \
-to $signalprobe_pin_name <ON|OFF> }
```

4.2.13. Allowing Signal Probe to Modify Fitting Results

To turn on **Modify latest fitting results**, type the following Tcl command:

```
set_global_assignment -name SIGNALPROBE_ALLOW_OVERUSE ON
```

4.3. Quick Design Debugging Using Signal Probe Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	Dita conversion.
May 2013	13.0.0	Changed sequence of flow to clarify that you need to perform a full compilation before reserving Signal Probe pins. Affected sections are "Debugging Using the Signal Probe Feature" on page 12–1 and "Reserving Signal Probe Pins" on page 12–2. Moved "Performing a Full Compilation" on page 12–2 before "Reserving Signal Probe Pins" on page 12–2.

continued...

Document Version	Intel Quartus Prime Version	Changes
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	• Revised for new UI. • Removed section Signal Probe ECO flows • Removed support for Signal Probe pin preservation when recompiling with incremental compilation turned on. • Removed outdated FAQ section. • Added links to Quartus II Help for procedural content.
November 2009	9.1.0	• Removed all references and procedures for APEX devices. • Style changes.
March 2009	9.0.0	• Removed the "Generate the Programming File" section • Removed unnecessary screenshots • Minor editorial updates
November 2008	8.1.0	• Modified description for preserving Signal Probe connections when using Incremental Compilation • Added plausible scenarios where Signal Probe connections are not reserved in the design
May 2008	8.0.0	• Added "Arria GX" to the list of supported devices • Removed the "On-Chip Debugging Tool Comparison" and replaced with a reference to the Section V Overview on page 13–1 • Added hyperlinks to referenced documents throughout the chapter • Minor editorial updates

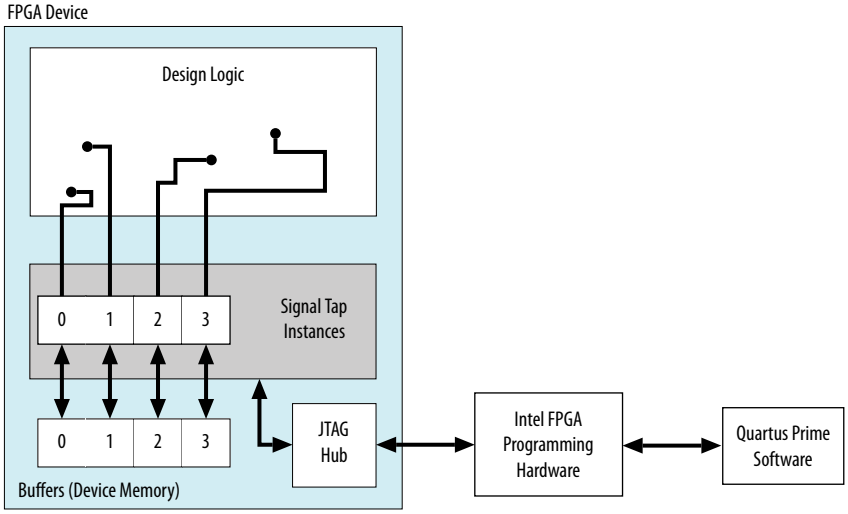
5. Design Debugging with the Signal Tap Logic Analyzer

5.1. The Signal Tap Logic Analyzer

The Signal Tap Logic Analyzer captures and displays real-time signal behavior in an FPGA design, allowing to examine the behavior of internal signals during normal device operation without the need for extra I/O pins or external lab equipment.

To facilitate the debugging process, you can save the captured data in device memory for later analysis. You can also filter data that is not relevant for debug by defining custom trigger-condition logic. The Signal Tap Logic Analyzer supports the highest number of channels, largest sample depth, and fastest clock speeds of any logic analyzer in the programmable logic market.

Figure 43. Signal Tap Logic Analyzer Block Diagram



Note to figure:

1. This diagram assumes that you compiled the Signal Tap Logic Analyzer with the design as a separate design partition using the Intel Quartus Prime incremental compilation feature. If you do not use incremental compilation, the Compiler integrates the Signal Tap logic with the design.

The Signal Tap Logic Analyzer is available as a stand-alone package or with a software subscription.

To take advantage of faster compile times when making changes to the Signal Tap Logic Analyzer, knowledge of the Intel Quartus Prime incremental compilation feature is helpful.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

5.1.1. Hardware and Software Requirements

You need the following hardware and software to perform logic analysis with the Signal Tap Logic Analyzer:

- Signal Tap Logic Analyzer

  The following software includes the Signal Tap Logic Analyzer:

  — Intel Quartus Prime Design Software

  — Intel Quartus Prime Lite Edition

  Alternatively, use the Signal Tap Logic Analyzer standalone software and standalone Programmer software.
- Download/upload cable
- Intel development kit or your design board with JTAG connection to device under test

Note: The Intel Quartus Prime Lite Edition software does not support incremental compilation integration with the Signal Tap Logic Analyzer.

During data acquisition, the memory blocks in the device store the captured data, and then transfer the data to the logic analyzer over a JTAG communication cable, such as or Intel FPGA Download Cable.

5.1.1.1. Opening the Standalone Signal Tap Logic Analyzer GUI

1. To open a new Signal Tap through the command-line, type:

```
quartus_stpw <stp_file.stp>
```

5.1.2. Signal Tap Logic Analyzer Features and Benefits

Feature	Benefit
Quick access toolbar	Provides single-click operation of commonly-used menu items. You can hover over the icons to see tool tips.
Multiple logic analyzers in a single device	Allows you to capture data from multiple clock domains in a design at the same time.
Multiple logic analyzers in multiple devices in a single JTAG chain	Allows you to capture data simultaneously from multiple devices in a JTAG chain.
Nios II plug-in support	Allows you to specify nodes, triggers, and signal mnemonics for IP, such as the Nios II processor.
Up to 10 basic, comparison, or advanced trigger conditions for each analyzer instance	Allows you to send complex data capture commands to the logic analyzer, providing greater accuracy and problem isolation.
Power-up trigger	Captures signal data for triggers that occur after device programming, but before manually starting the logic analyzer.
Custom trigger HDL object	You can code your own trigger in Verilog HDL or VHDL and tap specific instances of modules located anywhere in the hierarchy of your design, without needing to manually route all the necessary connections. This simplifies the process of tapping nodes spread out across your design.
State-based triggering flow	Enables you to organize your triggering conditions to precisely define what your logic analyzer captures.

continued...

Feature	Benefit
Incremental compilation	Allows you to modify the signals and triggers that the Signal Tap Logic Analyzer monitors without performing a full compilation, saving time.
Incremental route with rapid recompile	Allows you to manually allocate trigger input, data input, storage qualifier input, and node count, and perform a full compilation to include the Signal Tap Logic Analyzer in your design. Then, you can selectively connect, disconnect, and swap to different nodes in your design. Use Rapid Recompile to perform incremental routing and gain a 2-4x speedup over the initial full compilation.
Flexible buffer acquisition modes	The buffer acquisition control allows you to precisely control the data that is written into the acquisition buffer. Both segmented buffers and non-segmented buffers with storage qualification allow you to discard data samples that are not relevant to the debugging of your design.
MATLAB integration with included MEX function	Collects the data the Signal Tap Logic Analyzer captures into a MATLAB integer matrix.
Up to 2,048 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.
Resource usage estimator	Provides an estimate of logic and memory device resources that the Signal Tap Logic Analyzer configurations use.
No additional cost	Intel Quartus Prime subscription and the Intel Quartus Prime Lite Edition include the Signal Tap Logic Analyzer.
Compatibility with other on-chip debugging utilities	You can use the Signal Tap Logic Analyzer in tandem with any JTAG-based on-chip debugging tool, such as an In-System Memory Content editor, allowing you to change signal values in real-time while you are running an analysis with the Signal Tap Logic Analyzer.
Floating-Point Display Format	To enable, click **Edit ➤ Bus Display Format ➤ Floating-point** Supports: • Single-precision floating-point format **IEEE754 Single (32-bit)**. • Double-precision floating-point format **IEEE754 Double (64-bit)**.

Related Information

System Debugging Tools Overview on page 7

5.1.3. Backward Compatibility with Previous Versions of Intel Quartus Prime Software

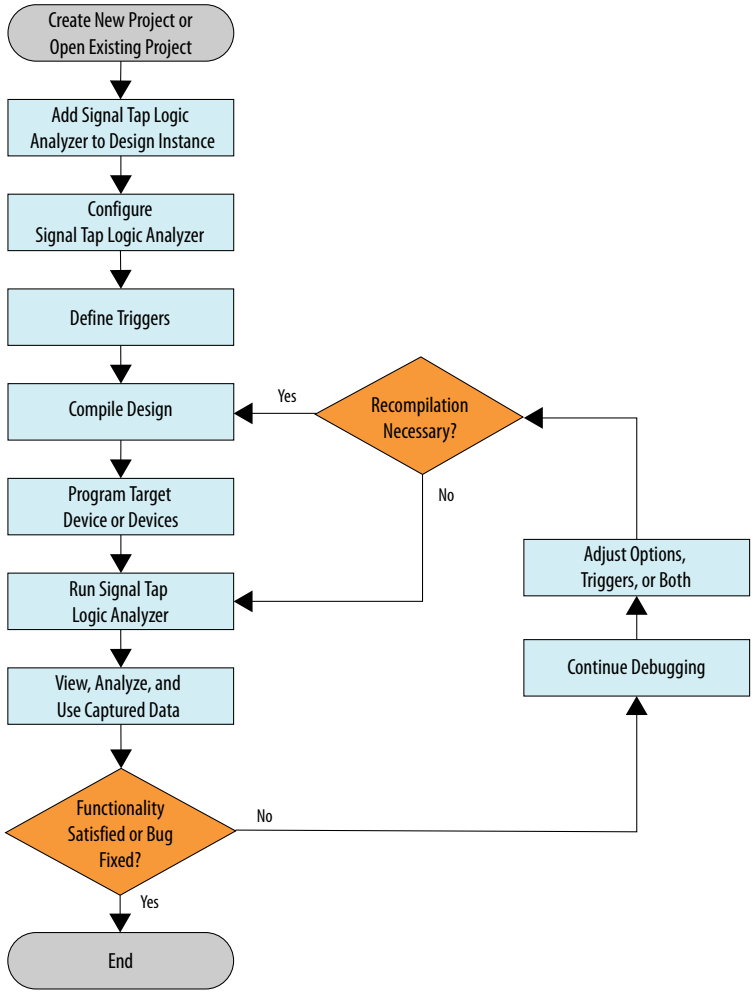
When you open an `.stp` file created in a previous version of Intel Quartus Prime software in a newer version of the software, the `.stp` file cannot be opened in a previous version of the Intel Quartus Prime software.

If you have a Intel Quartus Prime project file from a previous version of the software, you may have to update the `.stp` configuration file to recompile the project. You can update the configuration file by opening the Signal Tap Logic Analyzer. If you need to update your configuration, a prompt appears asking if you want to update the `.stp` to match the current version of the Intel Quartus Prime software.

5.2. Signal Tap Logic Analyzer Task Flow Overview

To use the Signal Tap Logic Analyzer to debug your design, you perform a number of tasks to add, configure, and run the logic analyzer.

Figure 44. Signal Tap Logic Analyzer Task Flow



5.2.1. Add the Signal Tap Logic Analyzer to Your Design

Create an `.stp` or create a parameterized HDL instance representation of the logic analyzer using the IP Catalog and parameter editor. If you want to monitor multiple clock domains simultaneously, add additional instances of the logic analyzer to your design, limited only by the available resources in your device.

5.2.2. Configure the Signal Tap Logic Analyzer

After you add the Signal Tap Logic Analyzer to your design, configure the logic analyzer to monitor the signals you want.

You can add signals manually or use a plug-in, such as the Nios II processor plug-in, to add entire sets of associated signals for a particular IP.

Specify settings for the data capture buffer, such as its size, the method in which the Signal Tap Logic Analyzer captures and stores the data. If your device supports memory type selection, you can specify the memory type to use for the buffer.

Related Information

Configuring the Signal Tap Logic Analyzer on page 151

5.2.3. Define Trigger Conditions

By default, the Signal Tap Logic Analyzer captures data continuously while the logic analyzer is running. To capture and store specific signal data you can set up triggers that specify conditions to start or stop capturing data.

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. Power-Up Triggers allow you to capture data from trigger events occurring immediately after the device enters user-mode after configuration.

Related Information

Defining Triggers on page 169

5.2.4. Compile the Design

Once you configure the `.stp` file and define trigger conditions, compile your project including the logic analyzer in your design.

Note: Because you may need to change monitored signal nodes or adjust trigger settings frequently during debugging, Intel FPGA recommends that you use the incremental compilation feature built into the Signal Tap Logic Analyzer, along with Intel Quartus Prime incremental compilation, to reduce recompile times. You can also use Incremental Route with Rapid Recompile to reduce recompile times.

Related Information

Compiling the Design on page 193

5.2.5. Program the Target Device or Devices

When you debug a design with the Signal Tap Logic Analyzer, you can program a target device directly from the `.stp` without using the Intel Quartus Prime Programmer. You can also program multiple devices with different designs and simultaneously debug them.

Related Information

- Program the Target Device or Devices on page 199
- Manage Multiple Signal Tap Files and Configurations on page 167

5.2.6. Run the Signal Tap Logic Analyzer

In normal device operation, you control the logic analyzer through the JTAG connection, specifying when to start looking for trigger conditions to begin capturing data. With Runtime or Power-Up Triggers, read and transfer the captured data from the on-chip buffer to the `.stp` for analysis.

Related Information

Running the Signal Tap Logic Analyzer on page 200

5.2.7. View, Analyze, and Use Captured Data

The data you capture and read into the `.stp` file is available for analysis and debugging. 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.

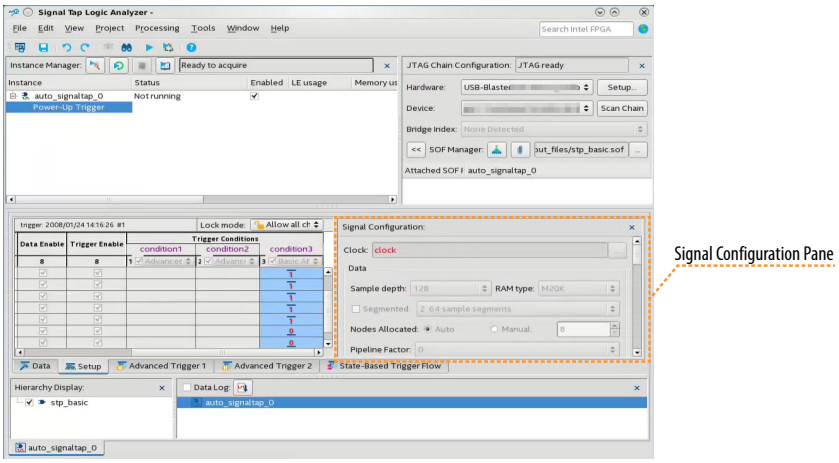
Related Information

View, Analyze, and Use Captured Data on page 204

5.3. Configuring the Signal Tap Logic Analyzer

You configure instances of the Signal Tap Logic Analyzer in the **Signal Configuration** pane of the **Signal Tap Logic Analyzer** window.

Figure 45. Signal Tap Logic Analyzer Signal Configuration Pane



5.3.1. Assigning an Acquisition Clock

To control how the Signal Tap Logic Analyzer acquires data you must assign a clock signal. The logic analyzer samples data on every positive (rising) edge of the acquisition clock. The logic analyzer does not support sampling on the negative (falling) edge of the acquisition clock.

You can use any signal in your design as the acquisition clock. However, for best results in data acquisition, use a global, non-gated clock that is synchronous to the signals under test. Using a gated clock as your acquisition clock can result in unexpected data that does not accurately reflect the behavior of your design. The Intel Quartus Prime static timing analysis tools show the maximum acquisition clock frequency at which you can run your design. To find the maximum frequency of the logic analyzer clock, refer to the Timing Analysis section of the Compilation Report.

Caution: Be careful when using a recovered clock from a transceiver as an acquisition clock for the Signal Tap Logic Analyzer. A recovered clock can cause incorrect or unexpected behavior, particularly when the transceiver recovered clock is the acquisition clock with the power-up trigger feature.

If you do not assign an acquisition clock in the Signal Tap Logic Analyzer Editor, Intel Quartus Prime software automatically creates a clock pin called `auto_stp_external_clk`. You must make a pin assignment to this pin, and make sure that a clock signal in your design drives the acquisition clock.

Related Information

- Adding Signals with a Plug-In on page 155
- Managing Device I/O Pins
  In *Intel Quartus Prime Standard Edition: Design Constraints*

5.3.2. Adding Signals to the Signal Tap File

Add the signals that you want to monitor to the `.stp` node list. You can also select signals to define triggers. You can assign the following two signal types:

- **Pre-synthesis**—These signals exist after design elaboration, but before any synthesis optimizations are done. This set of signals must reflect your Register Transfer Level (RTL) signals.
- **Post-fitting**—These signals exist after physical synthesis optimizations and place-and-route.

Note: If you are not using incremental compilation, add only pre-synthesis signals to the `.stp`. Using pre-synthesis helps when you want to add a new node after you change a design. After you perform Analysis and Elaboration, the source file changes appear in the Node Finder.

Intel Quartus Prime software does not limit the number of signals available for monitoring in the Signal Tap window waveform display. However, the number of channels available 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 tap signals if a routing resource (row or column interconnects) exists to route the connection to the Signal Tap instance. For example, you cannot tap 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 tap the signal that is driving a logic array block (LAB) from an IOE, or, for output pins, you can tap the signal from the LAB that is driving an IOE.

Related Information

Faster Compilations with Intel Quartus Prime Incremental Compilation on page 194

5.3.2.1. Pre-Synthesis Signals

When you add pre-synthesis signals, make all connections to the Signal Tap Logic Analyzer before synthesis. The Compiler allocates logic and routing resources to make the connection as if you changed your design files. For signals driving to and from IOEs, pre-synthesis signal names coincide with the pin's signal names.

5.3.2.2. Post-Fit Signals

When you tap post-fit signals, you are connecting to actual atoms in the post-fit netlist. You can only tap signals that exist in the post-fit netlist, and existing routing resources must be available.

In the case of post-fit output signals, tap 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 tap, the signal from the atom may be inverted. You can check this by locating the signal in either the Resource Property Editor or the Technology Map Viewer. You can also use the Technology Map viewer and the Resource Property Editor to find post-fit node names.

Related Information

Design Flow with the Netlist Viewers

5.3.2.2.1. Assigning Data Signals with the Technology Map Viewer

The Technology Map Viewer allows you to to add post-fit signal.

1. After compilation, launch the Technology Map Viewer from the **Intel Quartus Prime** software, by clicking **Tools ➤ Netlist Viewers ➤ Technology Map Viewer (Post-Fitting)**.
2. Find the node that you want to tap.
3. Copy the node to either the active `.stp` for the design or a new `.stp`.

5.3.2.3. Signal Preservation

The Intel Quartus Prime software provides synthesis attributes that prevent the Compiler from performing optimizations on specific signals, allowing them to persist into the post-fit netlist.

The Intel Quartus Prime software optimizes the RTL signals during synthesis and place-and-route. RTL signal names may not appear in the post-fit netlist after optimizations.

The optimization attributes are:

- `keep`—Prevents removal of combinational signals during optimization.
- `preserve`—Prevents removal of registers during optimization.

However, preserving attributes can increase device resource utilization or decrease timing performance.

Note: These processing results can cause problems with the incremental compilation flow in Signal Tap Logic Analyzer. Because you can only add post-fitting signals to the Signal Tap Logic Analyzer in partitions of type **post-fit**, RTL signals that you want to monitor may not be available, preventing their use. To avoid this issue, add synthesis attributes that preserve signals during synthesis and place-and-route.

Preserving nodes is often necessary when you add groups of signals for an IP with a plug-in. If you are debugging an encrypted IP core, such as the Nios II CPU, you might need to preserve nodes from the core to keep available for debugging with the Signal Tap Logic Analyzer.

In incremental compilation flows, pre-synthesis nodes may not be connected to the Signal Tap Logic Analyzer for post-fit partitions. Signal Tap issues a critical warning for all pre-synthesis node names that do not exist in the post-fit netlist.

5.3.2.4. Node List Signal Use Options

When you add a signal to the node list, you can select options that specify how the logic analyzer uses the signal.

To prevent a signal from triggering the analysis, disable the signal's **Trigger Enable** option in the `.stp` file. This option is useful when you only want to see the signal's captured data.

You can turn off the ability to view data for a signal by disabling the **Data Enable** column in the `.stp` file. This option is useful when you want to trigger on a signal, but have no interest in viewing that signal's data.

Related Information

Defining Triggers on page 169

5.3.2.4.1. Disabling and Enabling a Signal Tap Instance

Disable and enable Signal Tap instances in the **Instance Manager** pane. Physically adding or removing instances requires recompilation after disabling and enabling a Signal Tap instance.

5.3.2.5. Signals Unavailable for Signal Tap Debugging

Not all the post-fitting signals in your design are available in the **Signal Tap: post-fitting filter** in the **Node Finder** dialog box.

You cannot tap any of the following signal types:

- **Post-fit output pins**—You cannot tap a post-fit output pin directly. To make an output signal visible, tap the register or buffer that drives the output pin. This includes pins defined as bidirectional.
- **Signals that are part of a carry chain**—You cannot tap 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 tap the JTAG control (`TCK`, `TDI`, `TDO`, and `TMS`) signals.

- **ALTGXB IP core**—You cannot directly tap any ports of an ALTGXB instantiation.
- **LVDS**—You cannot tap the data output from a serializer/deserializer (SERDES) block.
- **DQ**, **DQS Signals**—You cannot directly tap the DQ or DQS signals in a DDR/DDRII design.

5.3.3. Adding Signals with a Plug-In

Instead of adding individual or grouped signals through the **Node Finder**, you can use a plug-in to add groups of relevant signals of a particular type of IP. Besides easy signal addition, plug-ins provide features such as pre-designed mnemonic tables, useful for trigger creation and data viewing, as well as the ability to disassemble code in captured data. The Signal Tap Logic Analyzer comes with one plug-in for the Nios II processor.

The Nios II plug-in, for example, 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 signals to the `.stp` file using a plug-in, perform the following steps after running Analysis and Elaboration on your design:

1. To ensure that all the required signals are available, in the Intel Quartus Prime software, click **Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**. Turn on **Create debugging nodes for IP cores**.
   All the signals included in the plug-in are added to the node list.
2. Right-click the node list. On the **Add Nodes with Plug-In** submenu, select the plug-in you want to use, such as the included plug-in named **Nios II**.
   The **Select Hierarchy Level** dialog box appears showing the IP hierarchy of your design. If the IP for the selected plug-in does not exist in your design, a message informs you that you cannot use the selected plug-in.
3. 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.
4. 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**.

Related Information

- Defining Triggers on page 169
- View, Analyze, and Use Captured Data on page 151

5.3.4. Adding Finite State Machine State Encoding Registers

Finding the signals to debug finite state machines (FSM) can be challenging. Finding nodes from the post-fit netlist may be impossible, since the Compiler may change or optimize away FSM encoding signals. To find and map FSM signal values to the state names that you specified in your HDL, you must perform an additional step.

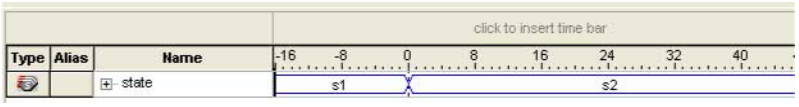
The Signal Tap Logic Analyzer can detect FSMs in your compiled design. The configuration automatically tracks the FSM state signals as well as state encoding through the compilation process.

To add all the FSM state signals to your logic analyzer with a single command Shortcut menu commands allow you .

For each FSM added to your Signal Tap configuration, the FSM debugging feature adds a mnemonic table to map the signal values to the state enumeration that you provided in your source code. The mnemonic tables enable you to visualize state machine transitions in the waveform viewer. The FSM debugging feature supports adding FSM signals from both the pre-synthesis and post-fit netlists.

Figure 46. Decoded FSM Mnemonics

The waveform viewer with decoded signal values from a state machine added with the FSM debugging feature.



Related Information

State Machine HDL Guidelines

5.3.4.1. Modify and Restore Mnemonic Tables for State Machines

Edit any mnemonic table using the **Mnemonic Table Setup** dialog box. When you add FSM state signals via the FSM debugging feature, the Signal Tap Logic Analyzer GUI creates a mnemonic table using the format *<StateSignalName>*`_table`, where **StateSignalName** is the name of the state signals that you have declared in your RTL.

If you want to restore a mnemonic table that was modified, right-click anywhere in the node list window and select **Recreate State Machine Mnemonics**. By default, restoring a mnemonic table overwrites the existing mnemonic table that you modified. To restore a FSM mnemonic table to a new record, turn off **Overwrite existing mnemonic table** in the **Recreate State Machine Mnemonics** dialog box.

Note: If you have added or deleted a signal from the FSM state signal group from within the setup tab, delete the modified register group and add the FSM signals back again.

Related Information

Creating Mnemonics for Bit Patterns on page 207

5.3.4.2. Additional Considerations for State Machines in Signal Tap

- The Signal Tap configuration GUI recognizes state machines from your design only if you use Intel Quartus Prime Integrated Synthesis. Conversely, the state machine debugging feature is not able to track the FSM signals or state encoding if you use other EDA synthesis tools.
- If you add post-fit FSM signals, the Signal Tap Logic Analyzer FSM debug feature may not track all optimization changes that are a part of the compilation process.
- If the following two specific optimizations are enabled, the Signal Tap FSM debug feature may not list mnemonic tables for state machines in the design:
  - If you enabled the **Physical Synthesis** optimization, state registers may be resource balanced (register retiming) to improve f_{MAX}. The FSM debug feature does not list post-fit FSM state registers if register retiming occurs.
  - The FSM debugging feature does not list state signals that the Compiler packed into RAM and DSP blocks during synthesis or Fitter optimizations.
- You can still use the FSM debugging feature to add pre-synthesis state signals.

Related Information

Enabling Physical Synthesis Optimization

5.3.5. Specifying Sample Depth

The **Sample depth** setting specifies the number of samples the Signal Tap Logic Analyzer captures and stores, for each signal in the captured data buffer.

To specify the sample depth:

1. Select the desired number in the **Sample Depth** drop-down menu.

   The sample depth ranges from 0 to 128K.

In cases with limited device memory resources, the design may not be able to compile due to the selected sample buffer size. Try reducing the sample depth to reduce resource usage.

5.3.6. Capture Data to a Specific RAM Type

You have the option to select the RAM type where the Signal Tap Logic Analyzer stores acquisition data. Once 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.

RAM selection allows you to preserve a specific memory block for your design, and allocate another portion of memory for Signal Tap Logic Analyzer 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, you can use MLAB, M512, or M4k blocks for data acquisition and leave M9k blocks for the rest of your design.

To specify the RAM type to use for the Signal Tap Logic Analyzer buffer, go to the **Signal Configuration** pane in the **Signal Tap** window, and select one **Ram type** from the drop-down menu.

Use this feature only when the acquired data is smaller than the available memory of the RAM type that you selected. The amount of data appears in the Signal Tap resource estimator.

5.3.7. Select the Buffer Acquisition Mode

When you specify how the logic analyzer organizes the captured data buffer, you can potentially reduce the amount of memory that Signal Tap requires for data acquisition.

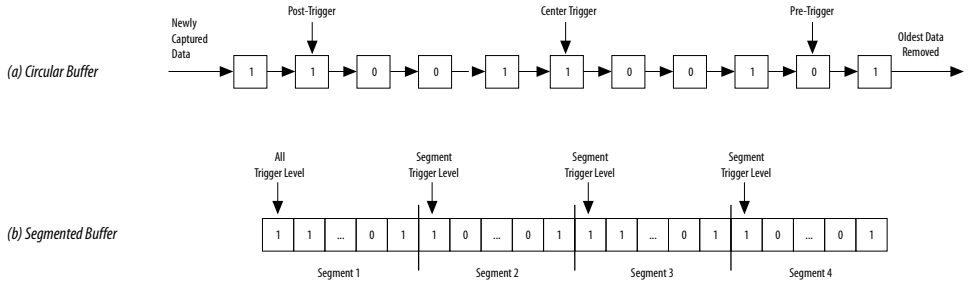
There are two types of acquisition buffer within the Signal Tap Logic Analyzer—a non-segmented (or circular) buffer and a segmented buffer.

- With a non-segmented buffer, the Signal Tap Logic Analyzer treats entire memory space as a single FIFO, continuously filling the buffer until the logic analyzer reaches a defined set of trigger conditions.
- With a 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 47. 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. Refer to *Specify Trigger Position* for more details.

Related Information

- Specify Trigger Position on page 190
- Filtering Relevant Samples on page 160

5.3.7.1. 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 most data before the trigger occurs, select **Post trigger position** from the list
- To capture most data after the trigger, 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.

Related Information

Specify Trigger Position on page 190

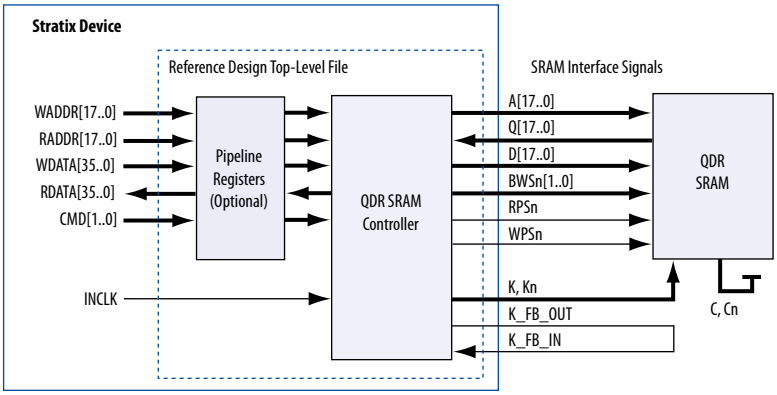
5.3.7.2. Segmented Buffer

In a segmented buffer, the acquisition memory is split into segments of even size, and you define a set of trigger conditions for each segment. 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 segmented buffers, you must use the state-based trigger flow. The figure shows an example of a segmented buffer system.

Figure 48. System that Generates Recurring Events

In this design, you want to ensure that the correct data is written to the SRAM controller by monitoring the RDATA port whenever the address `H'0F0F0F0F` is sent into the RADDR port.



With the buffer acquisition feature. you can monitor multiple read transactions from the SRAM device without running the Signal Tap Logic Analyzer again, because you 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 determine the number of segments to use. In the example in the figure, selecting sixty-four 64-sample segments allows you to capture 64 read cycles.

Related Information

Capturing Data Using Segmented Buffers on page 204

5.3.8. 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.

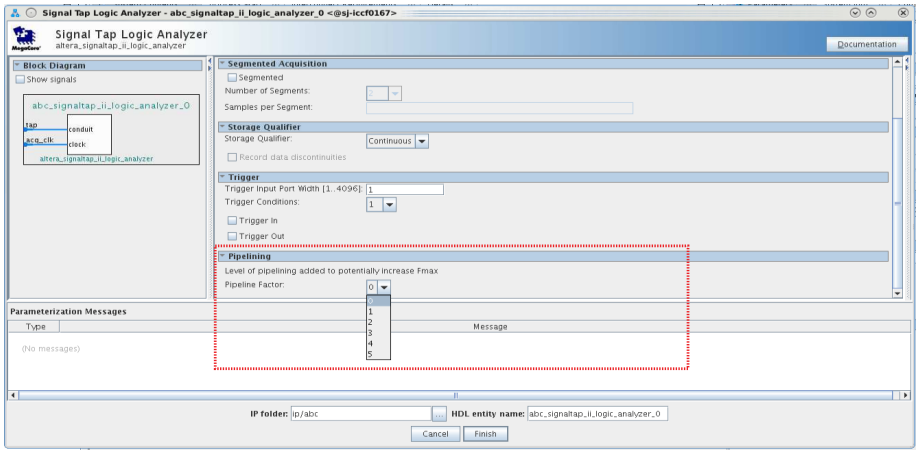
5.3.8.1. Specifying Pipeline Settings from Platform Designer (Standard)

The **Pipeline factor** setting indicates the number of pipeline registers that you can add to boost the f_{MAX} of the Signal Tap Logic Analyzer. You can specify the pipeline factor in the **Signal Configuration** pane. The pipeline factor ranges from 0 to 5, with a default value of 0.

To specify the pipeline factor when you instantiate the Signal Tap Logic Analyzer component from the Platform Designer (Standard) system:

1. Double-click **Signal Tap Logic Analyzer** component in the IP Catalog.
2. Specify the **Pipeline Factor**, along with other parameter values

Figure 49. Specifying the Pipeline Factor from Platform Designer (Standard)



5.3.9. Filtering Relevant Samples

The Storage Qualifier feature allows you to filter out individual samples not relevant to debugging the design.

The Signal Tap Logic Analyzer offers a snapshot in time of the data stored in the acquisition buffers. 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.

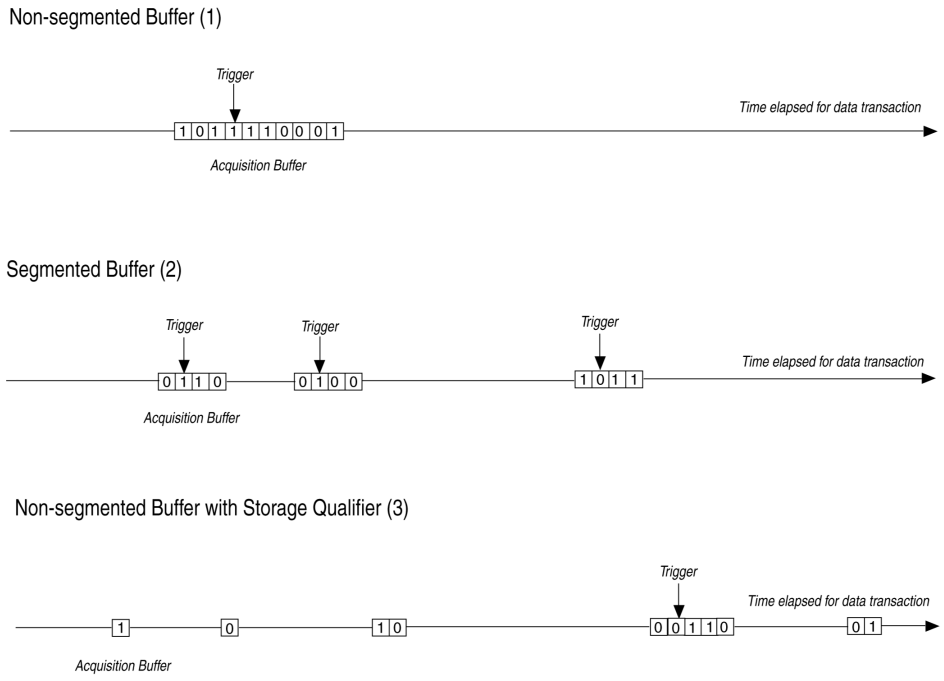
With analysis using acquisition buffers you can capture most functional errors in a chosen signal set, provided adequate trigger conditions and a generous sample depth for the acquisition. However, each data window can have a considerable amount of unnecessary data; for example, long periods of idle 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 50. Data Acquisition Using Different Modes of Controlling the Acquisition Buffer



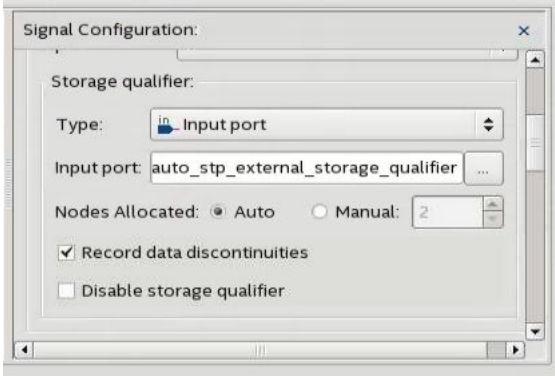
Notes to figure:

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 51. 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.

Related Information

Define Trigger Conditions on page 150

5.3.9.1. Input Port Mode

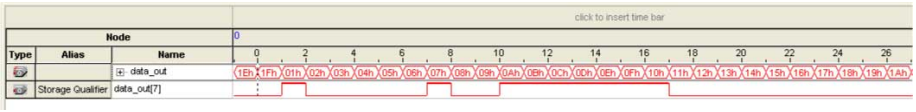
When using the Input port mode, the Signal Tap Logic Analyzer takes any signal from your design as an input. During acquisition, if the signal is high on the clock edge, the Signal Tap Logic Analyzer stores the data in the buffer. If the signal is low on the clock edge, the Logic Analyzer ignores the data sample. If you don't specify an internal node, the Logic Analyzer creates and connects a pin to this input port.

If you are creating a Signal Tap Logic Analyzer instance through an `.stp` file, specify the storage qualifier signal using the input port field located on the **Setup** tab. You must specify this port for your project to compile.

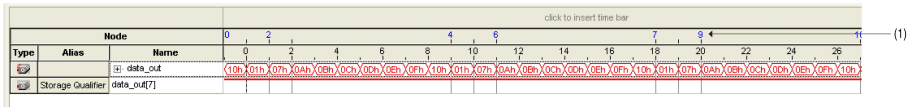
If you use the parameter editor, the storage qualification input port, if specified, appears in the generated instantiation template. You can then connect this port to a signal in your RTL.

Figure 52. Comparing Continuous and Input Port Capture Mode in Data Acquisition of a Recurring Data Pattern

- Continuous Mode:



- Input Port Storage Qualifier:



(1) Markers display samples when the logic analyzer paused a write into acquisition memory. These markers are enabled with the option "Record data discontinuities."

5.3.9.2. 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 53. Transitional Storage Qualifier Setup

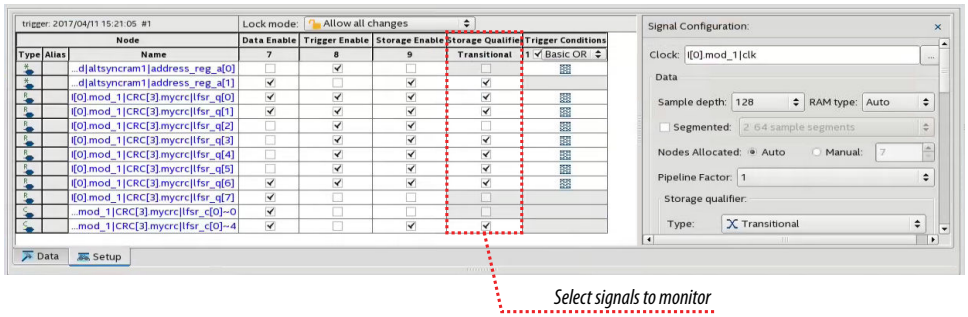
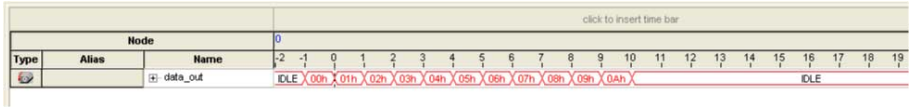
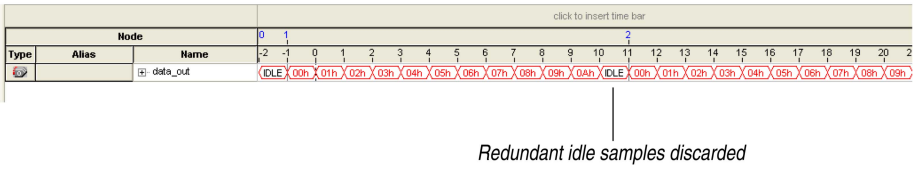


Figure 54. Comparing Continuous and Transitional Capture Mode in Data Acquisition of a Recurring Data Pattern

- Continuous:



- Transitional mode:



5.3.9.3. 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 55. Conditional Storage Qualifier Setup

The figure details the conditional storage qualifier setup in the `.stp` file.

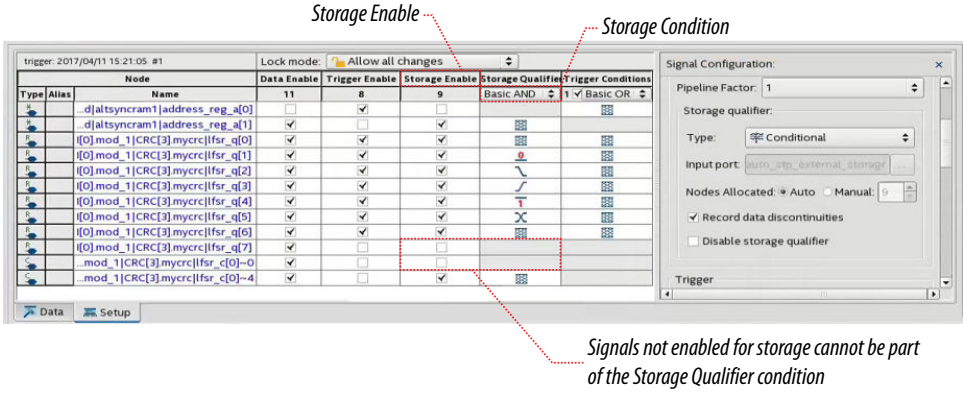
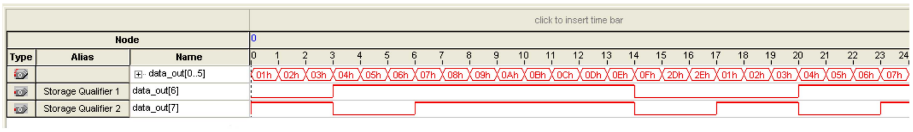


Figure 56. Comparing Continuous and Conditional Capture Mode in Data Acquisition of a Recurring Data Pattern

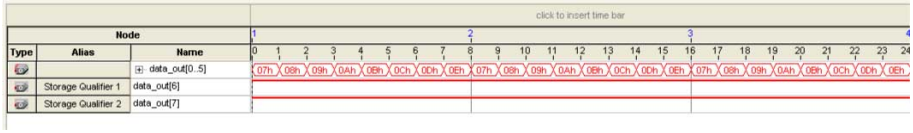
The data pattern is the same in both cases.

- Continuous sampling capture mode:



(1) Storage Qualifier condition is set up to evaluate data_out[6] AND data_out[7].

- Conditional sampling capture mode:



Related Information

- Basic Trigger Conditions on page 169
- Comparison Trigger Conditions on page 170
- Advanced Trigger Conditions on page 172

5.3.9.4. 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, 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.

Figure 57. Start/Stop Mode Storage Qualifier Setup

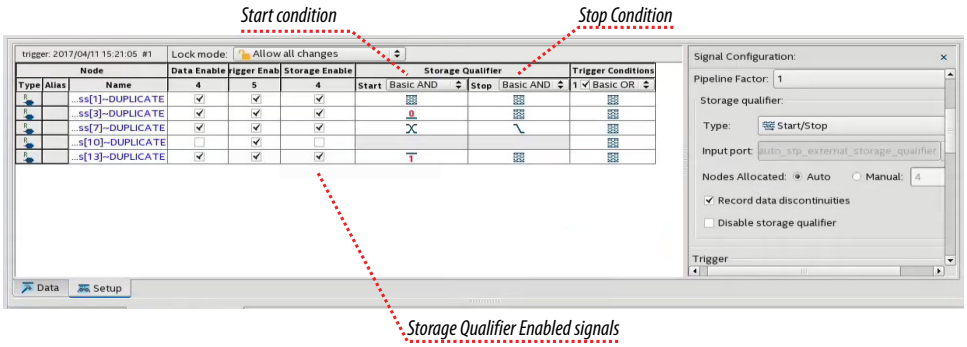
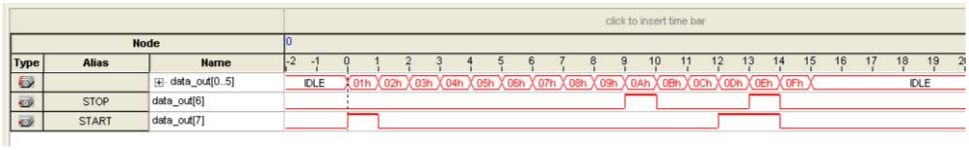
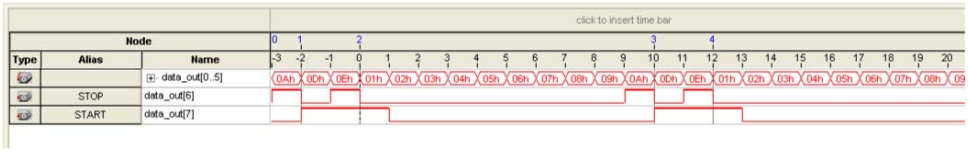


Figure 58. Comparing Continuous and Start/Stop Acquisition Modes for a Recurring Data Pattern

- Continuous Mode:



- Start/Stop Storage Qualifier:



5.3.9.5. State-Based

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 183

5.3.9.6. 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.

5.3.9.7. Disable Storage Qualifier

You can quickly turn off the storage qualifier with the **Disable Storage Qualifier** option, and perform a continuous capture. This option is run-time reconfigurable. Changing storage qualifier mode from the **Type** field requires a recompilation of the project.

Related Information

Runtime Reconfigurable Options on page 201

5.3.10. Manage Multiple Signal Tap Files and Configurations

You can debug different blocks in your design by grouping related monitoring signals. Likewise, 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.

Signal Tap Logic Analyzer allows you to save debug configurations in more than one `.stp` file. Alternatively, you can embed multiple configurations within the same `.stp` file, and use the Data Log as a managing tool.

Note: Each `.stp` file is associated with a programming (`.sof`) file. To function correctly, the settings in the `.stp` file you use at runtime must match Signal Tap settings in the `.sof` file you use to program the device.

Related Information

Ensure Setting Compatibility Between .stp and .sof Files on page 200

5.3.10.1. Data Log Pane

The Data Log pane displays all Signal Tap configurations and data capture results stored within a single `.stp` file.

- To save the current configuration or capture in the Data Log—and `.stp` file, click **Edit ➤ Save to Data Log**. Alternatively, click the **Save to Data Log** icon at the top of the Data Log pane.
- To generate a log entry after every data capture, click **Edit ➤ Enable Data Log**. Alternatively, check 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 59. Data Log Items

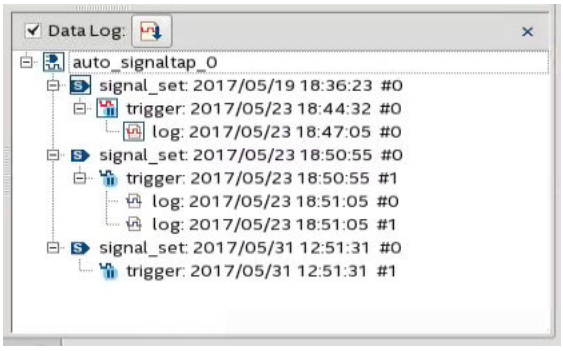
Item	Icon		Contains one or more	Comments
	Unselected	Selected		
Instance			Signal Set	
Signal Set			Trigger	The Signal Set changes whenever you add a new signal to Signal Tap. After a change in the Signal Set, you need to recompile.
Trigger			Capture Log	A trigger changes when you change any trigger condition. These changes do not require recompilation.
Capture Log				

The name on each entry displays the wall-clock time when Signal Tap Logic Analyzer triggered, and the time elapsed from start acquisition to trigger activation. You can rename entries so they make sense to you.

To switch between configurations, double-click an entry in the Data Log. As a result, the **Setup** tab updates to display the active signal list and trigger conditions.

Example 21. Simple Data Log

On this example, the Data Log displays one instance with three signal set configurations.

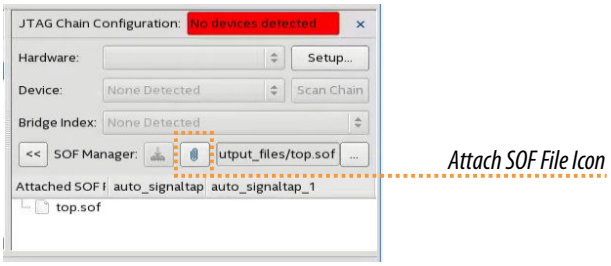


5.3.10.2. SOF Manager

The SOF Manager is in the **JTAG Chain Configuration** pane.

With the SOF Manager you can embed multiple SOFs into one `.stp` file. This action lets you move the `.stp` file to a different location, either on the same computer or across a network, without including the associated `.sof` separately. To embed a new SOF in the `.stp` file, click the **Attach SOF File** icon .

Figure 59. 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` matches the design programmed into the target device.

Related Information

Data Log Pane on page 167

5.4. Defining Triggers

At runtime the Signal Tap Logic Analyzer continuously samples activity from the monitored signals. A trigger activates—that is, the logic analyzer stops and displays the data—when the monitored signals reach the condition or set of conditions that you specify.You specify trigger conditions in the Signal Tap Logic Analyzer **Signal Configuration** pane.

5.4.1. Basic Trigger Conditions

If you select the **Basic AND** or **Basic OR** trigger type, you must specify the trigger pattern for each signal you added in the `.stp`. To specify the trigger pattern, right-click the **Trigger Conditions** column and click the desired pattern. Set the trigger pattern to any of the following conditions:

- **Don't Care**
- **Low**
- **High**
- **Falling Edge**
- **Rising Edge**
- **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.

Related Information

View, Analyze, and Use Captured Data on page 204

5.4.1.1. Using the Basic OR Trigger Condition with Nested Groups

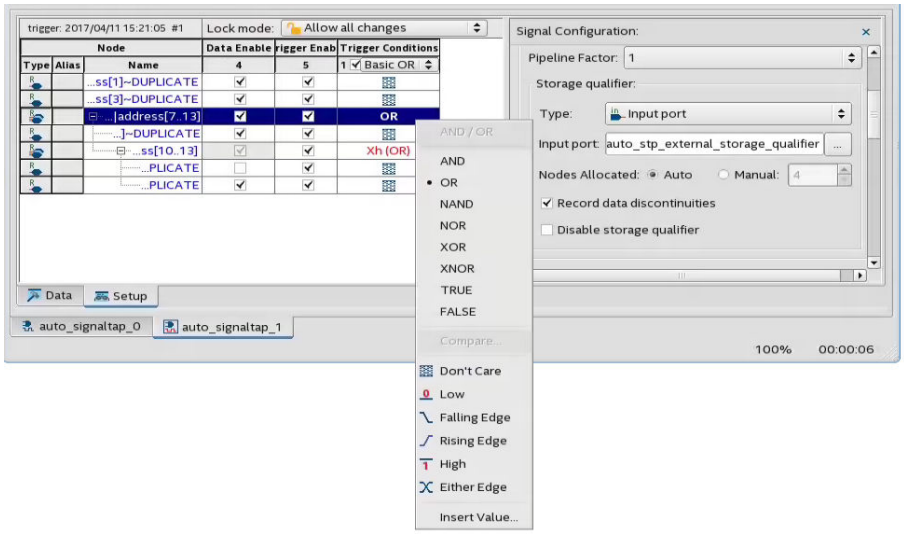
When you specify a set of signals as a nested group (group of groups) with the **Basic OR** trigger type, Signal Tap Logic Analyzer generates an advanced trigger condition. This condition sorts signals within groups to minimize the need to recompile your design. As long as the parent-child relationships of nodes are kept constant, the advanced trigger condition does not change. You can modify the sibling relationships of nodes and not need to recompile your design.

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 60. Applying Trigger Condition to Nested Group



5.4.2. 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:

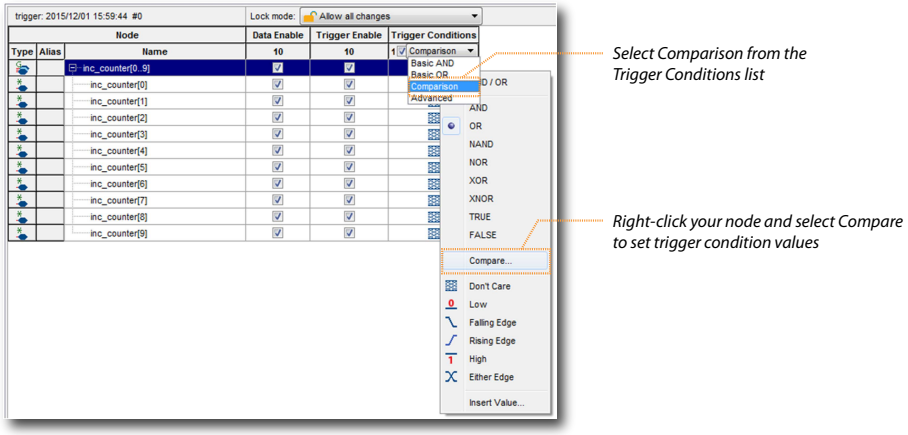
- Apply the **Comparison** trigger only to bus nodes consisting of leaf nodes.
- Do not form sub-groups within a bus node.
- Do not enable or disable individual trigger nodes within a bus node.
- Do not specify comparison values (in case of single-value comparison) or boundary values (in case of interval check) exceeding the selected node's bus-width.

5.4.2.1. 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**.

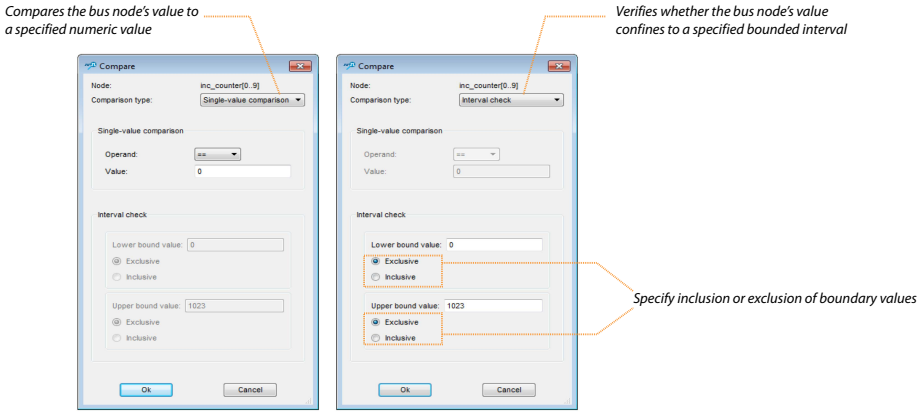
Figure 61. Selecting the Comparison Trigger Condition



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.

   You can also specify if you want to include or exclude the boundary values.

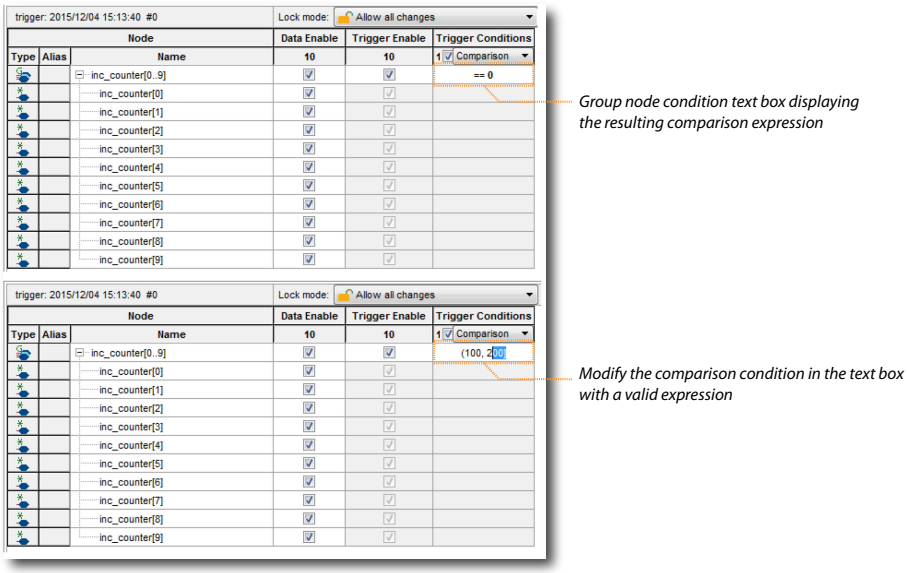
Figure 62. Specifying the Comparison Values



4. Click **OK**. The trigger editor displays the resulting comparison expression in the group node condition text box.

   Note: You can modify the comparison condition in the text box with a valid expression.

Figure 63. Resulting Comparison Condition in Text Box



5.4.3. 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** drop-down menu.

Figure 64. Accessing the Advanced Trigger Condition Tab

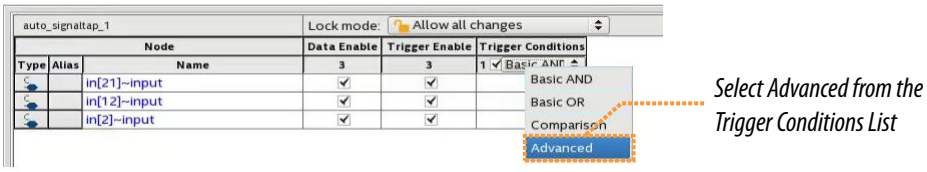
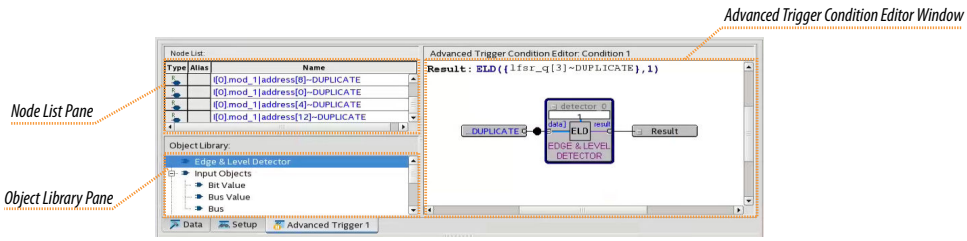


Figure 65. Advanced Trigger Condition Tab



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 60. Advanced Triggering Operators

Category	Name
Signal Detection	Edge and Level Detector
Input Objects	Bit Bit Value Bus 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

continued...

Category	Name
	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.

5.4.3.1. Examples of Advanced Triggering Expressions

The following examples show how to use Advanced Triggering:

Figure 66. Bus outa Is Greater Than or Equal to Bus outb

Trigger when bus `outa` is greater than or equal to `outb`.

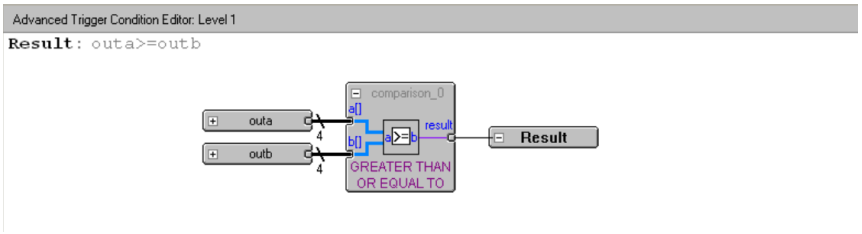


Figure 67. Enable Signal Has a Rising Edge

Trigger when bus `outa` is greater than or equal to bus `outb`, and when the enable signal has a rising edge.

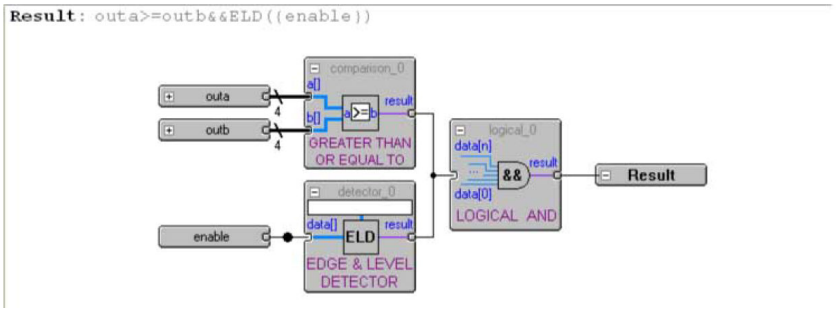
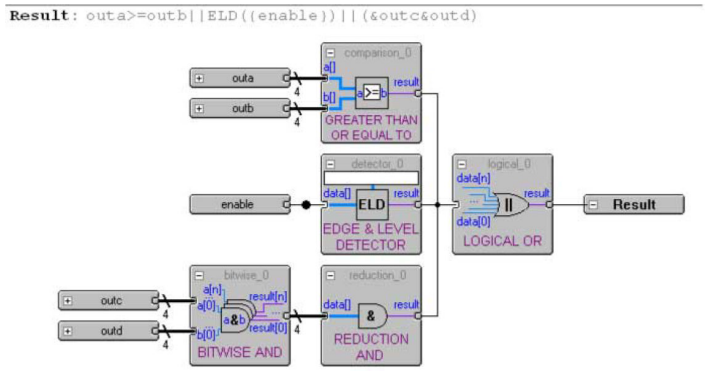


Figure 68. Bitwise AND Operation

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.

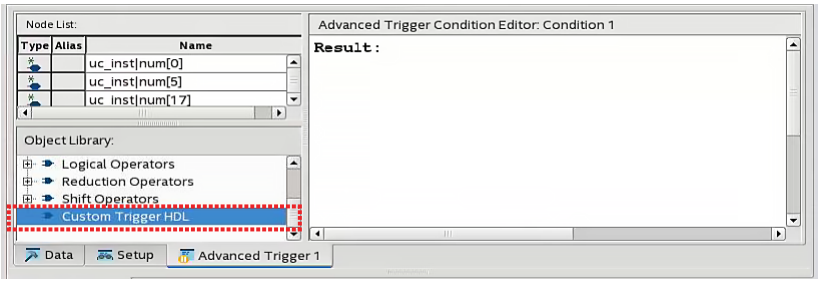


5.4.4. Custom Trigger HDL Object

Signal Tap Logic Analyzer allows you to use your own HDL module to create 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 tap 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 69. Object Library

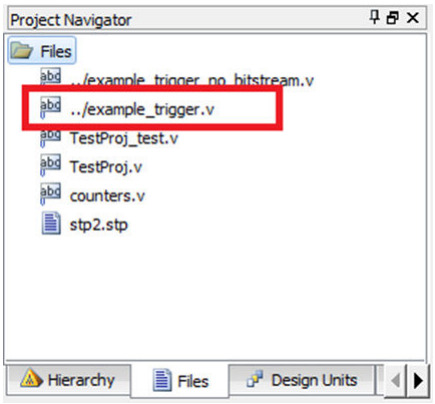


5.4.4.1. Using the Custom Trigger HDL Object

To define a custom trigger flow:

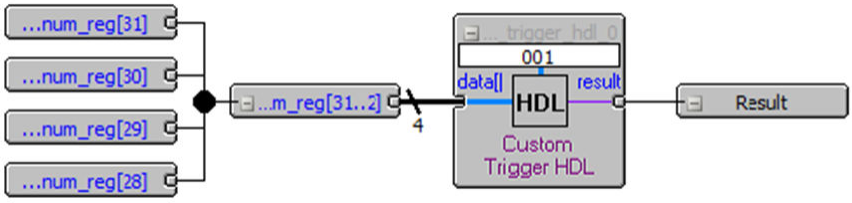
1. Select the trigger you want to edit.
2. Open the **Advanced Trigger** tab by selecting **Advanced** in the **Trigger Conditions** drop-down menu.
3. Add to your project the HDL source file that contains the trigger module using the **Project Navigator**.
   - Alternatively, append the HDL for your trigger module to a source file already included in the project.

Figure 70. HDL Trigger in the Project Navigator



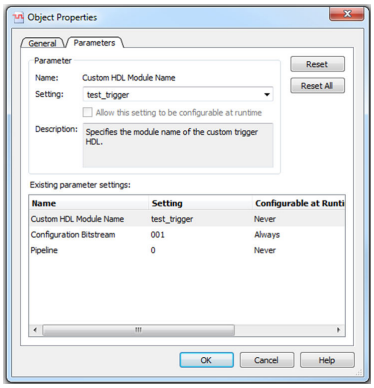
4. Implement the inputs and outputs that your Custom Trigger HDL module requires.
5. Drag in your Custom Trigger HDL object and connect the object's data input bus and result output bit to the final trigger result.

Figure 71. Custom Trigger HDL Object



6. Right-click your Custom Trigger HDL object and configure the object's properties.

Figure 72. Configure Object Properties



7. Compile your design.
8. Acquire data with Signal Tap using your custom Trigger HDL object.

Example 22. 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
```

5.4.4.2. Required Inputs and Outputs of Custom Trigger HDL Module

Table 61. 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

5.4.4.3. Custom Trigger HDL Module Properties

Table 62. 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.

5.4.5. Trigger Condition Flow Control

The Trigger Condition Flow 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.

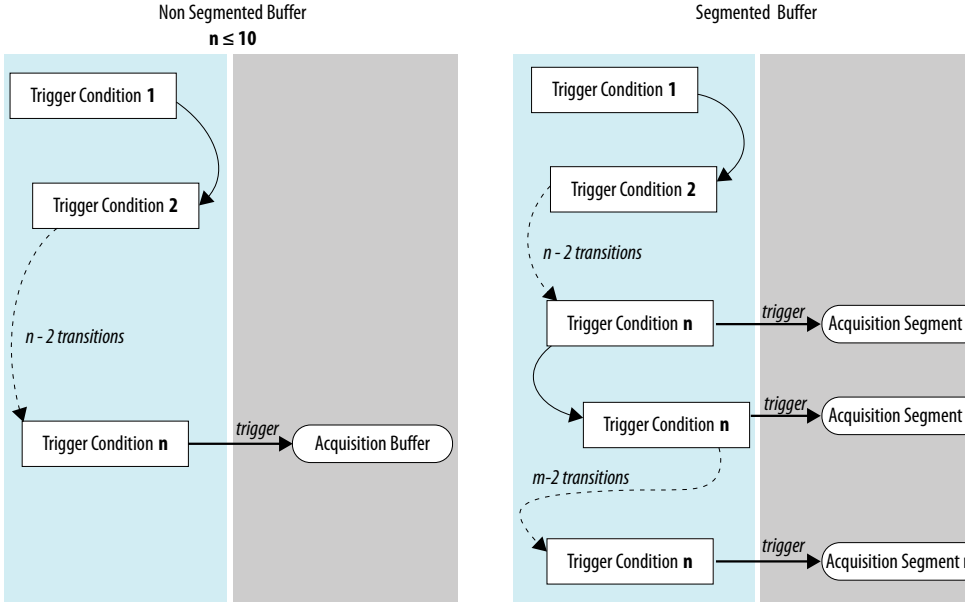
5.4.5.1. Sequential Triggering

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 Simple Sequential Triggering feature allows you to specify basic triggers, comparison triggers, advanced triggers, or a mix of all three.

Figure 73. Sequential Triggering Flow

The figure illustrates the simple sequential triggering flow for non-segmented and segmented buffers.



Notes to figure:

1. The acquisition buffer starts capture when all n triggering levels are satisfied, where $n \leq 10$.

The Signal Tap Logic Analyzer considers external triggers as level 0, evaluating external triggers before any other trigger condition.

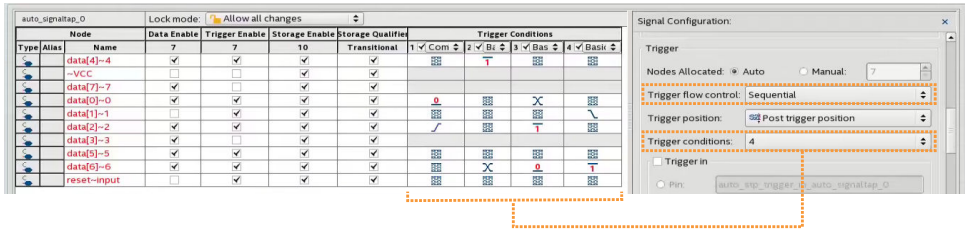
5.4.5.1.1. Configuring the Sequential Triggering Flow

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 74. Sequential Triggering Flow Configuration



5.4.5.1.2. Trigger that Skips Clock Cycles after Hitting Condition

Example 23. 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 75. Continuous Capture of Data Transaction

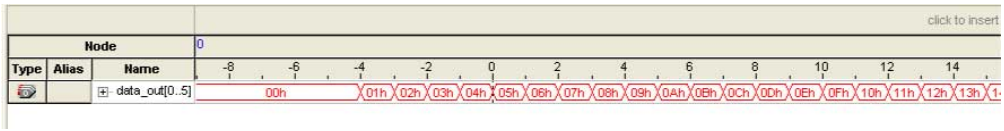
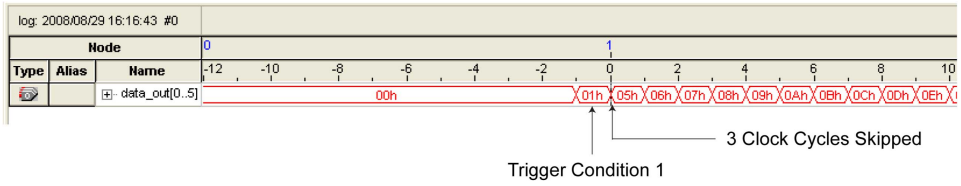


Figure 76. Capture of Data Transaction with Trigger Flow Description Applied



5.4.5.1.3. Storage Qualification with Post-Fill Count Value Less than m

Example 24. Real data acquisition of the previous scenario

Figure 77. Storage Qualification with Post-Fill Count Value Less than m (Acquisition Successfully Completes)

The data capture finishes successfully. It uses a buffer with a sample depth of 64, *m* = *n* = *10* , and `post-fill count = 5`.

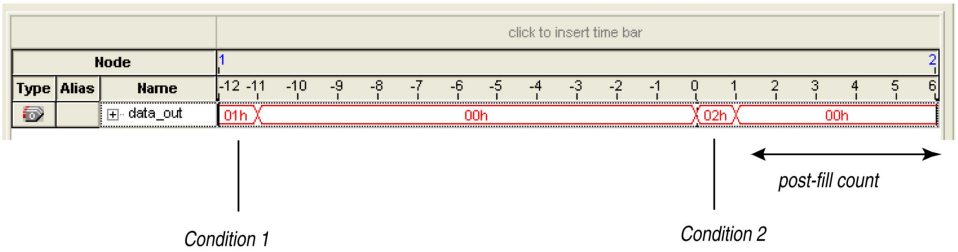


Figure 78. Storage Qualification with Post-Fill Count Value Greater than m (Acquisition Indefinitely Paused)

The logic analyzer pauses indefinitely, even after a trigger condition occurs due to a `stop_store` condition. This scenario uses a sample depth of 64, with `m = n = 10` and `post-fill count = 15`.

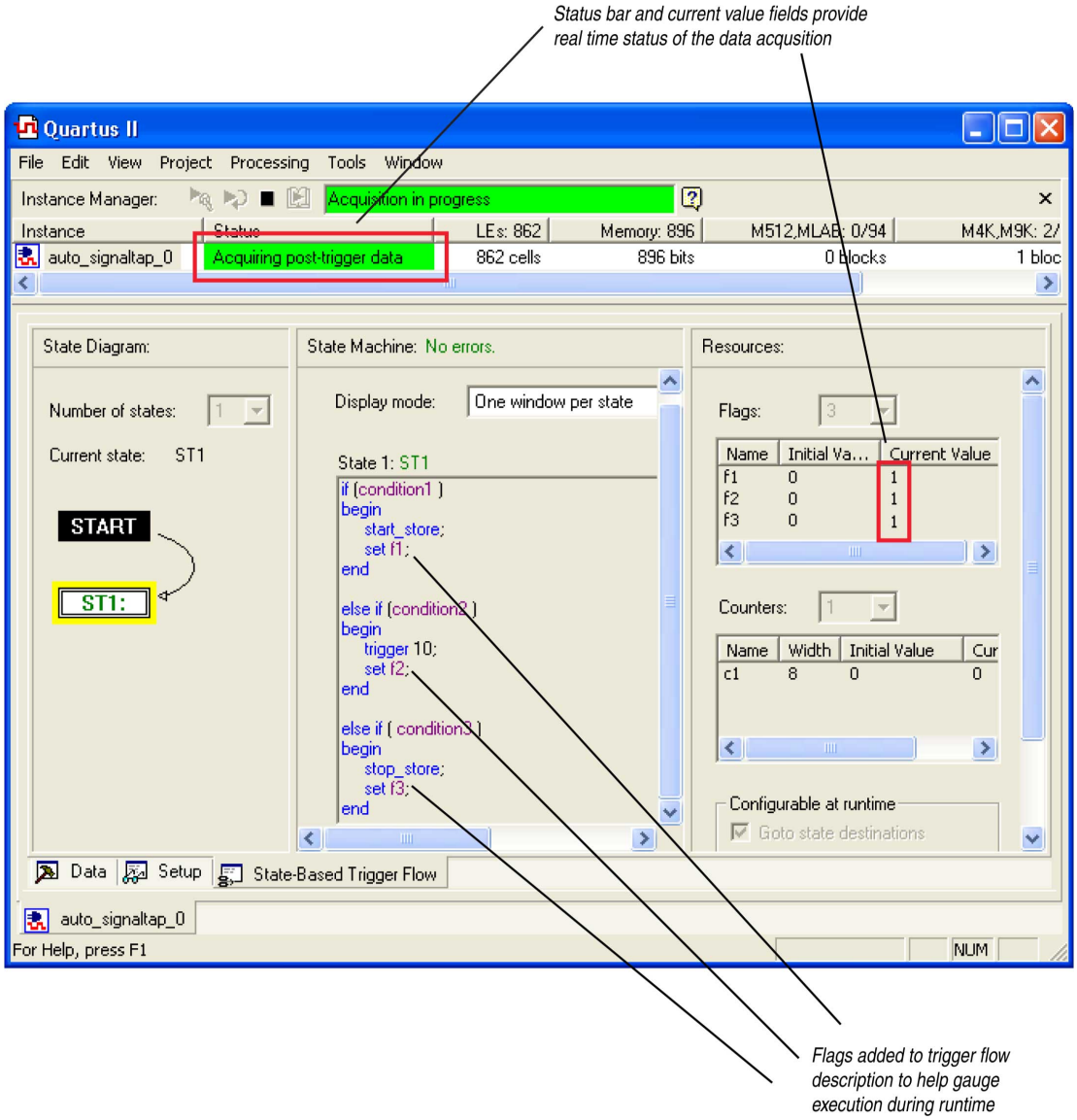
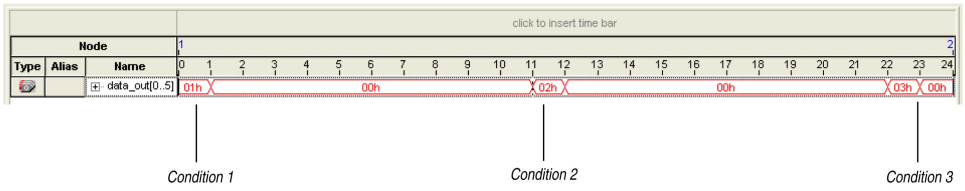


Figure 79. Waveform After Forcing the Analysis to Stop



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.

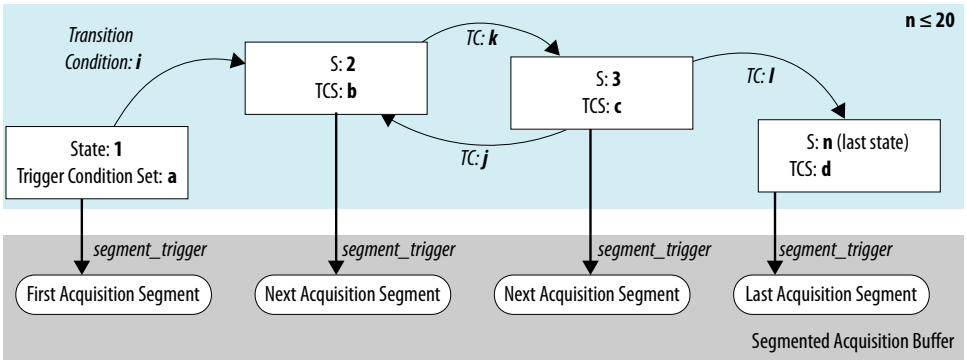
5.4.5.2. State-Based Triggering

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 within the flow GUI. Additionally, you can use the Signal Tap Trigger Flow Description Language, which is based upon conditional expressions.

Figure 80. State-Based Triggering Flow



Notes to figure:

1. You can define up to 20 different states.
2. The logic analyzer evaluates external trigger inputs that you define before any conditions in the custom state-based triggering flow.

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.

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.

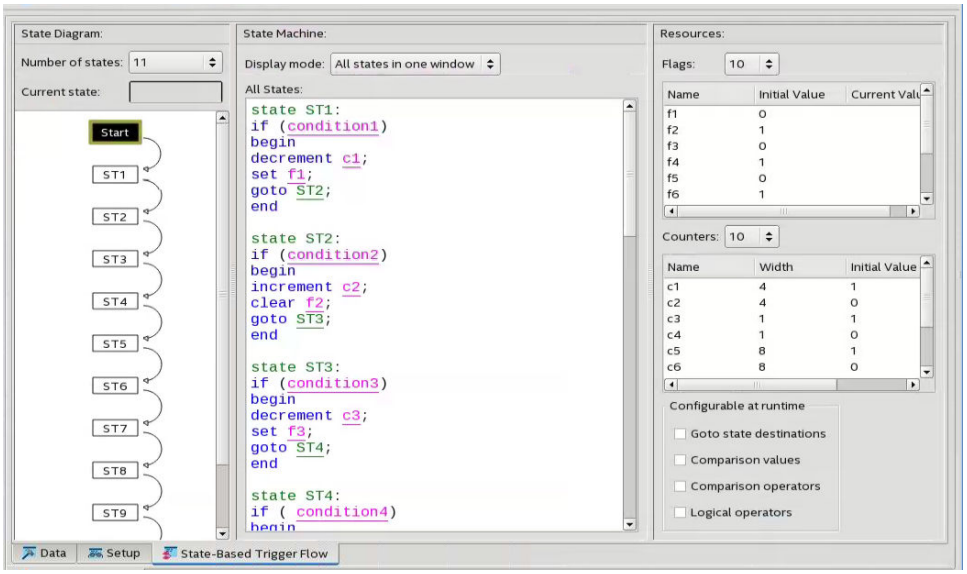
The state-based triggering flow allows you to capture a sequence of events that may not necessarily be contiguous in time. For example, a communication transaction between two devices that includes a hand shaking protocol containing a sequence of acknowledgments.

5.4.5.2.1. 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 81. State-Based Triggering Flow Tab



The **State-Based Trigger Flow** tab contains three panes:

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.

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 186

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 63. 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 198
- Runtime Reconfigurable Options on page 201

5.4.5.2.2. Trigger Lock Mode

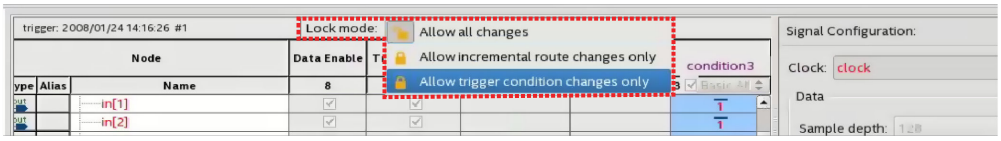
Trigger lock mode restricts changes to only the configuration settings that you specify as **Configurable at runtime**. The runtime configurable settings for the Custom Trigger Flow tab are on by default.

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 82. Allow Trigger Conditions Change Only



2. Modify the Trigger Flow conditions.

Incremental Route lock-mode restricts the GUI to only allow changes that require an Incremental Route compilation using Rapid Recompile. Use Rapid Recompile to perform incremental routing and gain a 2-4x speedup over the initial full compilation.

5.4.5.3. 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 the Logic Analyzer evaluates when a state is reached, you 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 187

<boolean_expression> on page 187

<action_list> on page 188

Related Information

Custom Triggering Flow Application Examples on page 214

5.4.5.3.1. <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.

5.4.5.3.2. <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 64. 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 65. 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.

5.4.5.3.3. <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:

Resource Manipulation Action

The resources the trigger flow description uses can be either counters or status flags.

Table 66. 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 67. 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`

continued...

Action	Description	Syntax
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 191

State Transition Action

Specifies the next state in the custom state control flow. The syntax is:
goto <*state_label*>;

5.4.5.4. 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.

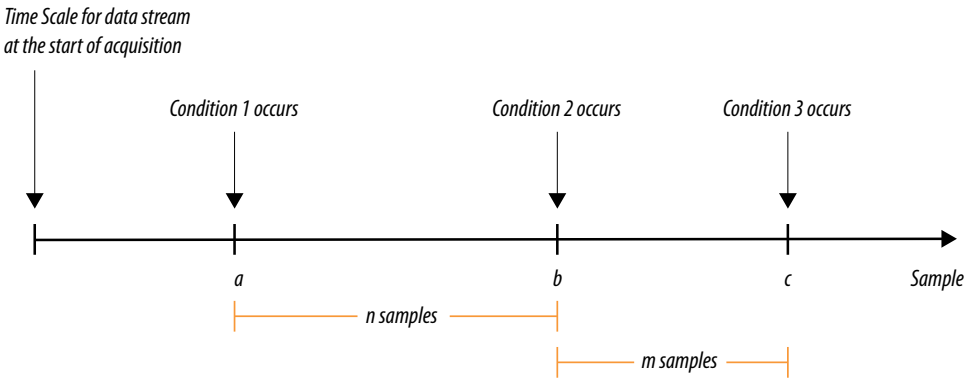
The start_store and stop_store commands are similar to the start and stop conditions of the **start/stop** storage qualifier mode. If you enable storage qualification, Signal Tap Logic Analyzer doesn't write data into the acquisition buffer until the start_store command occurs. However, in the state-based 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.

5.4.5.4.1. Storage Qualification Feature for the State-Based Trigger Flow.

This trigger flow description contains three trigger conditions that happen 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 83. 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.

5.4.6. 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.

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 segmented buffers.

Related Information

State-Based Triggering on page 183

5.4.6.1. 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 188

5.4.7. 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 is fully powered 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.

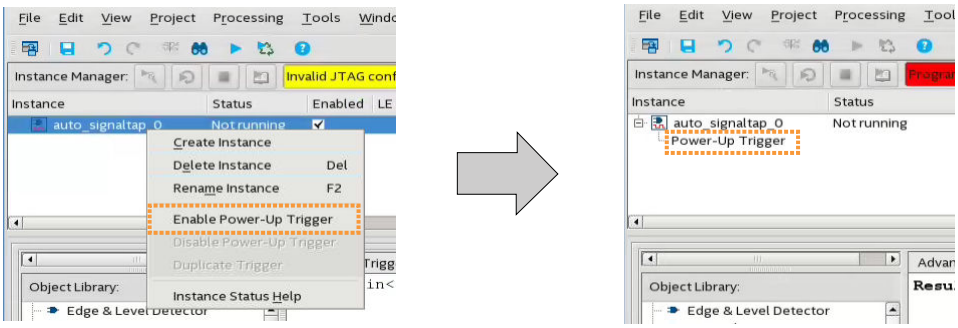
You can add a different Power-Up Trigger to each logic analyzer instance in the **Signal Tap Instance Manager** pane.

5.4.7.1. Enabling a Power-Up Trigger

To enable the Power-Up Trigger for a logic analyzer instance:

- Right-click the instance and click **Enable Power-Up Trigger**.

Figure 84. Enabling Power-Up Trigger in Signal Tap Logic Analyzer Editor



Power-Up Trigger appears as a child instance below the name of the selected instance. The node list displays the default trigger conditions.

To disable a Power-Up Trigger, right-click the instance and click **Disable Power-Up Trigger**.

5.4.7.2. 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 232

5.4.7.3. Managing Signal Tap Instances with Run-Time and Power-Up Trigger Conditions

On instances that have two both 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**.

Note: Run-time trigger conditions allow fewer adjustments than power-up trigger conditions.

5.4.8. 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.

5.4.8.1. Using the Trigger Out of One Analyzer as the Trigger In of Another Analyzer

An advanced feature of the Signal Tap Logic Analyzer is the ability to use the **Trigger out** of one analyzer as the **Trigger in** to another analyzer. This feature allows you to synchronize and debug events that occur across multiple clock domains.

To perform this operation, first turn on **Trigger out** for the source logic analyzer instance. On the **Instance** list of the **Trigger out** trigger, select the targeted logic analyzer instance. For example, if the instance named `auto_signaltap_0` should trigger `auto_signaltap_1`, select `auto_signaltap_1|trigger_in` .

Turning on **Trigger out** automatically enables the **Trigger in** of the targeted logic analyzer instance and fills in the **Instance** field of the **Trigger in** trigger with the **Trigger out** signal from the source logic analyzer instance. In this example, `auto_signaltap_0` is targeting `auto_signaltap_1`. The Trigger In **Instance** field of `auto_signaltap_1` is automatically filled in with `auto_signaltap_0|trigger_out`.

5.5. Compiling the Design

To incorporate the Signal Tap logic in your design and enable the JTAG connection, you must compile your project. When you add a `.stp` file to your project, the Signal Tap Logic Analyzer becomes part of your design. When you debug your design with a traditional external logic analyzer, you must often make changes to the signals you want to monitor as well as the trigger conditions.

Note: Because these adjustments require that you recompile your design when using the Signal Tap Logic Analyzer, use the Signal Tap Logic Analyzer feature along with incremental compilation in the Intel Quartus Prime software to reduce recompilation time.

5.5.1. Faster Compilations with Intel Quartus Prime Incremental Compilation

You can add a Signal Tap Logic Analyzer instance to your design without recompiling your original source code. Incremental compilation enables you to preserve the synthesis and fitting results of your original design.

When you compile your design including a `.stp` file, Intel Quartus Prime software automatically adds the `sld_signaltap` and `sld_hub` entities to the compilation hierarchy. These two entities are the main components of the Signal Tap Logic Analyzer, providing the trigger logic and JTAG interface required for operation.

Incremental compilation is also useful when you want to modify the configuration of the `.stp` file. For example, you can change the buffer sample depth or memory type without performing a full compilation. Instead, you only recompile the Signal Tap Logic Analyzer, configured as its own design partition.

5.5.1.1. Enabling Incremental Compilation for Your Design

When enabled for your design, the Signal Tap Logic Analyzer is always a separate partition. After the first compilation, you can use the Signal Tap Logic Analyzer to analyze signals from the post-fit netlist. If your partitions are designed correctly, subsequent compilations due to Signal Tap Logic Analyzer settings take less time.

The netlist type for the top-level partition defaults to **source**. To take advantage of incremental compilation, specify the Netlist types for the partitions you want to tap as **Post-fit**.

Related Information

Intel Quartus Prime Incremental Compilation for Hierarchical and Team-Based Design documentation

5.5.1.2. Using Incremental Compilation with the Signal Tap Logic Analyzer

The Signal Tap Logic Analyzer uses the incremental compilation flow by default. For all signals that you want to connect to the Signal Tap Logic Analyzer from the post-fit netlist:

1. In the Design Partitions window, set the netlist type of the partition that contains the signals to **Post-Fit**, with a Fitter Preservation Level of **Placement and Routing**.
2. In the **Node Finder**, use the **Signal Tap: post-fitting filter** to add the signals of interest to your Signal Tap configuration file.
3. If you want to add signals from the pre-synthesis netlist, set the netlist type to **Source File** and use the **Signal Tap: pre-synthesis filter** in the **Node Finder**. Do not use the netlist type **Post-Synthesis** with the Signal Tap Logic Analyzer.

Caution: When using post-fit and pre-synthesis nodes:

- Read all incremental compilation guidelines to ensure the proper partitioning of a project.
- To speed up compile time, use only post-fit nodes for partitions specified as preservation-level post-fit.
- Do not mix pre-synthesis and post-fit nodes in any partition. If you must tap pre-synthesis nodes for a particular partition, make all tapped nodes in that partition pre-synthesis nodes and change the netlist type to **source** in the design partitions window.

Node names can differ between a pre-synthesis netlist and a post-fit netlist. In general, registers and user input signals share common names between the two netlists. During compilation, certain optimizations change the names of combinational signals in your RTL. If the type of node name chosen does not match the netlist type, the compiler may not be able to find the signal to connect to your Signal Tap Logic Analyzer instance for analysis. The compiler issues a critical warning to alert you of this scenario. The signal that is not connected is tied to ground in the **Signal Tap data** tab.

If you do use incremental compilation flow with the Signal Tap Logic Analyzer and source file changes are necessary, be aware that you may have to remove compiler-generated post-fit net names. Source code changes force the affected partition to go through resynthesis. During synthesis, the compiler cannot find compiler-generated net names from a previous compilation.

Note: Intel FPGA recommends using only registered and user-input signals as debugging taps in your `.stp` whenever possible.

Both registered and user-supplied input signals share common node names in the pre-synthesis and post-fit netlist. As a result, using only registered and user-supplied input signals in your `.stp` limits the changes you need to make to your Signal Tap Logic Analyzer configuration.

You can check the nodes that are connected to each Signal Tap instance using the In-System Debugging compilation reports. These reports list each node name you selected to connect to a Signal Tap instance, the netlist type used for the particular connection, and the actual node name used after compilation. If the incremental compilation flow is not used, the In-System Debugging reports are located in the Analysis & Synthesis folder. If the incremental compilation flow is used, this report is located in the Partition Merge folder.

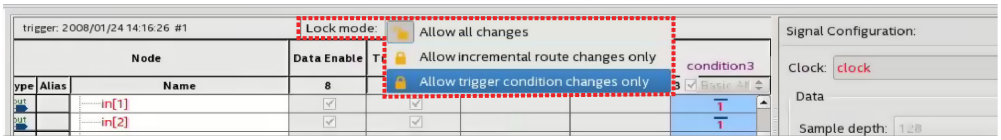
To verify that your original design was not modified, examine the messages in the **Partition Merge** section of the Compilation Report.

Unless you make changes to your design partitions that require recompilation, only the Signal Tap design partition is recompiled. If you make subsequent changes to only the `.stp`, only the Signal Tap design partition must be recompiled, reducing your recompilation time.

5.5.2. Prevent Changes Requiring Recompilation

Configure the `.stp` to prevent changes that normally require recompilation. To do this, select a **Lock mode** from above the node list in the **Setup** tab. To lock your configuration, choose **Allow trigger condition changes only**.

Figure 85. Allow Trigger Conditions Change Only



Related Information

Verify Whether You Need to Recompile Your Project on page 196

5.5.3. Verify Whether You Need to Recompile Your Project

Before starting a debugging session, do not make any changes to the `.stp` settings that require recompiling the project.

To verify whether a change you made requires recompiling the project, check the Signal Tap status display at the top of the **Instance Manager** pane. This feature allows you to undo the change, so that you do not need to recompile your project.

Related Information

Prevent Changes Requiring Recompilation on page 195

5.5.4. Incremental Route with Rapid Recompile

You can use Incremental Route with Rapid Recompile to decrease compilation times. After performing a full compilation on your design, you can use the Incremental Route flow to achieve a 2-4x speedup over a flat compile. The Incremental Route flow is not compatible with Partial Reconfiguration.

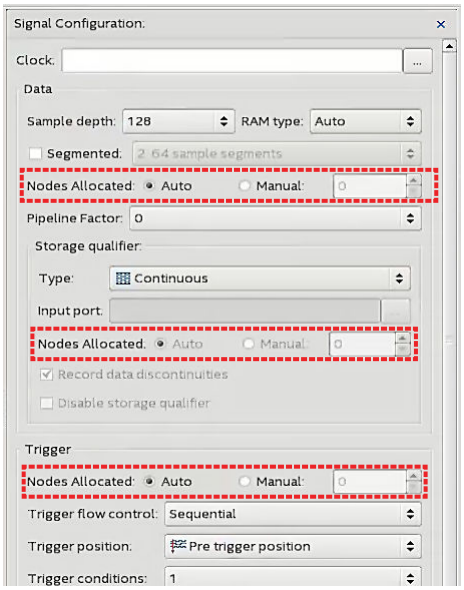
Intel Quartus Prime Standard Edition software supports Incremental Route with Rapid Recompile for Arria V, Cyclone V, and Stratix V devices.

5.5.4.1. Using the Incremental Route Flow

To use the Incremental Route flow:

1. Open your design and run **Analysis & Elaboration** (or a full compilation) to give node visibility in Signal Tap.
2. Add Signal Tap to your design.
3. In the Signal Tap **Signal Configuration** pane, specify **Manual** in the **Nodes Allocated** field for Trigger and Data nodes (and Storage Qualifier, if used).

Figure 86. Manually Allocate Nodes

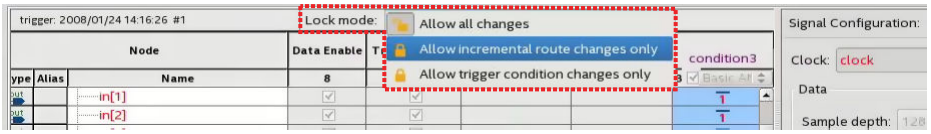


Manual node allocation allows you to control the number of nodes compiled into the design, which is critical for the Incremental Route flow.

When you select **Auto** allocation, the number of nodes compiled into the design matches the number of nodes in the **Setup** tab. If you add a node later, you create a mismatch between the amount of nodes the device requires and the amount of compiled nodes, and you must perform a full compilation.

4. Specify the number of nodes that you estimate necessary for the debugging process. You can increase the number of nodes later, but this requires more compilation time.
5. Add the nodes that you want to tap.
6. If you have not fully compiled your project, run a full compilation. Otherwise, start incremental compile using Rapid Recompile.
7. Debug and determine additional signals of interest.
8. (Optional) Select **Allow incremental route changes only** lock-mode.

Figure 87. Incremental Route Lock-Mode



9. Add additional nodes in the Signal Tap **Setup** tab.
   - Do not exceed the number of manually allocated nodes you specified.
   - Avoid making changes to non-runtime configurable settings.
10. Click the Rapid Recompile icon from the toolbar. Alternatively, click **Processing ➤ Start Rapid Recompile**.

Note: The previous steps set up your design for Incremental Route, but the actual Incremental Route process begins when you perform a Rapid Recompile.

5.5.4.2. Tips to Achieve Maximum Speedup

- Basic AND (which applies to Storage Qualifier as well as trigger input) is the fastest for the Incremental Route flow.
- Basic OR is slower for the Incremental Route flow, but if you avoid changing the parent-child relationship of nodes within groups, you can minimize the impact on compile time. You can change the sibling relationships of nodes.
  - Basic OR and advanced triggers require re-synthesis when you change the number/names of tapped nodes.
- Use the Incremental Route lock-mode to avoid inadvertent changes requiring a full compilation.

5.5.5. Timing Preservation with the Signal Tap Logic Analyzer

In addition to verifying functionality, timing closure is one of the most crucial processes in successful operation of a design.

Note: When you compile a project with a Signal Tap Logic Analyzer without the use of incremental compilation, you must add IP to the existing design. This addition often impacts the existing placement, routing, and timing of the design. To minimize the effect that the Signal Tap Logic Analyzer has on the design, use incremental compilation for the project. Incremental compilation is the default setting in new designs. You can easily enable incremental compilation in existing designs. When the Signal Tap Logic Analyzer is in a design partition, it has little to no affect on the design.

For Intel Arria 10 devices, the Intel Quartus Prime Standard Edition software does not support timing preservation for post-fit taps with Rapid Recompile.

The following techniques can help you maintain timing:

- 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 Standard Edition User Guide: Design Optimization*

5.5.6. 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.

5.5.6.1. Signal Tap Logic in Critical Path

If Signal Tap logic is part of your critical path, follow these tips to speed up the performance of the Signal Tap Logic Analyzer:

- **Disable runtime configurable options**—Certain resources are allocated to accommodate for runtime flexibility. If you use either advanced triggers or State-based triggering flow, disable runtime configurable parameters for a boost in f_{MAX} of the Signal Tap logic.
  - If you are using State-based triggering flow, try disabling the **Goto state destination** option and performing a recompilation before disabling the other runtime configurable options. The **Goto state destination** option has the greatest impact on f_{MAX}, as compared to the other runtime configurable options.
- **Minimize the number of signals that have Trigger Enable selected**—By default, Signal Tap Logic Analyzer enable 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.

5.5.6.2. Signal Tap Logic Using Critical Resources

If your design is resource constrained, follow these tips to reduce the logic or memory the Signal Tap Logic Analyzer uses:

- **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 uses if you limit the segments in your sampling buffer
- **Disable the Data Enable for signals that you use only for triggering**—By default, 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 on memory resources.

5.6. Program the Target Device or Devices

After you add the Signal Tap Logic Analyzer to your project and re-compile, you can configure the FPGA target device.

If you want to debug multiple designs simultaneously, configure the device from the `.stp` instead of the Intel Quartus Prime Programmer. This allows you to open more than one `.stp` file and program multiple devices.

5.6.1. Ensure Setting Compatibility Between .stp and .sof Files

A `.stp` file is compatible with a `.sof` file when the settings for the logic analyzer, such as the size of the capture buffer and the monitoring and triggering signals match the programming settings of 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 Editor.

- To ensure programming compatibility, program the device with the `.sof` file generated in the most recent compilation.
- To check whether a particular `.sof` is compatible with the current Signal Tap configuration, attach the `.sof` to the SOF manager.

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.

As a best practice, use the `.stp` file with a Intel Quartus Prime project. The project database contains information about the integrity of the current Signal Tap Logic Analyzer session. Without the project database, there is no way to verify that the current `.stp` file matches the `.sof` file in the device. If you have an `.stp` file that does not match the `.sof` file, the Signal Tap Logic Analyzer can capture incorrect data.

Related Information

Manage Multiple Signal Tap Files and Configurations on page 167

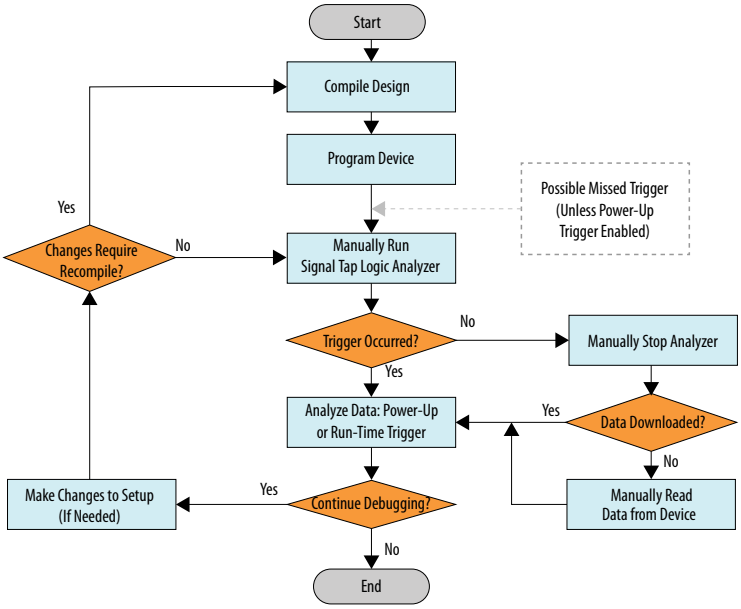
5.7. Running the Signal Tap Logic Analyzer

Debugging Signal Tap Logic Analyzer is similar using an external logic analyzer. You initialize the logic analyzer by starting an analysis. When your trigger event occurs, the logic analyzer stores the captured data in the device's memory buffer, and then transfers this data to the `.stp` file with the JTAG connection.

You can also perform the equivalent of a force trigger instruction that lets you view the captured data currently in the buffer without a trigger event occurring.

The flowchart shows how you operate the Signal Tap Logic Analyzer. indicates where Power-Up and Runtime Trigger events occur and when captured data from these events is available for analysis.

Figure 88. Power-Up and Runtime Trigger Events Flowchart



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.

Related Information

Design Debugging Using In-System Sources and Probes on page 232

5.7.1. Runtime Reconfigurable Options

When you use Runtime Trigger mode, you can change certain settings in the `.stp` without recompiling your design.

Table 68. Runtime Reconfigurable Features

Runtime Reconfigurable Setting	Description
Basic Trigger Conditions and Basic Storage Qualifier Conditions	You can 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 scenario of events without the need to recompile the design. 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 **Allow Trigger Condition changes only** above the node list.

In Incremental Route lock mode, **Allow incremental route changes only**, limits to changes that only require an Incremental Route compilation, and not a full compile.

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.

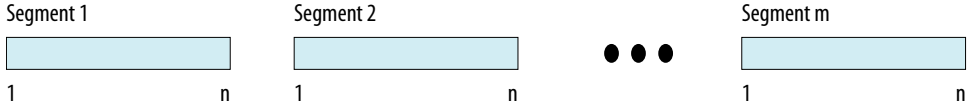
```
state ST1:
if ( condition1 && (c1 <= m) )// each "segment"  triggers on condition
                               // 1
begin                            // m  = number of total "segments"
   start_store;
   increment c1;
   goto ST2:
End

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 89. Segmented Buffer Created with Storage Qualifier and State-Based Trigger

Total sample depth is fixed, where *m* x *n* must equal sample depth.



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 added for a 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
```

5.7.2. 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 69. 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.

continued...

Message	Message Description
Acquiring (Power-Up) pre-trigger data (1)	The trigger condition has not been evaluated yet. 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 condition has occurred. 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.

5.8. View, Analyze, and Use Captured Data

The Signal Tap Logic Analyzer interface allows you to examine the data captured manually or with a trigger. When in the Data view, you isolate the data of interest with the drag-to-zoom feature, enabled with a left-click.

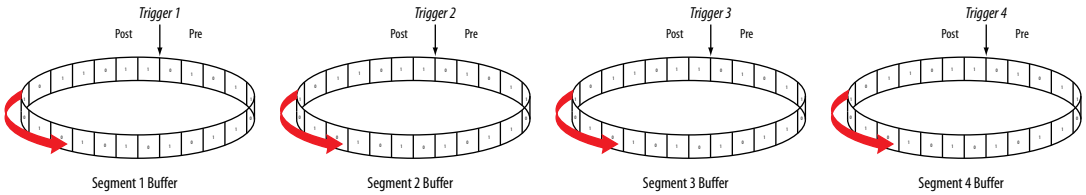
5.8.1. Capturing Data Using Segmented Buffers

Segmented Acquisition buffers can perform captures with separate trigger conditions for each acquisition segment. These 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 90. Segmented Acquisition Buffer

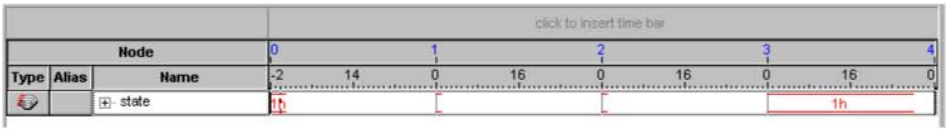


When the Signal Tap Logic Analyzer finishes an acquisition with a segment and advances to the next segment to start a new acquisition. The data capture that appears in the waveform viewer depends on when a trigger condition occurs. The figure illustrates the data capture method. The Trigger markers—Trigger 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 refer to trigger conditions 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 Segment 2 buffer starts when either Segment Buffer 1 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 91. 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 (10% of the data is before the trigger condition and 90% 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 159

5.8.2. Differences in Pre-Fill Write Behavior Between 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 with 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 perform a **Stop Analysis**, Signal Tap prevents the buffer from being dumped during the first acquisition prior to a trigger condition.

Buffers with Storage Qualification

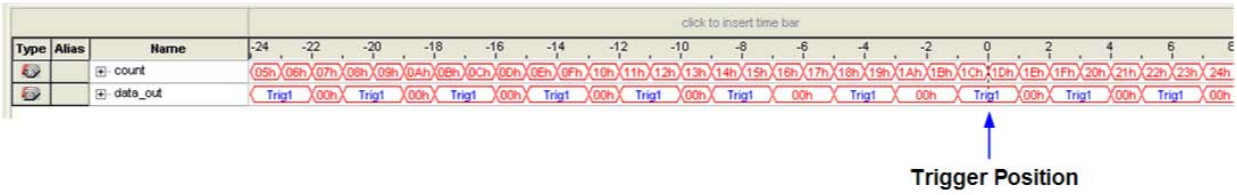
For buffers using a storage qualification mode, the Signal Tap Logic Analyzer immediately evaluates all trigger conditions while writing samples into the acquisition memory. This evaluation is especially important when using any storage qualification on the data set. The logic analyzer may miss a trigger 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.

5.8.2.1. Example

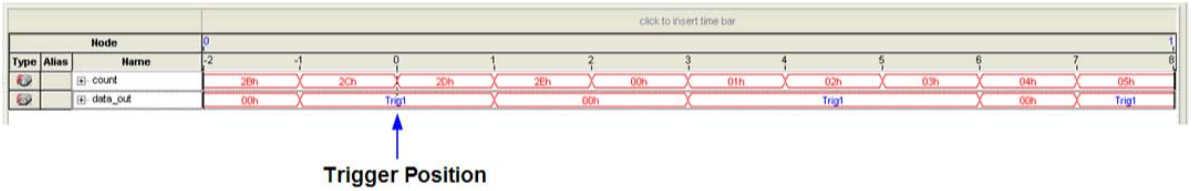
The figures for continuous data capture and conditional data capture show the difference between a non-segmented buffer in continuous mode and a non-segmented buffer using a storage qualifier. The configuration of the logic analyzer waveforms is a base trigger condition, sample depth of 64 bits, and **Post trigger position**.

Figure 92. 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 needs to 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 93. Signal Tap Logic Analyzer Conditional Data Capture



Note to figure:

1. Conditional capture, storage always enabled, post-fill count.
2. Signal Tap Logic Analyzer capture of 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.

5.8.3. 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.

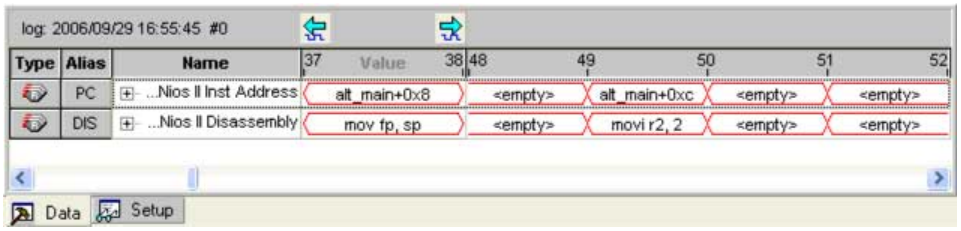
5.8.4. Automatic Mnemonics with a Plug-In

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, then 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 **Instance 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 94. Data Tab when the Nios II Plug-In is Used



5.8.5. 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 many of the tools found in the Intel Quartus Prime software, as well as in your design files. This lets you find the source of the problem quickly so you can modify your design to correct the flaw. To locate a signal from the Signal Tap Logic Analyzer in one of the Intel Quartus Prime software tools or your design files, 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

5.8.6. Saving Captured Data

When you save a data capture, 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 analysis is set to **Auto-run mode**, the Logic Analyzer creates a separate entry in the Data Log to store the data captured each time the trigger occurred. This allows you to review the captured data for each trigger event.

The default name for a log is based time stamp when the Logic Analyzer acquired 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 167

5.8.7. Exporting Captured Data to Other File Formats

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 Signal Tap Logic Analyzer, on the File menu, click **Export** and specify the **File Name**, **Export Format**, and **Clock Period**.

5.8.8. Creating a Signal Tap List File

A `.stp` list file contains all the data the logic analyzer captures for a trigger event, in text format.

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.

The `.stp` 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 `.stp` list file in the Intel Quartus Prime software, click **File ➤ Create/Update ➤ Create Signal Tap List File**.

Related Information

Adding Signals with a Plug-In on page 155

5.9. Other Features

The Signal Tap Logic Analyzer provides optional features not specific to a task flow. The following techniques can offer advantages in specific scenarios.

5.9.1. Creating Signal Tap File from Design Instances

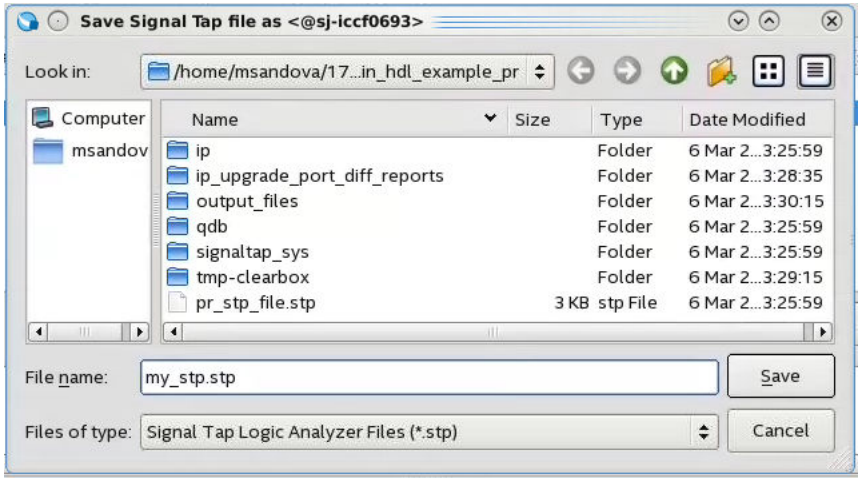
In addition to providing GUI support for generation of `.stp` files, the Intel Quartus Prime software supports generation of a Signal Tap instance from logic defined in HDL source files. This technique is helpful to modify runtime configurable trigger conditions, acquire data, and view acquired data on the data log via Signal Tap utilities.

5.9.1.1. Creating a .stp File from a Design Instance

To generate a `.stp` file from parameterized HDL instances within your design:

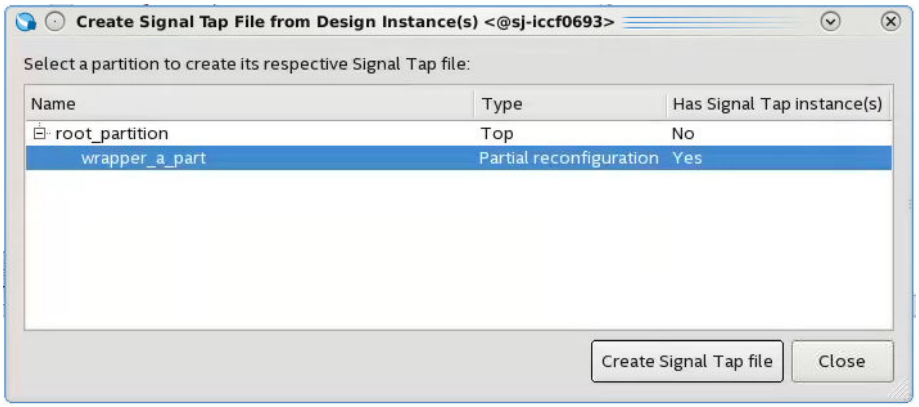
1. Open or create an Intel Quartus Prime project that includes one or more HDL instances of the Signal Tap logic analyzer.
2. Click **Processing ➤ Start ➤ Start Analysis & Synthesis**.
3. Click **File ➤ Create/Update ➤ Create Signal Tap File from Design Instance(s)**.
4. Specify a location for the `.stp` file that generates, and click **Save**.

Figure 95. Create Signal Tap File from Design Instances Dialog Box



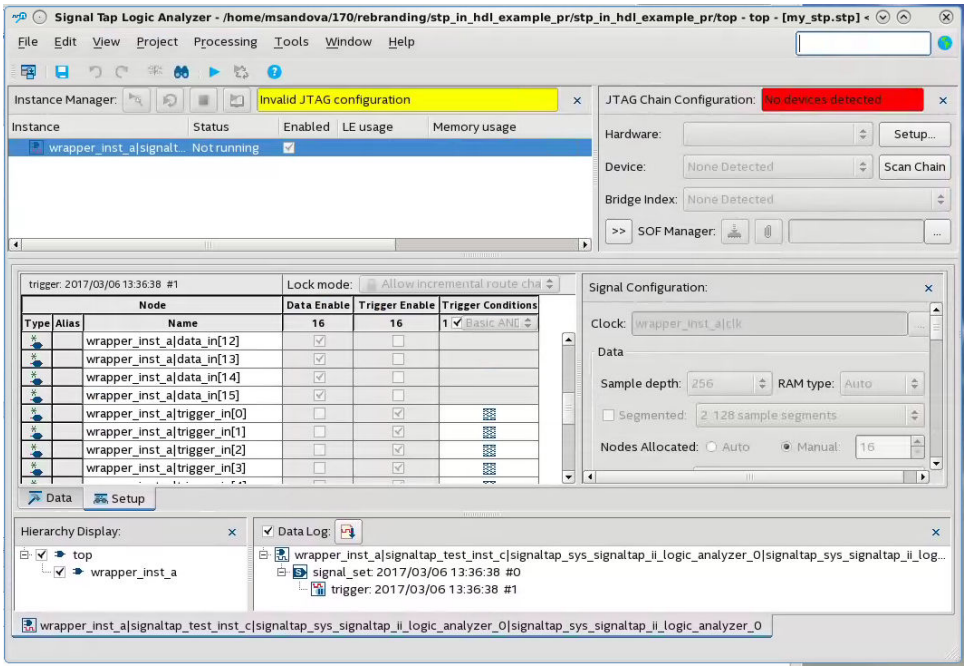
Note: If your project contains partial reconfiguration partitions, the **Create Signal Tap File from Design Instance(s)** dialog box displays a tree view of the PR partitions in the project. Select a partition from the view, and click **Create Signal Tap file**. The resultant `.stp` file that generates contains all HDL instances in the corresponding PR partition. The resultant `.stp` file does not include the instances in any nested partial reconfiguration partition.

Figure 96. Selecting Partition for `.stp` File Generation



After successful `.stp` file creation, the **Signal Tap Logic Analyzer** appears. All the fields are read-only, except runtime-configurable trigger conditions.

Figure 97. Generated `.stp` File



Related Information

Custom Trigger HDL Object on page 175

5.9.2. Using the Signal Tap MATLAB MEX Function to Capture Data

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 by calling the MATLAB MEX function `alt_signaltap_run`, built into the Intel Quartus Prime software. If you use the MATLAB MEX function in a loop, you can perform as many acquisitions in the same amount of time as you can when using Signal Tap in the Intel Quartus Prime software environment.

Note: The Signal Tap MATLAB MEX function 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. Other MATLAB MEX function options are described in the table:

Table 70. 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.

5.9.3. Using Signal Tap in a Lab Environment

You can install a stand-alone version of the Signal Tap Logic Analyzer. This version is particularly useful in a lab environment in which you do not have a workstation that meets the requirements for a complete Intel Quartus Prime installation, or if you do not have a license for a full installation of the Intel Quartus Prime software. The standalone version of the Signal Tap Logic Analyzer is included with and requires the Intel Quartus Prime stand-alone Programmer which is available from the Downloads page of the Intel website.

5.9.4. Remote Debugging Using the Signal Tap Logic Analyzer

5.9.4.1. Debugging Using a Local PC and an SoC

You can use the System Console with Signal Tap Logic Analyzer to remote debug your Intel FPGA SoC. This method requires one local PC, an existing TCP/IP connection, a programming device at the remote location, and an Intel FPGA SoC.

Related Information

Remote Hardware Debugging over TCP/IP

5.9.4.2. Debugging Using a Local PC and a Remote PC

You can use the Signal Tap Logic Analyzer to debug a design that is running on a device attached to a PC in a remote location.

To perform a remote debugging session, you must have the following setup:

- The Intel Quartus Prime software installed on the local PC
- Stand-alone Signal Tap Logic Analyzer or the full version of the Intel Quartus Prime software installed on the remote PC
- Programming hardware connected to the device on the PCB at the remote location
- TCP/IP protocol connection

5.9.4.2.1. Equipment Setup

1. On the PC in the remote location, install the standalone version of the Signal Tap Logic Analyzer, included in the Intel Quartus Prime stand-alone Programmer, or the full version of the Intel Quartus Prime software.
2. Connect the remote computer to Intel programming hardware, such as the or Intel FPGA Download Cable.
3. On the local PC, install the full version of the Intel Quartus Prime software.
4. Connect the local PC to the remote PC across a LAN with the TCP/IP protocol.

5.9.5. Using the Signal Tap Logic Analyzer in Devices with Configuration Bitstream Security

Certain 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. However, note that JTAG configuration is not possible after the security key has been programmed into the device.

Intel FPGA recommends that you use an unencrypted bitstream during the prototype and debugging phases of the design. Using an unencrypted bitstream allows you to generate new programming files and reconfigure the device over the JTAG connection during the debugging cycle.

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 the FPGA is configured with a Signal Tap Logic Analyzer instance in the design, when you open the Signal Tap Logic Analyzer in the Intel Quartus Prime software, you then scan the chain and are ready to acquire data with the JTAG connection.

5.9.6. Monitor FPGA Resources Used by the Signal Tap Logic Analyzer

The Signal Tap Logic Analyzer has a built-in resource estimator that calculates the logic resources and amount of memory that each logic analyzer instance uses. Furthermore, because the most demanding on-chip resource for the logic analyzer is memory usage, the resource estimator reports the ratio of total RAM usage in your design to the total amount of RAM available, given the results of the last compilation. The resource estimator provides a warning if a potential for a “no-fit” occurs.

You can see resource usage (by instance and total) in the columns of the **Instance Manager** pane of the Signal Tap Logic Analyzer Editor. Use this feature when you know that your design is running low on resources.

The logic element value that the resource usage estimator reports may vary by as much as 10% from the actual resource usage.

5.10. Design Example: Using Signal Tap Logic Analyzers

The system in this example contains many components, including a Nios 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 processor executes a simple C program from on-chip memory and waits for you to press a button.

Related Information

Debug Nios® II Systems with SignalTap* II Embedded Logic Analyzer

5.11. Custom Triggering Flow Application Examples

The custom triggering flow in the Signal Tap Logic Analyzer is most useful for organizing a number of triggering conditions and for precise control over the acquisition buffer. This section provides two application examples for defining a

custom triggering flow within the Signal Tap Logic Analyzer. Both examples can be easily copied and pasted directly into the state machine description box by using the state display mode **All states in one window**.

Related Information

On-chip Debugging Design Examples website

5.11.1. Design Example 1: Specifying a 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.

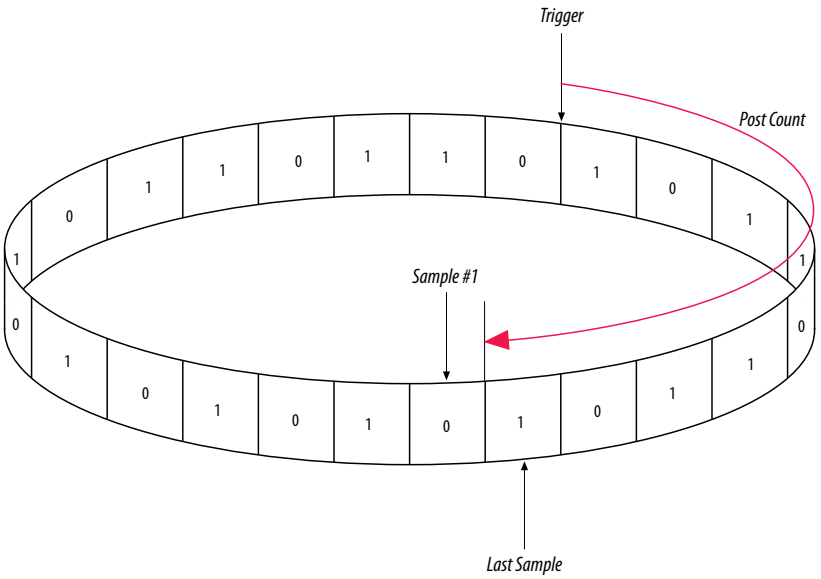
The example shows how to apply a trigger position to all segments in the acquisition buffer. The example describes a triggering flow for an acquisition buffer split into four segments. If each acquisition segment is 64 samples in depth, the trigger position for each buffer is at sample #34. The acquisition 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 98. Specifying a Custom Trigger Position



5.11.2. Design Example 2: Trigger When triggercond1 Occurs Ten Times between triggercond2 and triggercond3

The custom trigger flow description is often useful to count a sequence of events before triggering the acquisition buffer. The example shows such a sample flow. 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;
```

5.12. Signal Tap Scripting Support

The Intel Quartus Prime supports automating Signal Tap procedures in a scripting environment, as Tcl scripts or through the `quartus_stp` executable. 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 `quartus_sh --qhelp` at the command prompt.

Related Information

- Tcl Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*
- Command Line Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*

5.12.1. Signal Tap Command-Line Options

You can use the following options with the `quartus_stp` executable:

Table 71. 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.

continued...

Option	Usage	Description
		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.
`--create_signaltap_hdl_file`	Optional	Creates an `.stp` file representing the Signal Tap instance. You must use the `--stp_file` option to create an `.stp`. Equivalent to the **Create Signal Tap File from Design Instances** command in the Intel Quartus Prime software .

Examples

The first example illustrates how to compile a design with the Signal Tap Logic Analyzer at the command line.

```
quartus_stp filtref --stp_file stp1.stp --enable
quartus_map filtref --source=filtref.bdf --family=CYCLONE
quartus_fit filtref --part=EP1C12Q240C6 --fmax=80MHz --tsu=8ns
quartus_asm filtref
```

The `quartus_stp --stp_file stp1.stp --enable` command creates the QSF variable and instructs the Intel Quartus Prime software to compile the `stp1.stp` file with your design. The `--enable` option must be applied for the Signal Tap Logic Analyzer to compile into your design.

The following example creates a new `.stp` after building the Signal Tap Logic Analyzer instance with the IP Catalog.

```
quartus_stp filtref --create_signaltap_hdl_file --stp_file stp1.stp
```

5.12.2. 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 25. Continuously Capturing Data

This excerpt shows commands you can use to continuously capture data. Once the capture meets trigger condition **e**, 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
```

5.13. 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
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • Renamed topic: *Untappable Signals* to *Signals Unavailable for Signal Tap Debugging*.
2017.11.06	17.1.0	• 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.
2017.05.08	17.0.0	• Added: Open Standalone Signal Tap Logic Analyzer GUI. • Updated figures on Create Signal Tap File from Design Instance(s).
2016.10.31	16.1.0	• 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*.
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.
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.

continued...

Document Version	Intel Quartus Prime Version	Changes
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

7. In-System Debugging Using External Logic Analyzers

7.1. 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

Device Support Center

7.2. 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 72. 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...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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

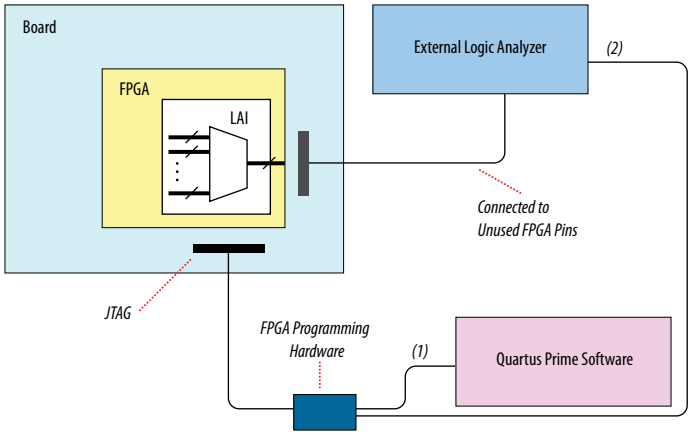
System Debugging Tools Overview on page 7

7.2.1. 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 99. LAI and Hardware Setup

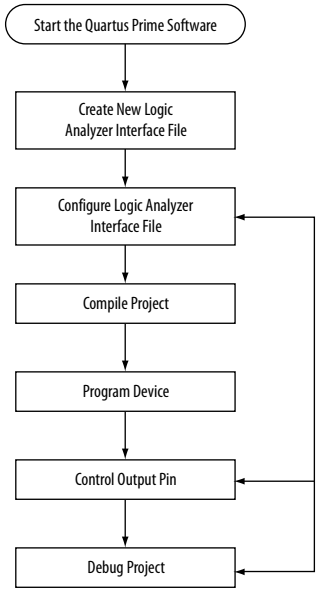


Notes to figure:

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 the JTAG port. Support varies by vendor.

7.3. Flow for Using the LAI

Figure 100. LAI Workflow



Notes to figure:

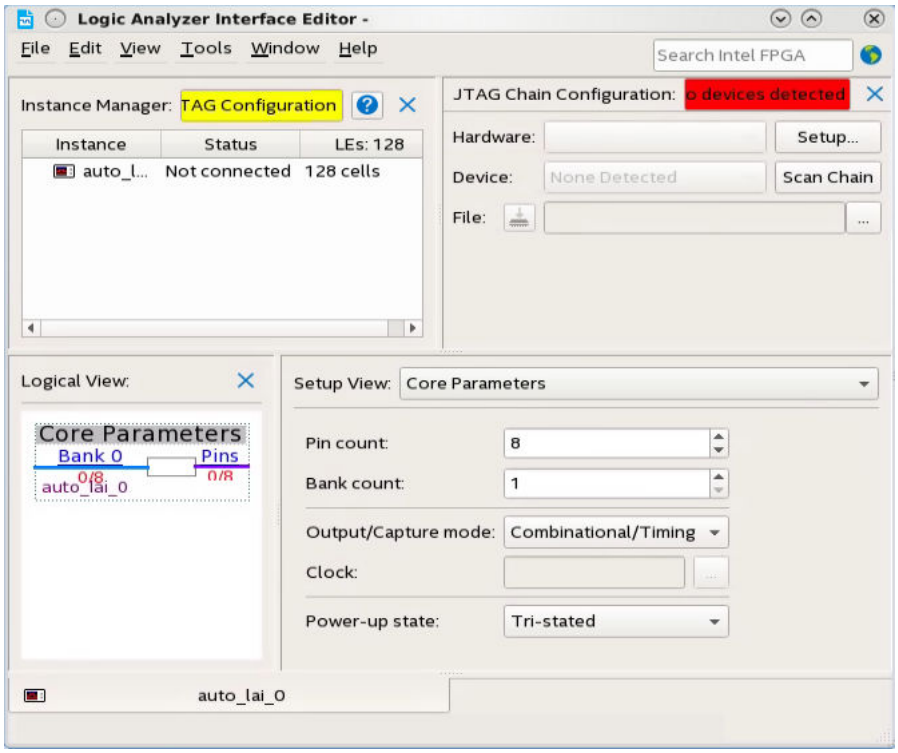
1. Configuration and control of the LAI using a computer loaded with the Intel Quartus Prime software via the JTAG port.
2. Configuration and control of the LAI using a third-party vendor logic analyzer via the JTAG port. Support varies by vendor.

7.3.1. 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 101. Logic Analyzer Interface Editor



- In the **Setup View** list, select **Core Parameters**.
- Specify the parameters of the LAI instance.

Related Information

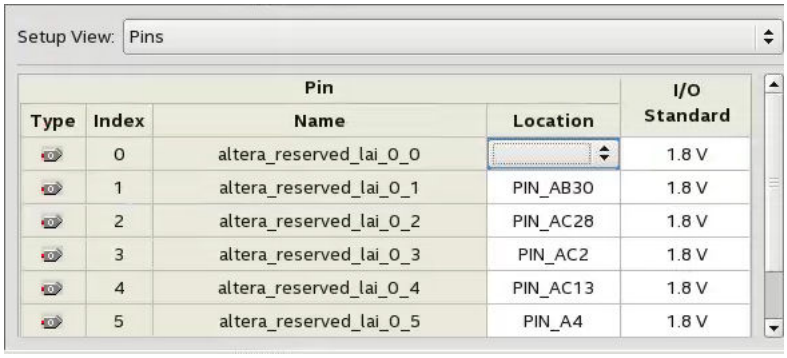
LAI Core Parameters on page 226

7.3.2. 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 102. 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

7.3.3. 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

7.3.4. 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, expand the entity hierarchy in the Project Navigator. If the LAI is compiled with your design, the `sld_hub` and `sld_multitap` entities are shown in the Project Navigator.

Figure 103. Project Navigator

Entity	Logic Cells	LC Registers
Stratix: EP1S10B672C7		
test	136 (1)	81
sld_multitap:auto_lai_0	35 (11)	15
sld_hub:sld_hub_inst	100 (25)	65

7.3.5. Programming Your Intel-Supported Device Using the LAI

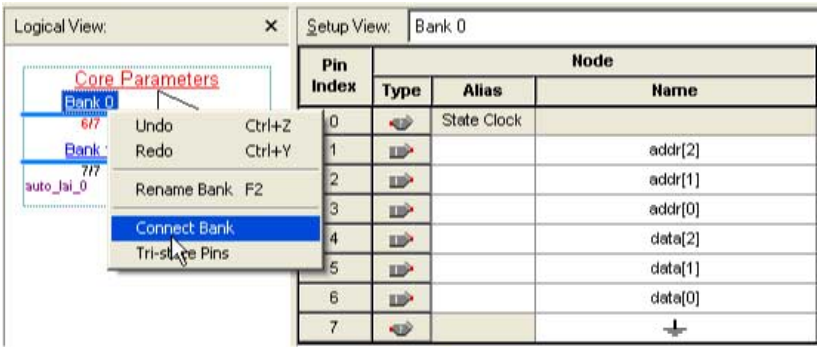
After compilation completes, you must configure your Intel-supported device before using the LAI.

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.

7.4. 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 104. Configuring Banks



7.4.1. 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.

7.5. Using the LAI with Incremental Compilation

The Incremental Compilation feature in the Intel Quartus Prime software allows you to preserve the synthesis and fitting results of your design. In cases where you only modify a portion of a design, or you wish to preserve the optimization results from a previous compilation, this ability allows reducing compilation times

Because LAI consists of only a small portion of your design, incremental compilation helps minimizing compilation time. Incremental compilation works best when you are only changing a small portion of your design. Incremental compilation yields an accurate representation of your design behavior when changing the `.lai` file through multiple compilations.

7.6. LAI Core Parameters

The table lists the LAI file core parameters:

Table 73. 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 223

7.7. In-System Debugging Using External Logic Analyzers Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
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 figure: LAI Instance in Compilation Report.
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

8. 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 7
- Design Debugging with the Signal Tap Logic Analyzer on page 146
- Megafunctions/LPM
  List of the types of memories and constants currently supported by the Intel Quartus Prime software

8.1. Setting Up In-System Modifiable Memories and Constants

When you specify that a memory or constant is run-time modifiable, the Intel Quartus Prime software changes the default implementation. A single-port RAM is converted to a dual-port RAM, and a constant is implemented in registers instead of look-up tables (LUTs). These changes enable run-time modification without changing the functionality of your design.

If you instantiate a memory or constant IP core directly with ports and parameters in VHDL or Verilog HDL, add or modify the `lpm_hint` parameter as follows:

In VHDL code, add the following:

```
lpm_hint => "ENABLE_RUNTIME_MOD = YES,
   INSTANCE_NAME = <instantiation name>";
```

In Verilog HDL code, add the following:

```
defparam <megafunction instance name>.lpm_hint =
   "ENABLE_RUNTIME_MOD = YES,
   INSTANCE_NAME = <instantiation name>";
```

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

8.2. Running the In-System Memory Content Editor

The In-System Memory Content Editor has three separate panes: the **Instance Manager**, the **JTAG Chain Configuration**, and the **Hex Editor**.

The **Instance Manager** pane displays all available run-time modifiable memories and constants in your FPGA device. The **JTAG Chain Configuration** pane allows you to program your FPGA and select the Intel FPGA device in the chain to update.

Using the In-System Memory Content Editor does not require that you open a project. 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 specific device selected in the **JTAG Chain Configuration** pane.

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 In-System Memory Content Editor can access the in-system memories and constants in a single device.

8.2.1. Instance Manager

You can read and write to in-system memory with the **Instance Manager** pane. When you scan the JTAG chain to update the **Instance Manager** pane, you can view a list of all run-time modifiable memories and constants in the design. The **Instance Manager** pane displays the Index, Instance, Status, Width, Depth, Type, and Mode of each element in the list.

Note: In addition to the buttons available in the **Instance Manager** pane, you can read and write data by selecting commands from the **Processing** menu, or the right-click menu in the **Instance Manager** pane or **Hex Editor** pane.

The status of each instance is also displayed beside each entry in the **Instance Manager** pane. The status indicates if the instance is **Not running**, **Offloading data**, or **Updating data**. The health monitor provides information about the status of the editor.

The Intel Quartus Prime software assigns a different index number to each in-system memory and constant to distinguish between multiple instances of the same memory or constant function. View the **In-System Memory Content Editor Settings** section of the Compilation Report to match an index number with the corresponding instance ID.

Related Information

Instance Manager Pane
In Intel Quartus Prime Help

8.2.2. Editing Data Displayed in the Hex Editor Pane

You can edit data read from your in-system memories and constants displayed in the **Hex Editor** pane by typing values directly into the editor or by importing memory files.

8.2.3. Importing and Exporting Memory Files

The In-System Memory Content Editor allows you to import and export data values for memories that have the In-System Updating feature enabled. Importing from a data file enables you to quickly load an entire memory image. Exporting to a data file enables you to save the contents of the memory for future use.

8.2.4. 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
- Command Line Scripting
- API Functions for Tcl
  In Intel Quartus Prime Help

8.2.5. 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.

8.2.6. Example: Using the In-System Memory Content Editor with the Signal Tap Logic Analyzer

The following scenario describes how you can use the In-System Updating of Memory and Constants feature with the Signal Tap Logic Analyzer to efficiently debug your design. You can use the In-System Memory Content Editor and the Signal Tap Logic Analyzer simultaneously with the JTAG interface.

Scenario: After completing your FPGA design, you find that the characteristics of your FIR filter design are not as expected.

1. To locate the source of the problem, change all your FIR filter coefficients to be in-system modifiable and instantiate the Signal Tap Logic Analyzer.
2. Using the Signal Tap Logic Analyzer to tap and trigger on internal design nodes, you find the FIR filter to be functioning outside of the expected cutoff frequency.
3. Using the **In-System Memory Content Editor**, you check the correctness of the FIR filter coefficients. Upon reading each coefficient, you discover that one of the coefficients is incorrect.
4. Because your coefficients are in-system modifiable, you update the coefficients with the correct data with the **In-System Memory Content Editor**.

In this scenario, you can quickly locate the source of the problem using both the In-System Memory Content Editor and the Signal Tap Logic Analyzer. You can also verify the functionality of your device by changing the coefficient values before modifying the design source files.

You can also modify the coefficients with the In-System Memory Content Editor to vary the characteristics of the FIR filter, for example, filter attenuation, transition bandwidth, cut-off frequency, and windowing function.

8.3. In-System Modification of Memory and Constants Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2018.05.07	18.0.0	• Removed obsolete example.
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

9. 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.

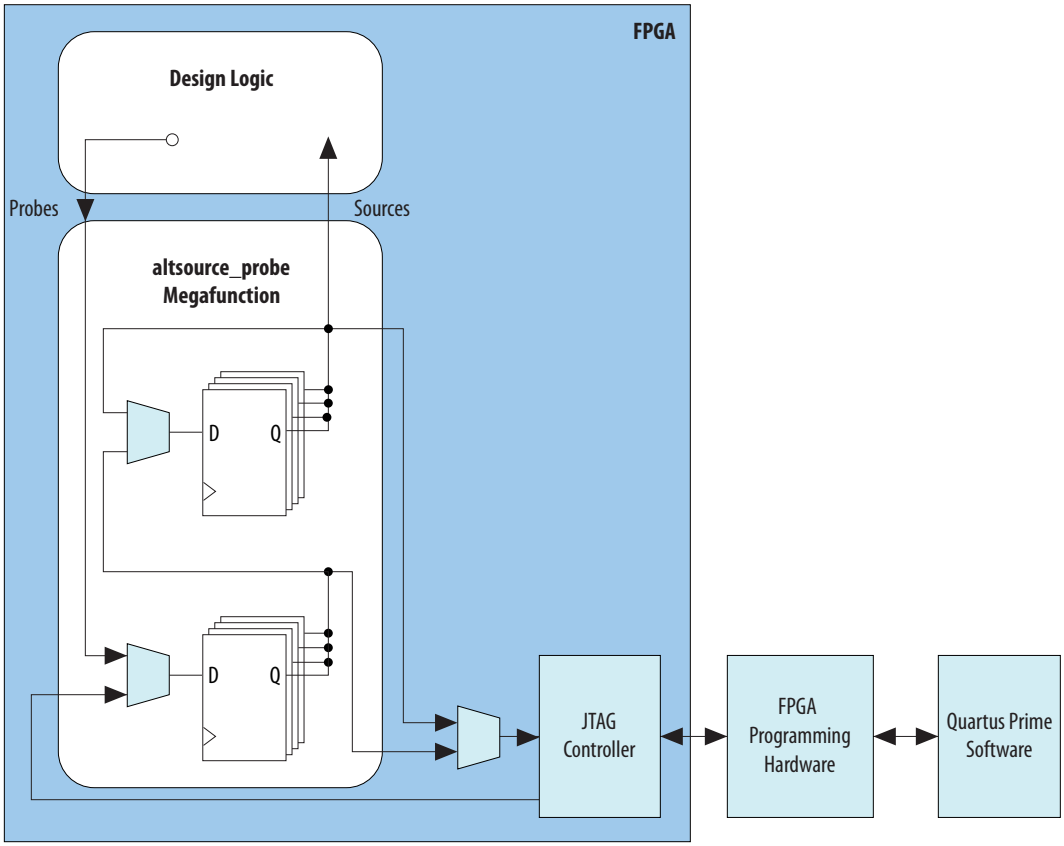
The Virtual JTAG IP core and the In-System Memory Content Editor also give you the capability to drive virtual inputs into your design. The Intel Quartus Prime software offers a variety of on-chip debugging tools.

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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 105. 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

9.1. 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:

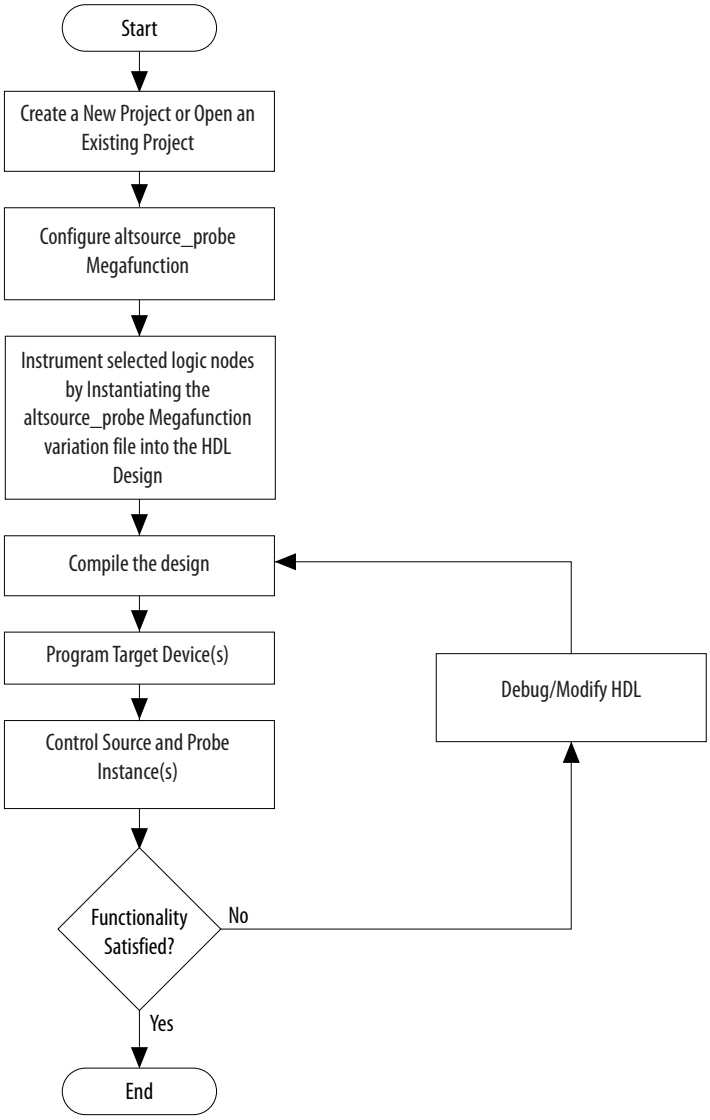
- Arria® series
- Stratix® series
- Cyclone® series
- MAX® series

9.2. 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 106. FPGA Design Flow Using the In-System Sources and Probes Editor



9.2.1. 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.

9.2.2. In-System Sources and Probes IP Core Parameters

Use the template to instantiate the variation file in your design.

Table 74. 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.

You can use the Intel Quartus Prime incremental compilation feature to reduce compilation time. Incremental compilation allows you to organize your design into logical partitions. During recompilation of a design, incremental compilation preserves the compilation results and performance of unchanged partitions and reduces design iteration time by compiling only modified design partitions.

9.3. Compiling the Design

When you compile your design that includes the In-System Sources and ProbesIP 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.

You can use the Intel Quartus Prime incremental compilation feature to reduce compilation design into logical partitions. During recompilation of a design, incremental compilation preserves the compilation results and performance of unchanged partitions and reduces design iteration time by compiling only modified design partitions.

9.4. 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.**

9.4.1. 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 a 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.

9.4.2. 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.

9.4.3. 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**.

9.4.4. 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.

9.4.4.1. 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.

9.4.4.2. 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**.

9.4.4.3. 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.

9.5. 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 75. In-System Sources and Probes Tcl Commands

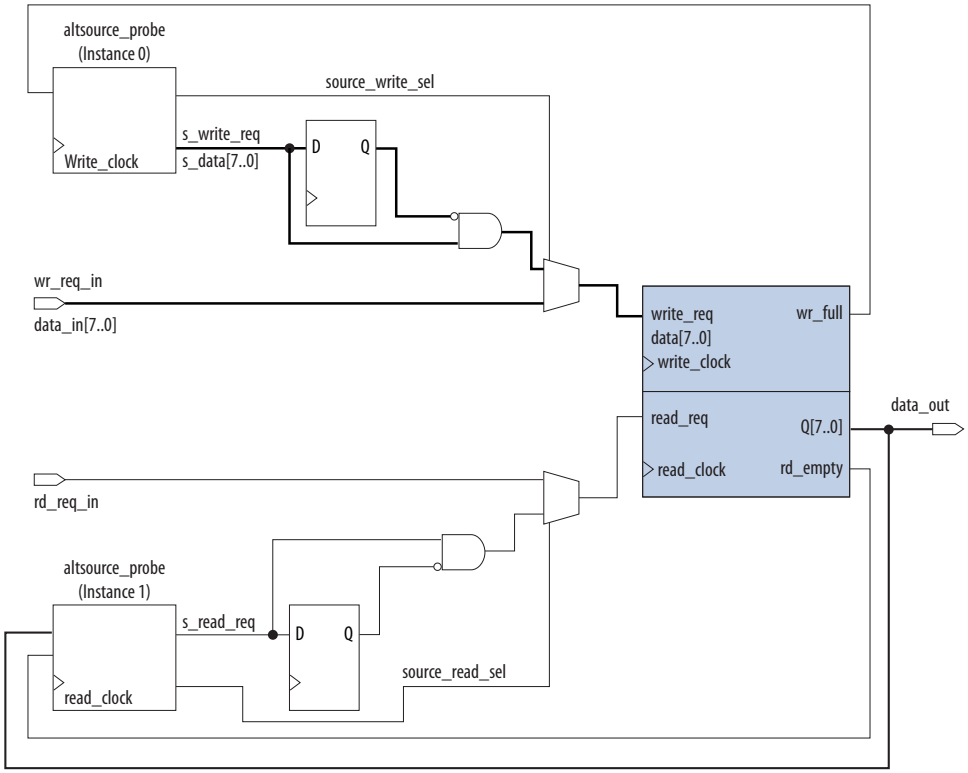
Command	Argument	Description
`start_insystem_source_probe`	`-device_name` *<device name>* `-hardware_name` *<hardware name>*	Opens a handle to a device with the specified hardware. Call this command before starting any transactions.
`get_insystem_source_probe_instance_info`	`-device_name` *<device name>* `-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 107. 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

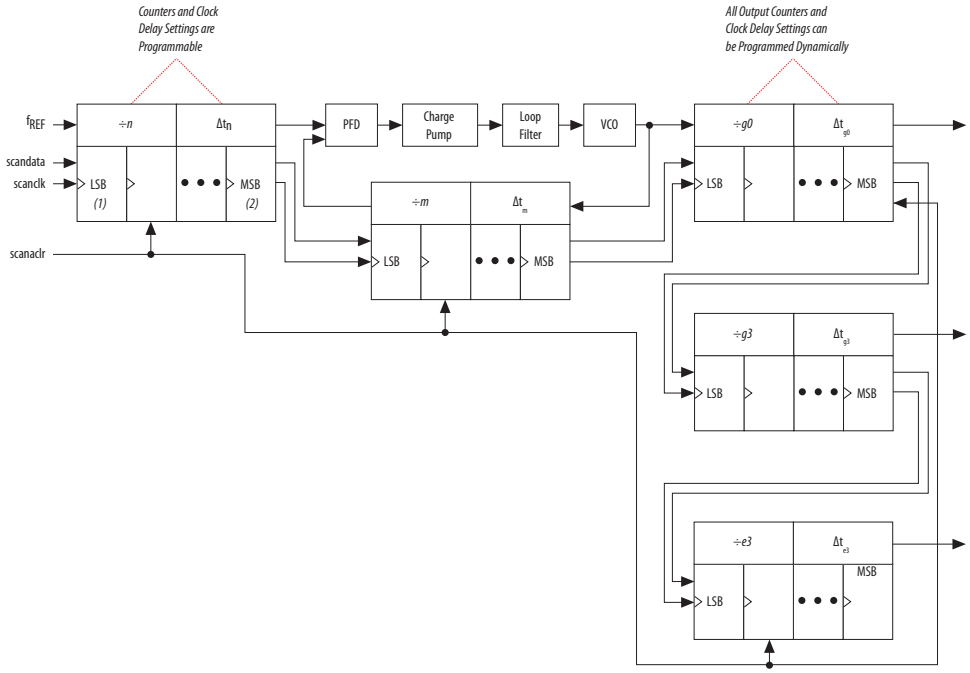
- Tcl Scripting
- Intel Quartus Prime Settings File Manual
- Command Line Scripting

9.6. 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.

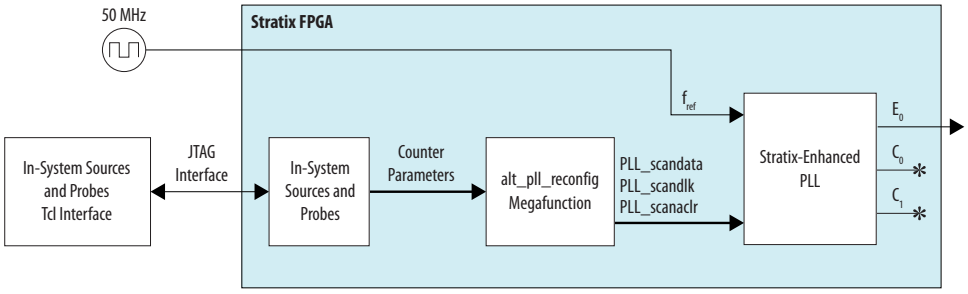
Stratix PLLs allow you to dynamically update PLL coefficients during run time. Each enhanced PLL within the Stratix device contains a register chain that allows you to modify the pre-scale counters (m and n values), output divide counters, and delay counters. In addition, the ALTPLL_RECONFIG IP core 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 108. 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.

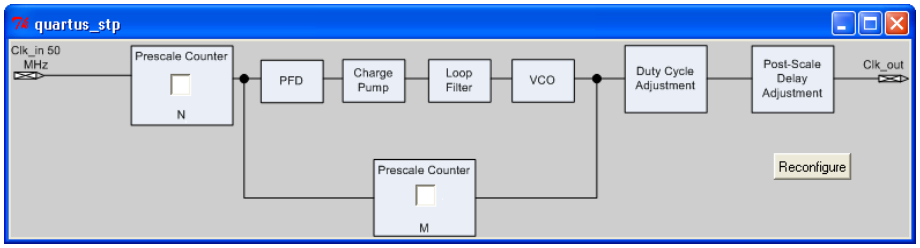
Figure 109. Block Diagram of Dynamic PLL Reconfiguration Design Example



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 110. 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

9.7. 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
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2018.05.07	18.0.0	Added details on finding the In-System Sources and Probes in the IP Catalog.
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

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Timing Analyzer

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Timing Analysis Introduction 3
1.1. Timing Analysis Basic Concepts 3
1.1.1. Timing Path and Clock Analysis 4
1.1.2. Clock Setup Analysis 7
1.1.3. Clock Hold Analysis 8
1.1.4. Recovery and Removal Analysis 9
1.1.5. Multicycle Path Analysis 10
1.1.6. Metastability Analysis 14
1.1.7. Timing Pessimism 15
1.1.8. Clock-As-Data Analysis 16
1.1.9. Multicorner Analysis 17
1.2. Document Revision History 19
2. Using the Intel Quartus Prime Timing Analyzer 20
2.1. Enhanced Timing Analysis for Intel Arria® 10 Devices 20
2.2. Basic Timing Analysis Flow 20
2.2.1. Step 1: Open a Project and Run the Fitter 20
2.2.2. Step 2: Specify Timing Constraints 21
2.2.3. Step 3: Specify General Timing Analyzer Settings 22
2.2.4. Step 4: Run Timing Analysis 24
2.2.5. Step 5: Analyze Timing Analysis Results 25
2.3. Using Timing Constraints 32
2.3.1. Recommended Initial SDC Constraints 32
2.3.2. SDC File Precedence 35
2.3.3. Iterative Constraint Modification 36
2.3.4. Creating Clocks and Clock Constraints 37
2.3.5. Creating I/O Constraints 50
2.3.6. Creating Delay and Skew Constraints 52
2.3.7. Creating Timing Exceptions 55
2.3.8. Example Circuit and SDC File 80
2.4. Timing Analyzer Tcl Commands 82
2.4.1. The quartus_sta Executable 82
2.4.2. Collection Commands 83
2.4.3. Identifying the Intel Quartus Prime Software Executable from the SDC File 87
2.5. Timing Analysis of Imported Compilation Results 87
2.6. Using the Intel Quartus Prime Timing Analyzer Document Revision History 87
A. Intel Quartus Prime Standard Edition User Guides 90

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 Intel® 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 required timing relationships are met for your design to correctly function, and confirms actual signal arrival times against the constraints that you specify. This use guide provides an introduction to basic timing analysis concepts, along with step-by-step instructions for using the Intel Quartus Prime Timing Analyzer.

1.1. 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 input/output elements. In Intel 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.
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	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.
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.

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Term	Definition
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 setup or hold relationship 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 design nodes, such as the output of a register to the input of another register.

1.1.1. Timing Path and Clock Analysis

The Timing Analyzer measures the timing performance for all timing paths identified in your design. The Timing Analyzer requires a timing netlist that describes your design's nodes and connections for analysis. The Timing Analyzer 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.

1.1.1.1. 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. You can generate the timing netlist in the Timing Analyzer any time after running the Fitter or full compilation.

The following figures illustrate how the timing netlist divides the design elements into cells, pins, nets, and ports for measurement of delay.

Figure 1. Simple Design Schematic

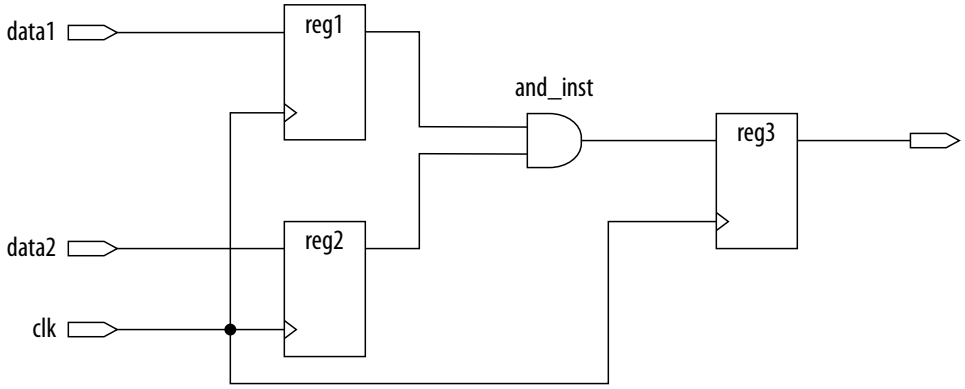
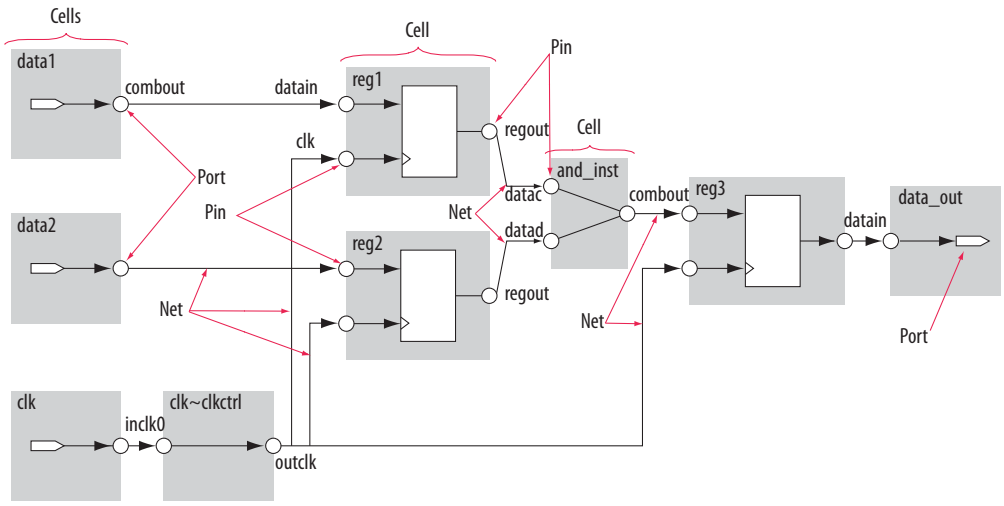


Figure 2. Division of Simple Design Schematic Elements in Timing Netlist



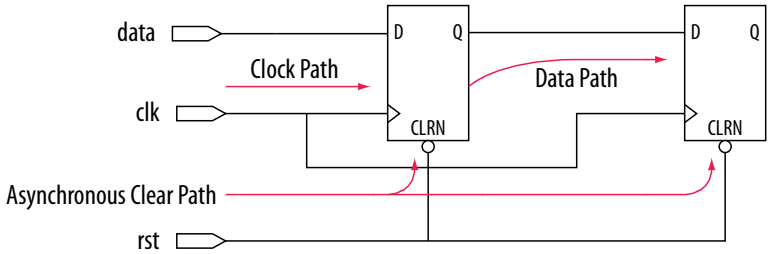
1.1.1.2. Timing Paths

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.

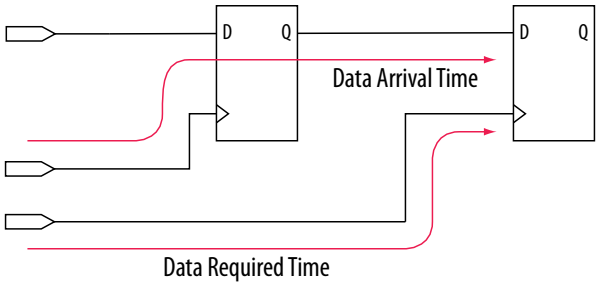
1.1.1.3. Data and Clock Arrival Times

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, including any clock port buffer delays, and subtracts the micro setup time (μt_{SU}) 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

$$\text{Data Arrival Time} = \text{Launch Edge} + \text{Source Clock Delay} + \mu t_{co} + \text{Register-to-Register Delay}$$
$$\text{Data Required Time} = \text{Latch Edge} + \text{Destination Clock Delay} - \mu t_{SU}$$

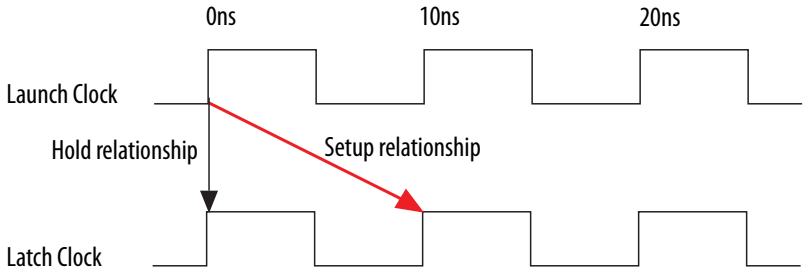
1.1.1.4. Launch and Latch Edges

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.



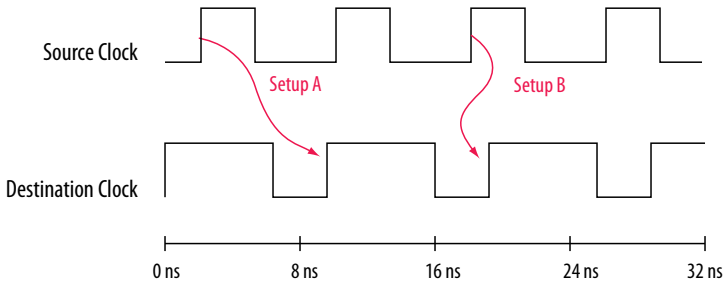
You create define constraints for all clocks and assign the constraints to nodes in your design. These clock constraints provide the structure required for repeatable data relationships. If you do not constrain the clocks in your design, the Intel Quartus Prime software analyzes in terms of a 1 GHz clock to maximize timing based Fitter effort. To ensure realistic slack values, you must constrain all clocks in your design with real values.

1.1.2. 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 timing requirement is met or not met. Positive slack indicates the margin by which a requirement is met; negative slack indicates the margin by which a requirement is not met.

Figure 8. Clock Setup Slack for Internal Register-to-Register Paths

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Arrival Time	=	Launch Edge + Clock Network Delay to Source Register + μt_{CO} + Register-to-Register Delay
Data Required Time	=	Latch Edge + Clock Network Delay to Destination Register – μt_{SU} – Setup Uncertainty

The Timing Analyzer performs setup checks using the maximum delay when calculating data arrival time, and minimum delay when calculating data required time.

Figure 9. Clock Setup Slack from Input Port to Internal Register

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Arrival Time	=	Launch Edge + Clock Network Delay + Input Maximum Delay + Port-to-Register Delay
Data Required Time	=	Latch Edge + Clock Network Delay to Destination Register – μt_{SU} – Setup Uncertainty

Figure 10. Clock Setup Slack from Internal Register to Output Port

Clock Setup Slack	=	Data Required Time – Data Arrival Time
Data Required Time	=	Latch Edge + Clock Network Delay to Output Port – Output Maximum Delay
Data Arrival Time	=	Launch Edge + Clock Network Delay to Source Register +μt_{CO} + Register-to-Port Delay

1.1.3. 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

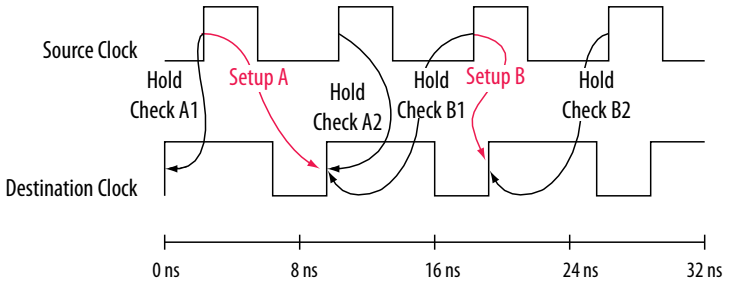


Figure 12. Clock Hold Slack for Internal Register-to-Register Paths

Clock Hold Slack = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + μt_{CO} + Register-to-Register Delay
Data Required Time = Latch Edge + Clock Network Delay to Destination Register + μt_{H} + 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

Clock Hold Slack = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay + Input Minimum Delay + Pin-to-Register Delay
Data Required Time = Latch Edge + Clock Network Delay to Destination Register + μt_{H}

Figure 14. Clock Hold Slack Calculation from Internal Register to Output Port

Clock Hold Slack = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + μt_{CO} + Register-to-Pin Delay
Data Required Time = Latch Edge + Clock Network Delay – Output Minimum Delay

1.1.4. 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

Recovery Slack Time = Data Required Time – Data Arrival Time
Data Required Time = Latch Edge + Clock Network Delay to Destination Register – μt_{SU}
Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + μt_{CO} + Register-to-Register Delay

Figure 16. Recovery Slack Calculation if the Asynchronous Control Signal is not Registered

Recovery Slack Time = Data Required Time – Data Arrival Time
Data Required Time = Latch Edge + Clock Network Delay to Destination Register – μt_{SU}
Data Arrival Time = Launch Edge + Clock Network Delay + Input Maximum Delay + Port-to-Register Delay

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 asynchronous control signals.

Figure 17. Removal Slack Calculation if the Asynchronous Control Signal is Registered

Removal Slack Time = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay to Source Register + μt_{CO} of Source Register + Register-to-Register Delay
Data Required Time = Latch Edge + Clock Network Delay to Destination Register + μt_{H}

Figure 18. Removal Slack Calculation if the Asynchronous Control Signal is not Registered

Removal Slack Time = Data Arrival Time – Data Required Time
Data Arrival Time = Launch Edge + Clock Network Delay + Input Minimum Delay of Pin + Minimum Pin-to-Register Delay
Data Required Time = Latch Edge + Clock Network Delay to Destination Register + μt_{H}

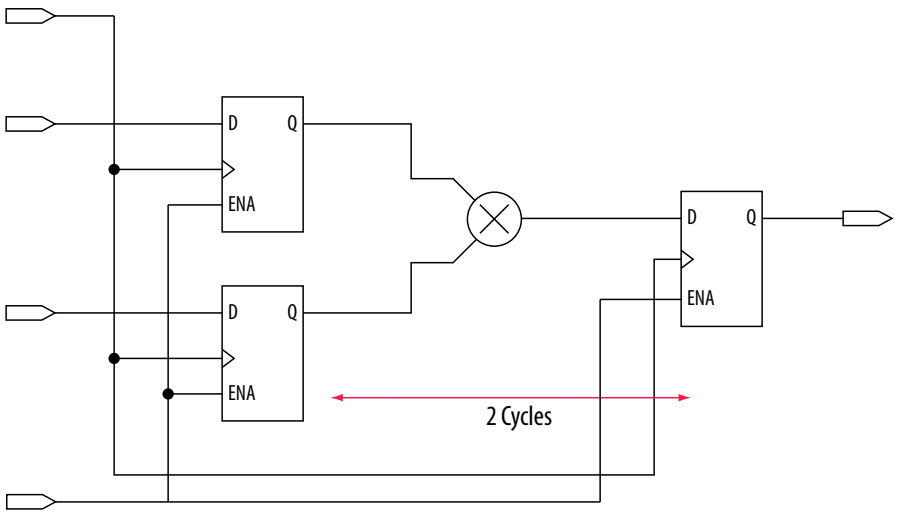
If the asynchronous reset signal is from a device pin, you must assign the **Input Minimum Delay** timing assignment to the asynchronous reset pin for the Timing Analyzer to perform removal analysis on the path.

1.1.5. Multicycle Path Analysis

Multicycle paths are data paths that require either a non-default setup or hold relationship, for proper analysis.

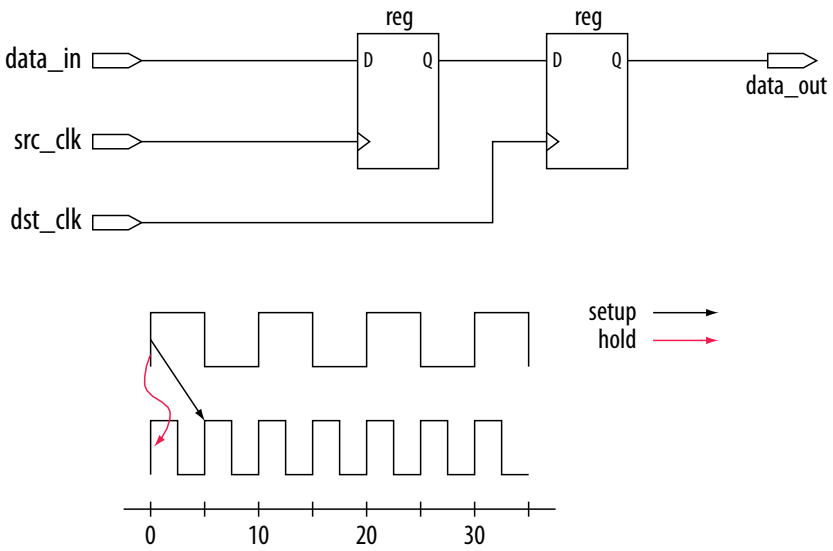
For example, a register may be required to capture data on every second or third rising clock edge. An example of a multicycle path between the input registers of a multiplier and an output register where the destination latches data on every other clock edge.

Figure 19. Multicycle Path



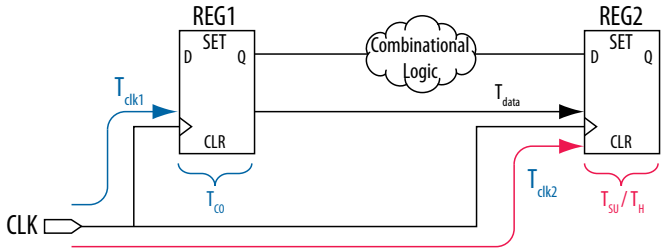
A register-to-register path used for the default setup and hold relationship, the respective timing diagrams for the source and destination clocks, and the default setup and hold relationships, when the source clock, `src_clk`, has a period of 10 ns and the destination clock, `dst_clk`, has a period 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



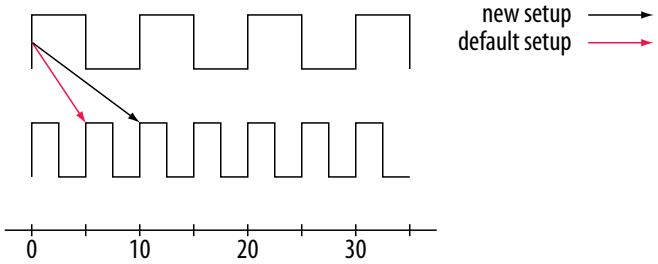
To accommodate the system requirements you can modify the default setup and hold relationships by specifying a multicycle timing constraint to a register-to-register path.

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



1.1.5.1. 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, which results in the hold relationship being equal to one clock period (launch edge – latch edge).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.

The calculation the Timing Analyzer performs to determine the hold check.

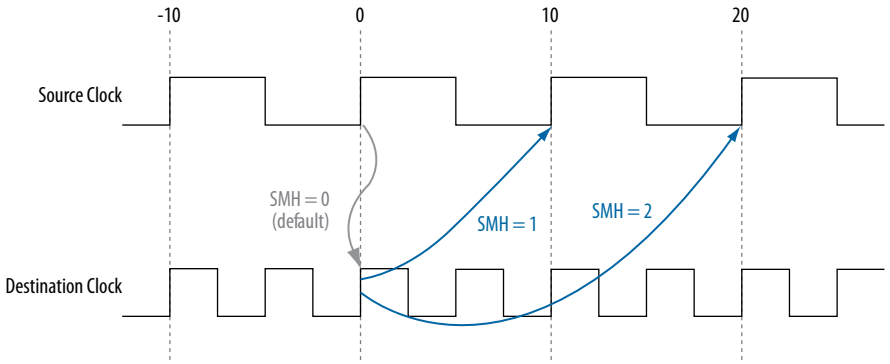
Figure 23. Hold Check

$$\text{hold check 1} = \text{current launch edge} - \text{previous latch edge}$$
$$\text{hold check 2} = \text{next launch edge} - \text{current latch edge}$$

Tip: If a hold check overlaps a setup check, the hold check is ignored.

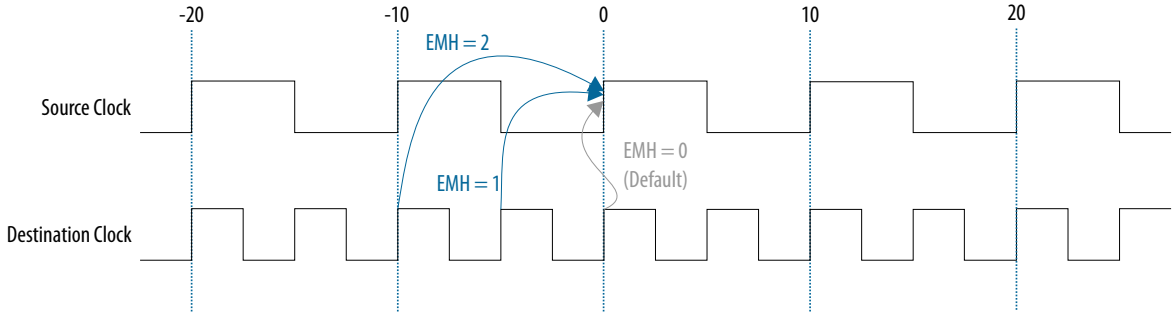
A start multicycle hold assignment modifies the launch edge of the destination clock by moving the latch edge the number of clock periods you specify to the right of the default launch edge. The following figure shows various values of the start multicycle hold (SMH) assignment and the resulting launch edge.

Figure 24. Start Multicycle Hold Values



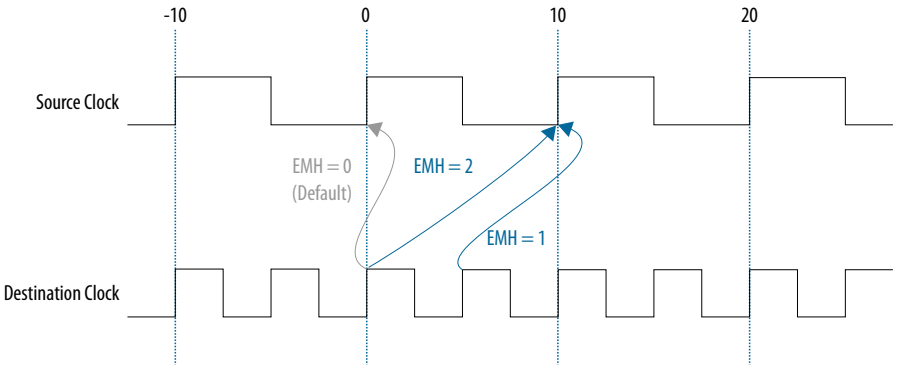
An end multicycle hold assignment modifies the latch edge of the destination clock by moving the latch edge the specific number of clock periods to the left of the default latch edge. The following figure shows various values of the end multicycle hold (EMH) assignment and the resulting latch edge.

Figure 25. End Multicycle Hold Values



The following shows the hold relationship the Timing Analyzer reports for the negative hold relationship:

Figure 26. End Multicycle Hold Values the Timing Analyzer Reports

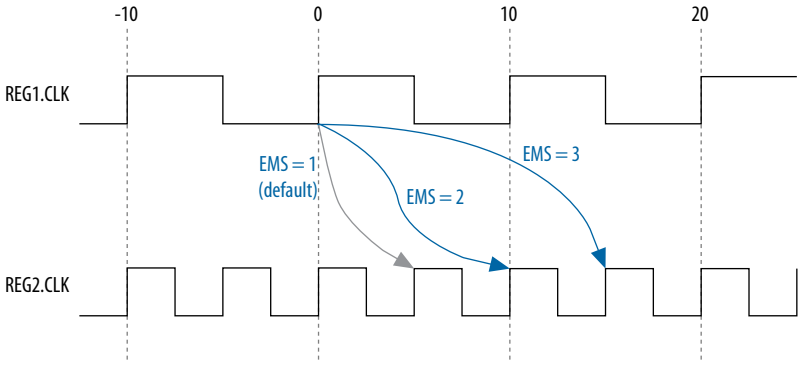


1.1.5.2. Multicycle Clock Setup

The setup relationship is defined as 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.

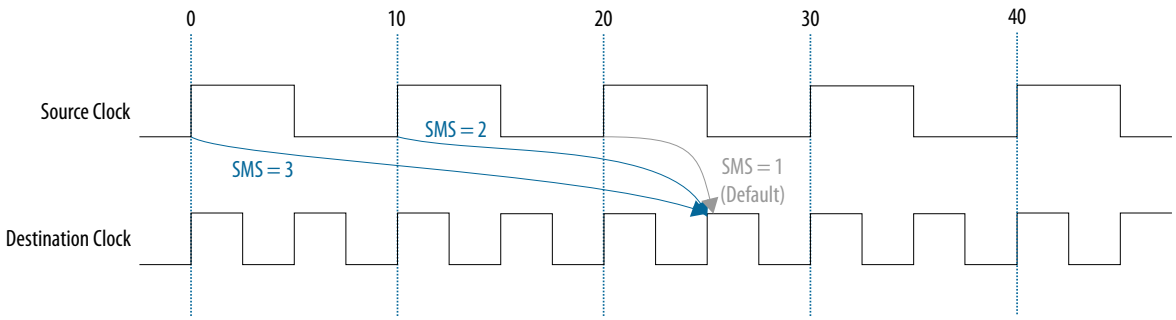
An end multicycle setup assignment modifies the latch edge of the destination clock by moving the latch edge the specified number of clock periods to the right of the determined default latch edge. The following figure shows various values of the end multicycle setup (EMS) assignment and the resulting latch edge.

Figure 27. End Multicycle Setup Values



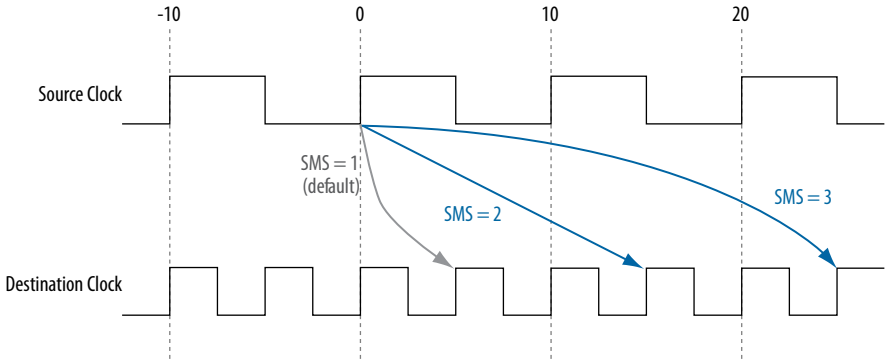
A start multicycle setup assignment modifies the launch edge of the source clock by moving the launch edge the specified number of clock periods to the left of the determined default launch edge. A start multicycle setup (SMS) assignment with various values can result in a specific launch edge.

Figure 28. Start Multicycle Setup Values



The setup relationship reported by the Timing Analyzer for the negative setup relationship.

Figure 29. Start Multicycle Setup Values Reported by the Timing Analyzer



1.1.6. 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 Intel Quartus Prime software also protects these registers from optimizations that might negatively impact MTBF, such as register duplication and logic retiming. The Intel Quartus Prime software can also optimize the MTBF of your design if the MTBF is too low.

Related Information

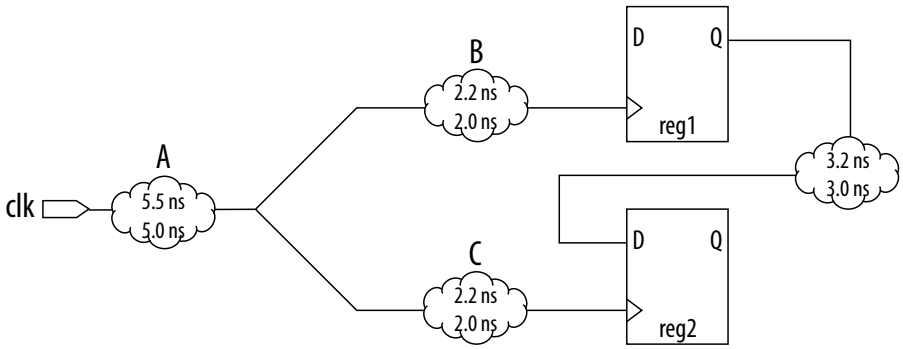
Understanding Metastability in FPGAs

1.1.7. 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 two different delay values are used for the same clock path. For example, in a simple setup analysis, the maximum clock path delay to the source register is used to determine the data arrival time. The minimum clock path delay to the destination register is used to determine 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 are used to 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



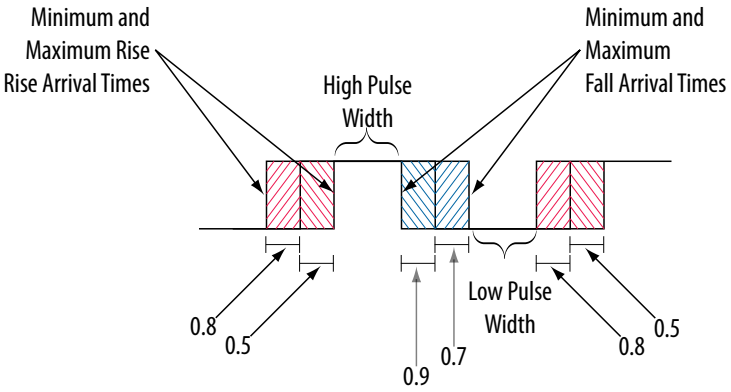
Segment A is the common clock path between `reg1` and `reg2`. The minimum delay is 5.0 ns; the maximum delay is 5.5 ns. The difference between the maximum and minimum delay value equals the common clock path pessimism removal value; in this case, the common clock path pessimism is 0.5 ns. The Timing 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 to be recognized 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 being latched 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 specified 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 be increased 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 be increased 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).

1.1.8. Clock-As-Data Analysis

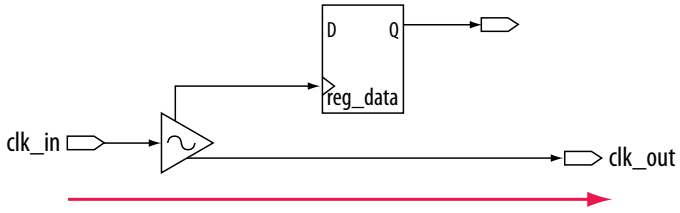
The majority of FPGA designs contain simple connections between any two nodes known as either a data path or a clock path.

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. Clock-as-data analysis is performed in circuits with elements such as clock dividers and DDR source-synchronous outputs.

The connection between the input clock port and output clock port can be treated either as a clock path or a data path. 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`.

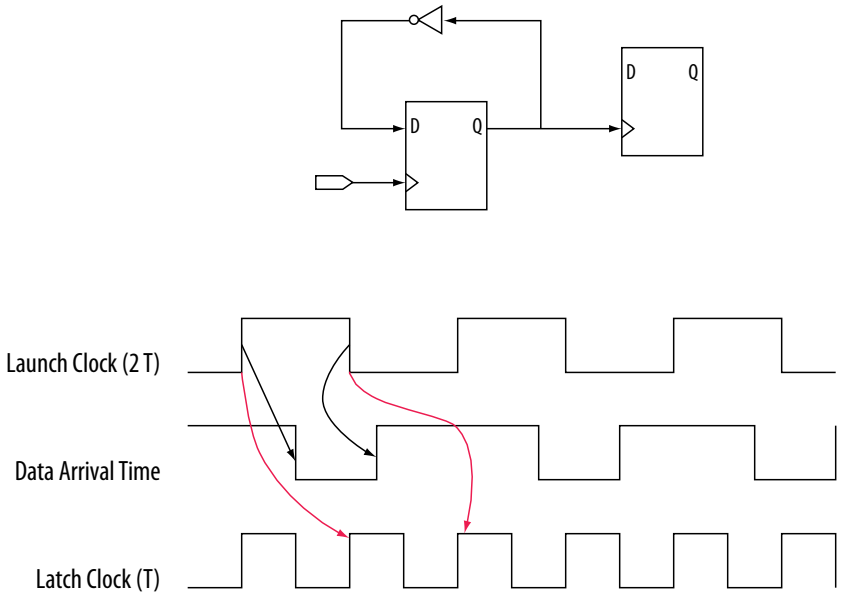
Figure 32. Simplified Source Synchronous Output



With clock-as-data analysis, the Timing Analyzer provides a more accurate analysis of the path based on user constraints. For the clock path analysis, any phase shift associated with the phase-locked loop (PLL) is taken into consideration. For the data path analysis, any phase shift associated with the PLL is taken into consideration rather than ignored.

The clock-as-data analysis also applies to internally generated clock dividers. An internally generated clock divider. In this figure, waveforms are for the inverter feedback path, analyzed during timing analysis. The output of the divider register is used to determine the launch time and the clock port of the register is used to determine the latch time.

Figure 33. Clock Divider



1.1.9. Multicorner Analysis

The Timing Analyzer performs multicorner timing analysis to verify your design under a variety of operating conditions—such as voltage, process, and temperature—while performing static timing analysis.

To change the operating conditions or speed grade of the current device for timing analysis, use the `set_operating_conditions` command.

If you specify an operating condition Tcl object, the `-model`, `-speed`, `-temperature`, and `-voltage` options are available. If you do not specify an operating condition Tcl object, Tcl requires the `-model` option. `-speed`, `-temperature`, and `-voltage` are optional.

Tip: To obtain a list of available operating conditions for the target device, use the `get_available_operating_conditions -all` command.

To ensure that no violations occur under various conditions during the device operation, perform static timing analysis under all available operating conditions.

Table 2. Operating Conditions for Slow and Fast Models

Model	Speed Grade	Voltage	Temperature
Slow	Slowest speed grade in device density	V_{cc} minimum supply (1)	Maximum T_J (1)
Fast	Fastest speed grade in device density	V_{cc} maximum supply (1)	Minimum T_J (1)

Note :
1. Refer to the DC & Switching Characteristics chapter of the applicable device Handbook for V_{cc} and T_J. values

In your design, you can set the operating conditions for to the slow timing model, with a voltage of 1100 mV, and temperature of 85° C with the following code:

```
set_operating_conditions -model slow -temperature 85 -voltage 1100
```

You can set the same operating conditions with a Tcl object:

```
set_operating_conditions 3_slow_1100mv_85c
```

The following block of code shows how to use the `set_operating_conditions` command to generate different reports for various operating conditions.

Example 1. Script Excerpt for Analysis of Various Operating Conditions

```
#Specify initial operating conditions
set_operating_conditions -model slow -speed 3 -grade c -temperature 85 -voltage
1100
#Update the timing netlist with the initial conditions
update_timing_netlist
#Perform reporting
#Change initial operating conditions. Use a temperature of 0C
set_operating_conditions -model slow -speed 3 -grade c -temperature 0 -voltage
1100
#Update the timing netlist with the new operating condition
update_timing_netlist
#Perform reporting
#Change initial operating conditions. Use a temperature of 0C and a model of fast
set_operating_conditions -model fast -speed 3 -grade c -temperature 0 -voltage
1100
#Update the timing netlist with the new operating condition
update_timing_netlist
#Perform reporting
```

1.2. Document Revision History

Table 3. Document Revision History

Date	Version	Changes
2018.09.24	18.1.0	• Minor text enhancements for clarity and style.
2016.05.02	16.0.0	Corrected typo in Fig 6-14: Clock Hold Slack Calculation from Internal Register to Output Port
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
2014.12.15	14.1.0	Moved Multicycle Clock Setup Check and Hold Check Analysis section from the Timing Analyzer chapter.
June 2014	14.0.0	Updated format
June 2012	12.0.0	Added social networking icons, minor text updates
November 2011	11.1.0	Initial release.

2. Using the Intel Quartus Prime Timing Analyzer

The Intel 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.

Related Information

Intel FPGA Technical Training

2.1. Enhanced Timing Analysis for Intel Arria® 10 Devices

The Timing Analyzer supports new timing algorithms for the Intel Arria® 10 device family which significantly improve the speed of the analysis.

These algorithms are enabled by default for Intel Arria 10 devices, and can be enabled for earlier families with an assignment. The new analysis engine analyzes the timing graph a fixed number of times. Previous Timing Analyzer analysis analyzed the timing graph as many times as there were constraints in your Synopsys Design Constraint (SDC) file.

The new algorithms also support incremental timing analysis, which allows you to modify a single block and re-analyze while maintaining a fully analyzed design.

You can turn on the new timing algorithms for use with Arria V, Cyclone® V, and Stratix V devices with the following QSF assignment:

```
set_global_assignment -name TIMEQUEST2 ON
```

2.2. Basic Timing Analysis Flow

The Intel Quartus Prime Timing Analyzer performs constraint validation and reports timing performance as part of the full compilation flow. After creating your design and setting up a project, you define the required timing parameters (that is, constraints) for your design in a Synopsys* Design Constraints (`.sdc`) file. The Fitter attempts to place logic to meet or exceed the constraints you specify. The Timing Analyzer reports conditions that do not meet your constraints, allowing you to locate and correct critical timing issues. The following steps describe the basic timing analysis flow in the Intel Quartus Prime software.

2.2.1. Step 1: Open a Project and Run the Fitter

Before running timing analysis, you must open an Intel Quartus Prime project and run the Fitter to elaborate the design hierarchy, synthesize logic, and perform place and route.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

1. Click **File > New Project Wizard** to create a new project, or click **File ➤ Open Project** to open an existing project.
2. To run the Fitter (and any prerequisite Compiler modules), click **Processing ➤ Start ➤ Start Fitter**.

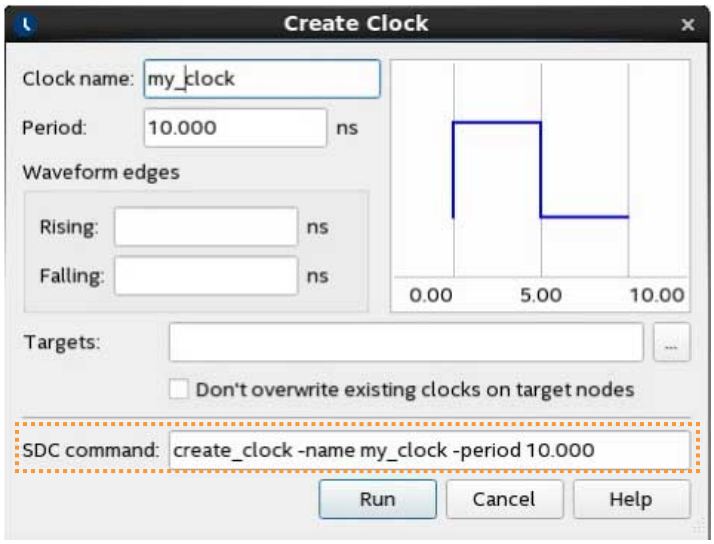
2.2.2. Step 2: Specify Timing Constraints

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 during timing analysis. You define timing constraints in one or more Synopsys Design Constraints (`.sdc`) files that you add to the project.

If you are unfamiliar with `.sdc` files, you can create an initial `.sdc` file in the Timing Analyzer GUI, or with provided `.sdc` file templates. If you are familiar with timing analysis, you can create an `.sdc` file in any text editor, and then add the file to the project.

1. Use any combination of the following to enter the timing constraints for your design in an `.sdc` file:
   - Enter constraints in the Timing Analyzer GUI—click **Tools ➤ Timing Analyzer**, click **Update Timing Netlist**, and then enter constraints from the Constraints menu. The GUI displays the corresponding SDC command that applies.
   - Create an `.sdc` file on your own. You can start by adding the Recommended Initial SDC Constraints on page 32, and then iteratively modify .sdc constraints and reanalyze the timing results. You must first create clock constraints before entering any constraints dependent on the clock.

Figure 34. Create Clock Dialog Defines Clock Constraints



2. Save the `.sdc` file. When entering constraints in the Timing Analyzer GUI, click **Constraints ➤ Write SDC File** to save the constraints you enter in the GUI to an `.sdc` file.
3. Add the `.sdc` file to your project, as Step 3: Specify General Timing Analyzer Settings on page 22 describes.

2.2.3. Step 3: Specify General Timing Analyzer Settings

Before running timing analysis, you can consider and optionally specify the following Timing Analyzer and Compiler settings that have an impact on the analysis results:

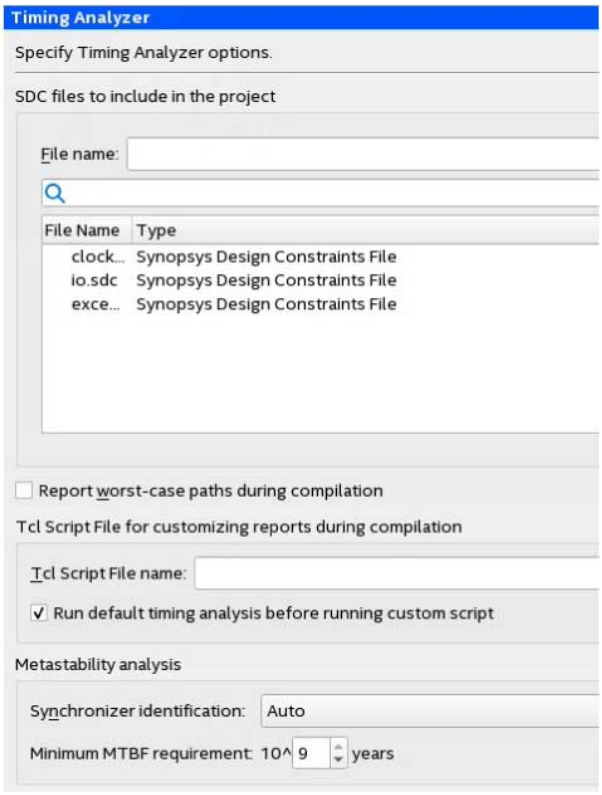
Table 4. Timing Analyzer and Compiler Settings

Setting	Description	Location
SDC files to include in the project	Specifies the name and order of Synopsis Design Constraint (`.sdc`) files in the project.	**Assignments ➤ Settings ➤ Timing Analyzer**
Report worst-case paths during compilation	Displays summary of the worst-case timing paths in the design.	**Assignments ➤ Settings ➤ Timing Analyzer**
Tcl Script File name	Specifies the file name for a custom analysis script. You can specify whether to **Run default timing analysis before running the custom script**.	**Assignments ➤ Settings ➤ Timing Analyzer**
Metastability analysis	Specifies how the Timing Analyzer identifies registers as being part of a synchronization register chain for metastability analysis.	**Assignments ➤ Settings ➤ Timing Analyzer**
Enable multicorner support for Timing Analyzer and EDA Netlist Writer	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	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**

continued...

Setting	Description	Location
SDC Constraint Protection	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	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.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis)**
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.	**Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter)**

Figure 35. Timing Analyzer Settings



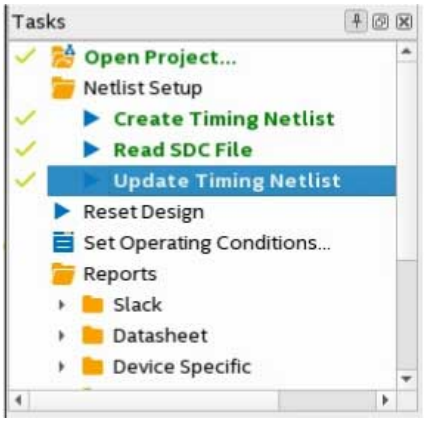
2.2.4. Step 4: Run Timing Analysis

After specifying initial timing constraints, you can run the Fitter or full compilation to generate the timing netlist and run the Timing Analyzer. During compilation, the Fitter attempts to place logic of your design to comply with the timing constraints that you specify. The Timing Analyzer reports the margin (slack) by which your design meets or fails each constraint.

1. To generate the timing netlist, perform either of the following:
   - To run full compilation that includes timing analysis, click **Processing ➤ Start Compilation**. The Timing Analyzer automatically performs multi-corner signoff timing analysis after the Fitter completes.

     Or
   - To run the Fitter, click **Processing ➤ Start ➤ Start Fitter**.
2. To launch the Timing Analyzer, click **Tools ➤ Timing Analyzer**.
3. In the **Tasks** pane, double-click **Update Timing Netlist**. The Timing Analyzer loads the timing netlist, reads all of the project's `.sdc` files, and generates a default set of timing reports, including the Timing Analyzer Summary and Advanced I/O Timing reports.

Figure 36. Timing Analyzer Tasks



4. In the In the **Tasks** pane, under **Reports**, double-click any individual task to generate the report and analyze the results, as Step 5: Analyze Timing Analysis Results on page 25 describes.

Related Information

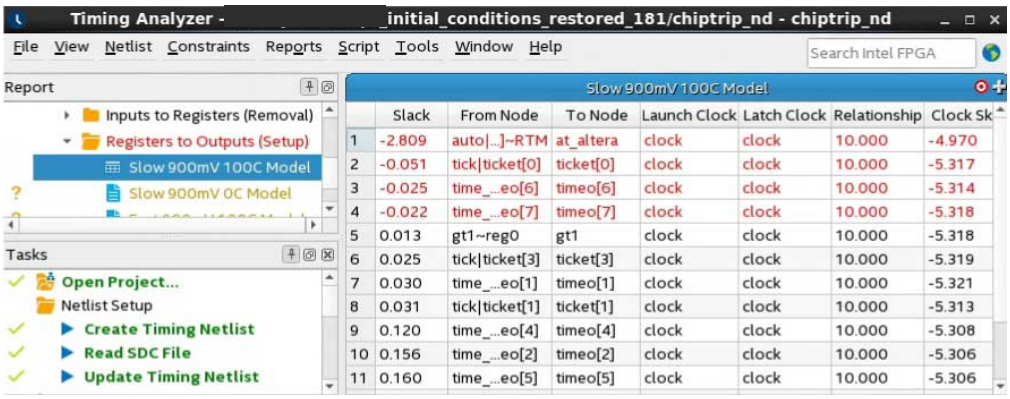
- Timing Analysis of Imported Compilation Results on page 87
- Timing Analyzer Tcl Commands on page 82
- Basic Timing Analysis Flow on page 20
- The quartus_sta Executable on page 82

2.2.5. Step 5: Analyze Timing Analysis Results

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 the design meets the constraint.

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 37. Timing Analyzer Shows Failing Paths in Red



Reports that indicate failing timing performance appear in red text, 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.

2.2.5.1. Timing Report Commands

The Timing Analyzer generates only a subset of all available reports by default, including the Timing Analyzer Summary report. However, you can generate many other detailed reports in the Timing Analyzer GUI, or with command-line commands. You can customize the display of information in the reports.

Table 5. Timing Analyzer Report Generation Command Summary

Timing Analyzer Tasks Pane GUI	Command-Line	Generates
Custom Reports ➤ Report Timing	`report_timing`	Timing report
Custom Reports ➤ Report Exceptions	`report_exceptions`	Exceptions report
Diagnostic ➤ Report Clock Transfers	`report_clock_transfers`	Clock Transfers report
Slack ➤ Report Minimum Pulse Width Summary	`report_min_pulse_width`	Minimum Pulse Width Summary report
Diagnostic ➤ Report Unconstrained Paths	`report_ucp`	Unconstrained Paths report

2.2.5.2. Set Operating Conditions

You can specify various operating conditions to analyze timing under different power and temperature ranges. There are four operating conditions available that represent the four "timing corners" in multi-corner timing analysis. The Timing Analyzer generates separate reports for each set of operating conditions.

- **Slow 900mV 100C Model**—specifies low voltage, high temperature operating conditions for timing analysis.
- **Slow 900mV 0C Model**—specifies low voltage, low temperature operating conditions for timing analysis.
- **Fast 900mV 100C Model**—specifies high voltage, high temperature operating conditions for timing analysis.
- **Fast 900mV 100C Model**—specifies high voltage, low temperature operating conditions for timing analysis.

Select a voltage/temperature combination and double-click **Report Timing** under **Custom Reports** in the **Tasks** pane to generate timing analysis reports for that model. After generating the report for that model, you can double-click the listings for the other models to generate analysis for those reports without re-generating the timing netlist.

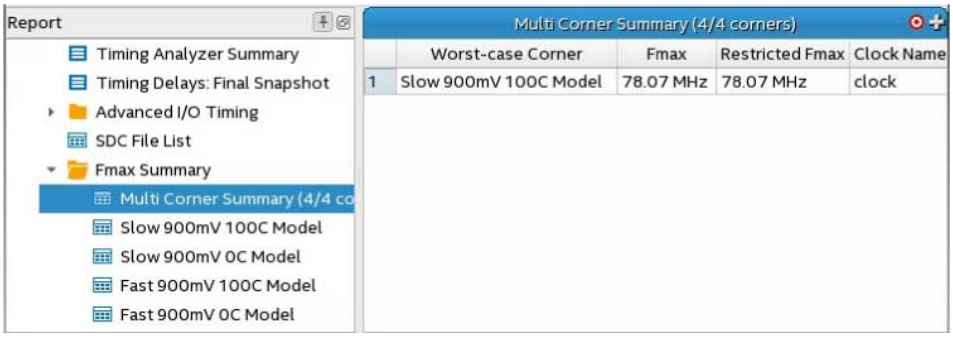
You can use the following context menu options to generate or regenerate reports in the **Report** window:

- **Regenerate**—Regenerates the report you select.
- **Generate in All Corners**—Generate a timing report using all four corners.
- **Regenerate All Out of Date**—Regenerate all reports.
- **Delete All Out of Date**—Removes all previous report data.

2.2.5.3. Fmax Summary Report

The Fmax Summary Report panel lists the maximum frequency of each clock in your design.

Figure 38. Fmax Summary Report



In some cases the Fmax Summary may indicate a "Limit due to hold check." Typically, hold checks do not limit the maximum frequency (f_{MAX}) because these checks are for same-edge relationships, and therefore independent of clock frequency. For example, when launch equals zero and latch equals zero.

However, if you have an inverted clock transfer, or a multicycle transfer (such as setup=2, hold=0), then the hold relationship is no longer a same-edge transfer and changes as the clock frequency changes.

The value in the **Restricted Fmax** column incorporates limits due to hold time checks, as well as minimum period and pulse width checks. If hold checks limit the f_{MAX} more than setup checks, that is indicated in the **Note** column as "Limit due to hold check."

2.2.5.4. Report Timing Command

The **Report Timing** command allows you to specify options for reporting the timing on any path or clock domain in the design.

To access **Report Timing** in the Timing Analyzer:

- In the **Tasks** pane, click **Reports ➤ Custom Reports ➤ Report Timing**.
- Right-click on nodes or assignments, and then click **Report Timing**.

You can specify the **Clocks**, **Targets**, **Analysis Type**, and **Output** options that you want to include in the report. For example, you can increase the number of paths to report, add a **Target** filter, add a **From Clock**, or write the report to a text file.

Figure 39. Report Timing Dialog Box

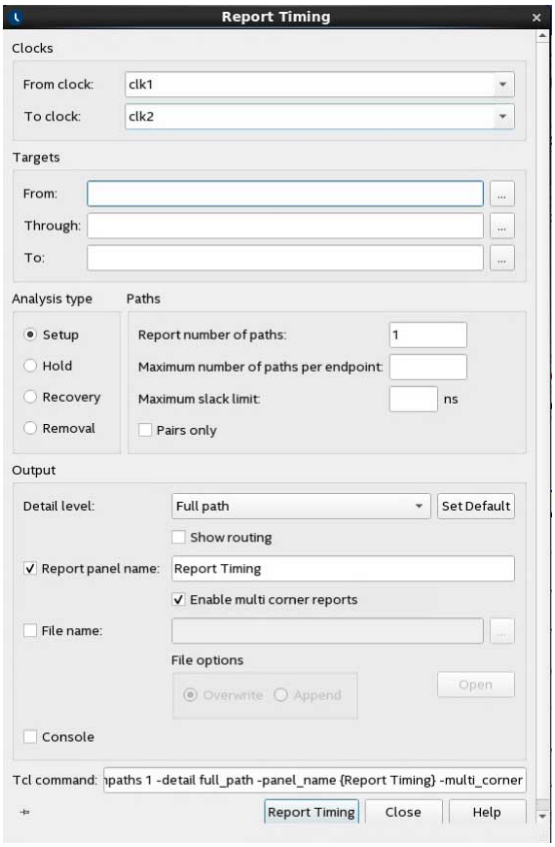


Table 6. Report Timing Options

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 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**.
Output	The **Detail** level, allows you to specify the path types the analysis includes in output. **Summary** level includes basic summary reports. **Path only** displays all the detailed information, except the **Data Path** tab displays the clock tree as one line item. 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. 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.
Enable multi corner reports	Enables or disables multi-corner timing analysis. This option is on by default.
Report panel name	Displays the name of the report panel. You can enable **File name** to write the information to a file. If you append `.htm` as a suffix, the Timing Analyzer produces the report as HTML.
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.
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.

2.2.5.5. Correlating Constraints to the Timing Report

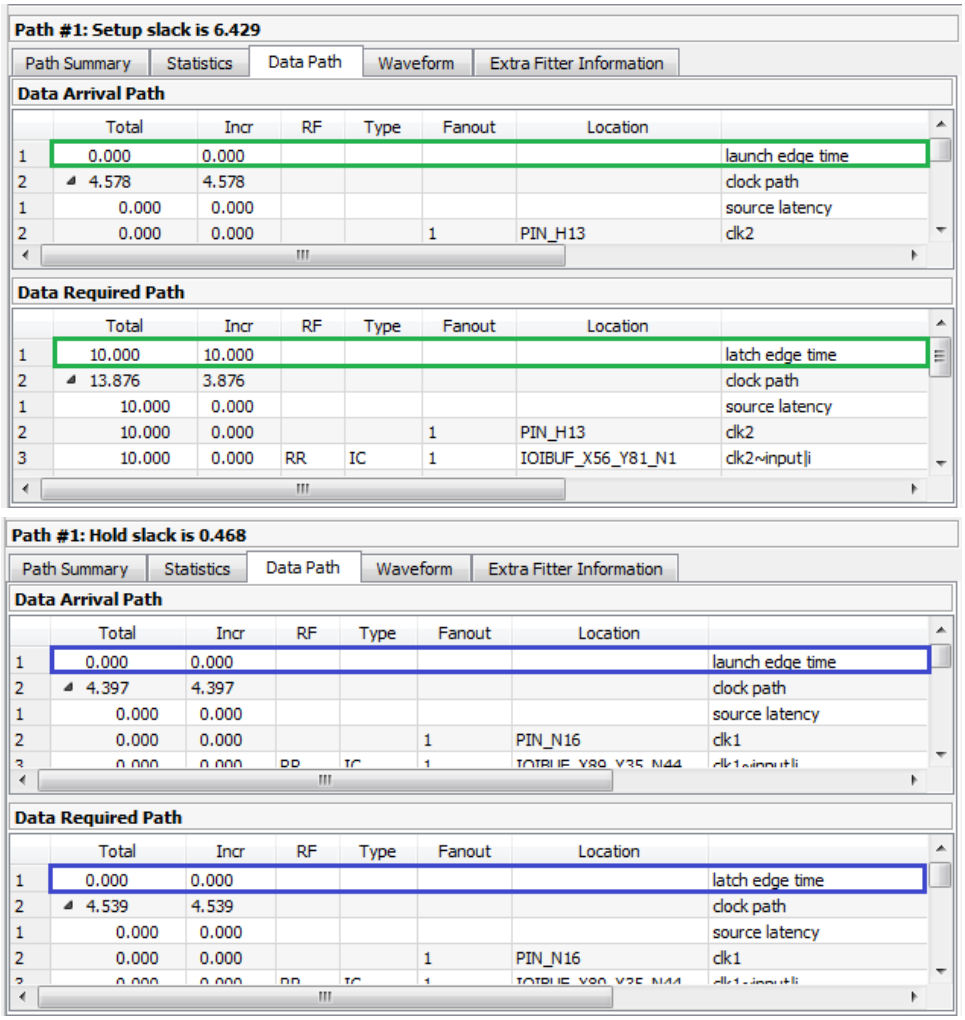
Understanding how timing constraints and violations appear in the timing analysis reports is critical to understanding the results. The following examples show how specific constraints impact the timing analysis reports. Most timing constraints only affect the clock launch and latch edges. Specifically, `create_clock` and `create_generated_clock` create clocks with default relationships. However, the `set_multicycle_path` exception modifies the default setup and hold relationship. 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.

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}]
```

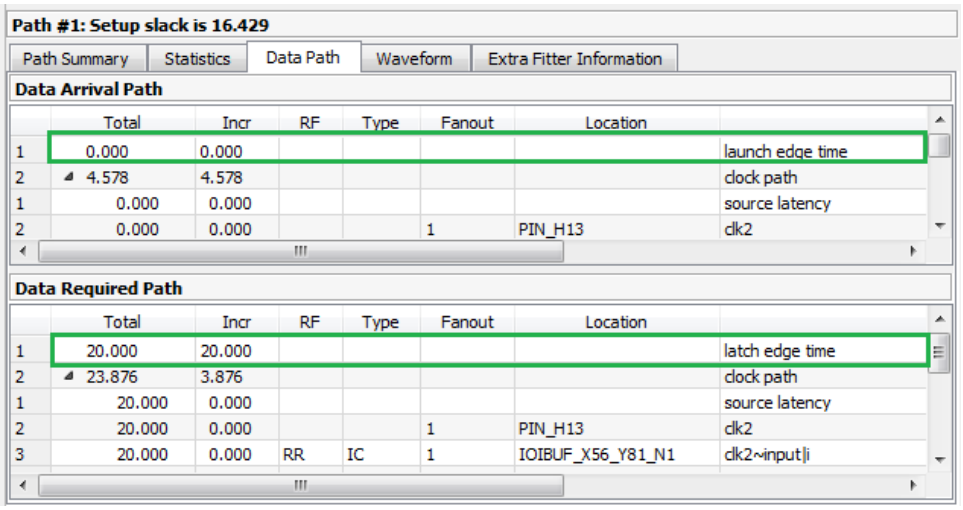
Figure 40. Setup Relationship 10ns, Hold Relationship 0ns



The `set_multicycle_path` constraint adds multicycles to relax the setup relationship, or open the window, making the setup relationship 20 ns while the hold relationship is still 0 ns:

```
set_multicycle_path -from clocktwo -to clocktwo -setup -end 2
set_multicycle_path -from clocktwo -to clocktwo -hold -end 1
```

Figure 41. Setup Relationship 20ns



Path #1: Setup slack is 16.429

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

Data Arrival Path

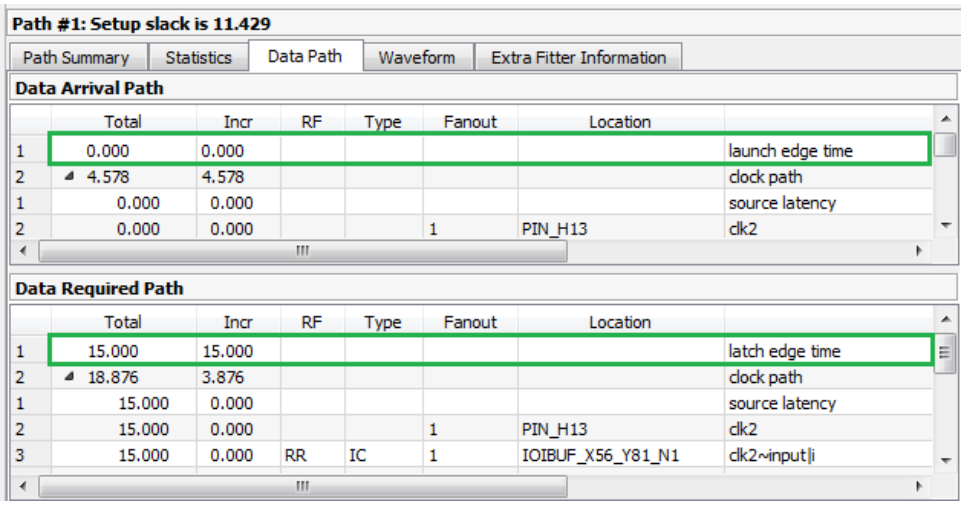
	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	◢ 4.578	4.578					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_H13	clk2

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	20.000	20.000					latch edge time
2	◢ 23.876	3.876					clock path
1	20.000	0.000					source latency
2	20.000	0.000			1	PIN_H13	clk2
3	20.000	0.000	RR	IC	1	IOIBUF_X56_Y81_N1	clk2~input|i

The `set_max_delay` and `set_min_delay` constraints explicitly override the setup relationship. 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 42. Using set_max_delay



Path #1: Setup slack is 11.429

Path Summary | Statistics | Data Path | Waveform | Extra Fitter Information

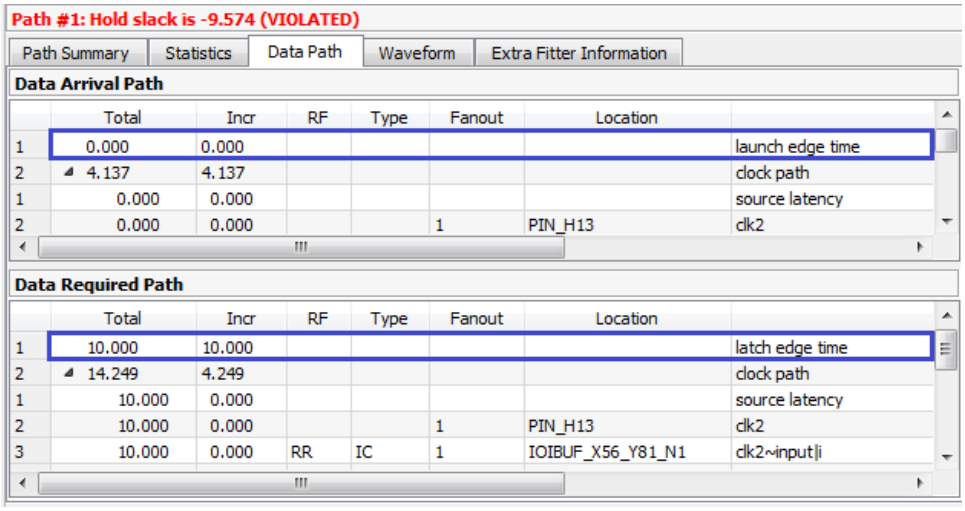
Data Arrival Path

	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					launch edge time
2	◢ 4.578	4.578					clock path
1	0.000	0.000					source latency
2	0.000	0.000			1	PIN_H13	clk2

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	15.000	15.000					latch edge time
2	◢ 18.876	3.876					clock path
1	15.000	0.000					source latency
2	15.000	0.000			1	PIN_H13	clk2
3	15.000	0.000	RR	IC	1	IOIBUF_X56_Y81_N1	clk2~input|i

Figure 43. Using set_min_delay



For I/O, you must add the `-max` and `-min` values. They are display as **iExt** or **oExt** in the **Type** column. An example is an output port with a `set_output_delay -max 1.0` and `set_output_delay -min -0.5`:

The clock relationships determine the launch and latch edge times, multicycles, and possibly `set_max_delay` or `set_min_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 on the left subtracts the `-max` value, making the setup relationship harder to meet, since the **Data Arrival Path** must be shorter than the **Data Required Path**. The Timing Analyzer also subtracts the `-min` value. This subtraction is why a negative number causes more restrictive hold timing. The **Data Arrival Path** must be longer than the **Data Required Path**.

Related Information

Relaxing Setup with Multicycle (set_multicyle_path) on page 59

2.2.5.6. Locating Timing Paths in Other Tools

You can locate from paths and elements in the Timing Analyzer to other tools in the Intel Quartus Prime software.

You can right-click most paths or node names in the Timing Analyzer GUI and click the **Locate** 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 Intel Quartus Prime tools.

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 2. 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
```

2.3. Using Timing Constraints

The following section describes correct application of SDC timing constraints that guide Fitter placement and allow accurate timing analysis. You can create an `.sdc` file with a set of initial recommended constraints, and then iteratively modify those constraints as the design progresses.

2.3.1. Recommended Initial SDC Constraints

Include the following basic SDC constraints in your initial `.sdc` file. The following example shows application of the recommended initial 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
```

Create Clock (create_clock) on page 32

Derive PLL Clocks (derive_pll_clocks) on page 33

Derive Clock Uncertainty (derive_clock_uncertainty) on page 34

Set Clock Groups (set_clock_groups) on page 34

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 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`. You can enter the command without using the `get_ports` collection command, as shown in the following example:

```
create_clock -name sys_clk -period 8.0 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 37

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.

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.

```
create_clock -period 10.0 -name fpga_sys_clk [get_ports fpga_sys_clk] \
    derive_pll_clocks
```

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 37
- Deriving PLL Clocks on page 43

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 a warning if you omit `derive_clock_uncertainty` from the `.sdc` file.

Related Information

Accounting for Clock Effect Characteristics on page 48

2.3.1.4. Set Clock Groups (set_clock_groups)

The **Set Clock Groups** (`set_clock_groups`) constraint allows you 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 groups of clock signals, and then define the relationship between the each group. You define the clock signals to include in each Group (-group), 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 are physically exclusive not active at the same time.
- The `-asynchronous`—defines completely unrelated clocks that have different ideal clock sources. 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.

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.

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 show which clocks are related in less than ten lines, while using `set_false_path` commands can use more than 50 lines.

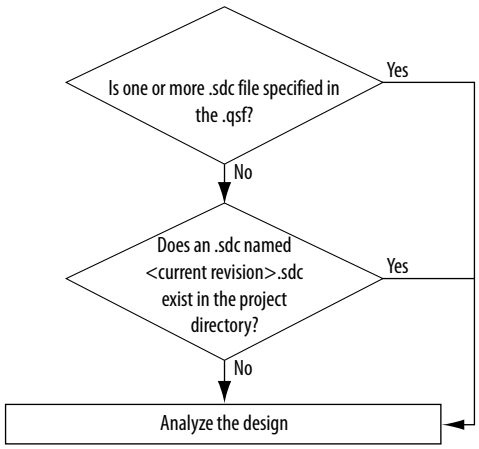
Related Information

- Creating Generated Clocks (create_generated_clock) on page 40
- Relaxing Setup with Multicycle (set_multicyle_path) on page 59
- Accounting for a Phase Shift (-phase) on page 60

2.3.2. SDC File Precedence

You must add any `.sdc` file that you create to the project to be read during fitting and timing analysis. The Fitter and the Timing Analyzer process `.sdc` files in the order they appear in the `.qsf`. If no `.sdc` appears in the `.qsf`, the Intel Quartus Prime software searches for an `.sdc` with the name `<current revision>.sdc` in the project directory.

Figure 44. .sdc File Order of Precedence



Click **Settings ➤ Timing Analyzer** to add, remove, or change the processing order of `.sdc` files in the project, as Step 3: Specify General Timing Analyzer Settings on page 22 describes.

If you use the Intel Quartus Prime Text Editor to create an `.sdc` file, the option to **Add file to the project** enables by default when you save the file. If you use any other editor to create an `.sdc` file, you must add the file to the project.

The `.sdc` file must contain only timing constraint commands. Tcl commands to manipulate the timing netlist or control the compilation must be in a separate Tcl script.

When you use IP from Intel, and some third-parties, the `.sdc` files become part of the project through an intermediate Intel Quartus Prime IP File (`.qip`). The `.qip` file references all source and constraints files for the IP. If `.sdc` files for IP blocks in your design are included through with a `.qip`, do not re-add them manually. An `.sdc` file can also be added from a Intel Quartus Prime IP File (`.qip`) included in the `.qsf`.

Note: If you type the `read_sdc` command at the command line without any arguments, the Timing Analyzer reads constraints embedded in HDL files, then follows the `.sdc` file precedence order.

2.3.3. Iterative Constraint Modification

You can iteratively modify `.sdc` constraints and reanalyze the timing results to ensure that you have the optimum constraints for your design.

Use the following steps to iteratively modify constraints:

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, as well as minimum pulse width checks, and a list of all the clock you define. These

summaries cover all paths you constrain in your design. Whenever modifying or correcting constraints, generate the **Diagnostic** reports to identify unconstrained parts of your design, or ignored constraints.

3. Analyze the results in the reports. When you are 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.

This method performs timing analysis using new constraints, without any change to logic placement. While the Fitter uses the original constraints for place and route, the Timing Analyzer applies the new constraints. If there is any failing timing against the new constraints, this indicates a need to run place-and-route again.

Related Information

Relaxing Setup with Multicycle (set_multicyle_path) on page 59

2.3.4. Creating Clocks and 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

2.3.4.1. 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 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 10 -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 10 -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 10 -name clk_100 [get_ports clk_sys]
create_clock -period 5 -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 exclusive.

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 10 –name clock_primary –waveform { 0 5 } \
    [get_ports clk]
create_clock –period 15 –name clock_secondary –waveform { 0 7.5 } \
    [get_ports clk] -add
```

Related Information

Accounting for Clock Effect Characteristics on page 48

2.3.4.1.1. 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
```

Warning: Do not use the `derive_clocks` command for final timing sign-off; instead, you create clocks for all clock sources with the `create_clock` and `create_generated_clock` commands. 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 thorough and realistic analysis of your design's timing requirements, 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.

2.3.4.2. 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 the clock at the external device connected to the FPGA.

To create virtual clocks, use the `create_clock` constraint with no value for the *<targets>* option.

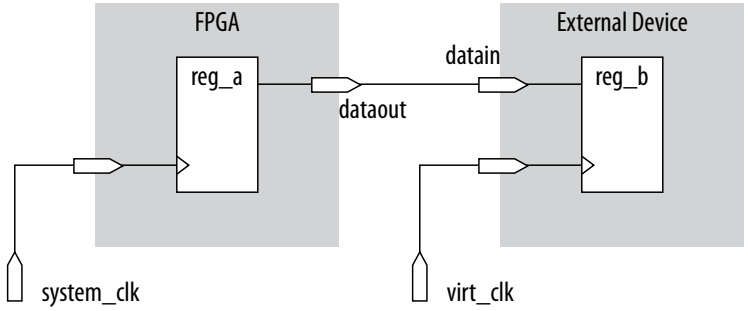
This following example defines a 100Mhz virtual clock because the command includes no *<targets>*.

```
create_clock -period 10 -name my_virt_clk
```

I/O Constraints with Virtual Clocks

For the output circuit shown in the following figure, 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. The following figure shows the base clock (`system_clk`), virtual clock (`virt_clk`), and output delay for the Virtual Clock Constraints example below.

Figure 45. Virtual Clock Board Topology



The following 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]
```

2.3.4.2.1. 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.

2.3.4.2.2. 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]
```

2.3.4.3. 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.

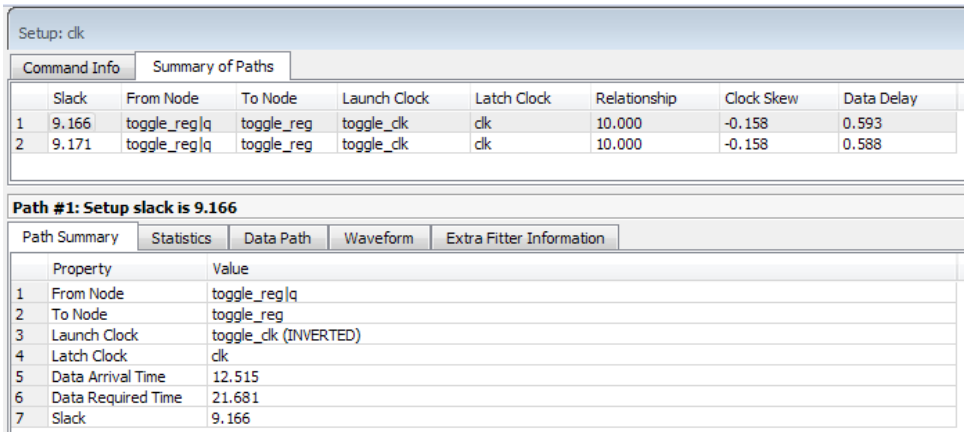
The `-source` option specifies the name of a node in the clock path that you use as reference for your generated clock. The source of the generated clock must be a node in your design netlist, and not the name of a clock you previously define. You can use any node name on the clock path between the input 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. The source of the generated clock decouples from the naming and hierarchy of the clock source. If you change the clock source, 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 signal name 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 46. 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.

Related Information

Deriving PLL Clocks on page 43

2.3.4.3.1. 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 10 -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 10 -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 47. Clock Divider

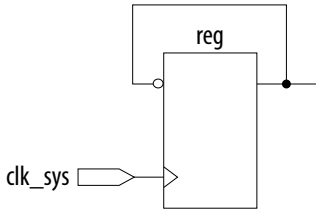
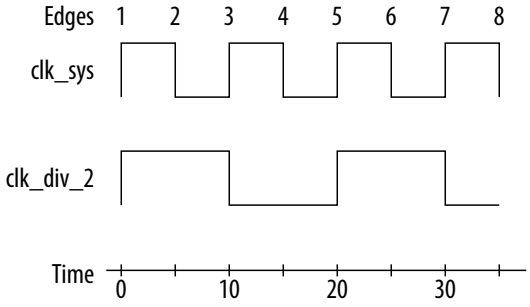


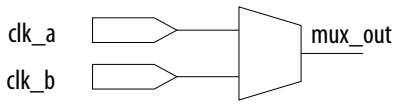
Figure 48. Clock Divider Waveform



2.3.4.3.2. Clock Multiplexor Example

The output of a clock multiplexor (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 49. 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 -exclusive -group clock_a_mux -group clock_b_mux
```

2.3.4.4. 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.

Create Base Clock for PLL input Clock Ports

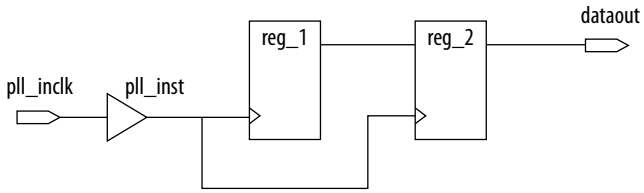
If your design contains transceivers, LVDS transmitters, or LVDS receivers, use the `derive_pll_clocks` to constrain this logic in your design and create timing exceptions for those blocks.

```
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 50. 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.

Related Information

Creating Clock Groups (set_clock_groups) on page 44

2.3.4.5. Creating Clock Groups (set_clock_groups)

The **Set Clock Groups** (`set_clock_groups`) constraint allows you 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.

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 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 7. set_clock_groups -group A

Dest\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 8. set_clock_groups -group {A B}

Dest\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 9. set_clock_groups -group A -group B

Dest\Source	A	B	C	D
A	Analyzed	Cut	Cut	Cut

continued...

B	Cut	Analyzed	Cut	Cut
C	Cut	Cut	Analyzed	Analyzed
D	Cut	Cut	Analyzed	Analyzed

Table 10. set_clock_groups -group {A C} -group {B D}

Dest\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 11. set_clock_groups -group {A C D}

Dest\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

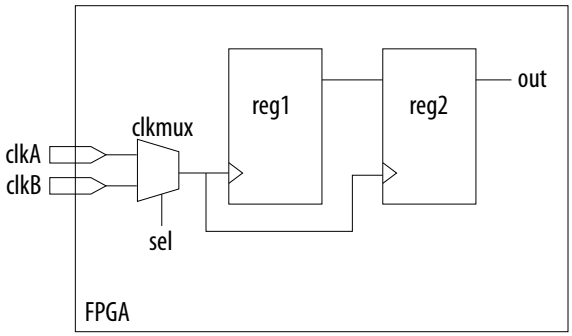
2.3.4.5.1. Exclusive Clock Groups (-exclusive)

You can use the `-exclusive` option to declare that two clocks are mutually exclusive.

If you define multiple clocks for the same node, you can use clock group assignments with the `-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 `-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 illustrate:

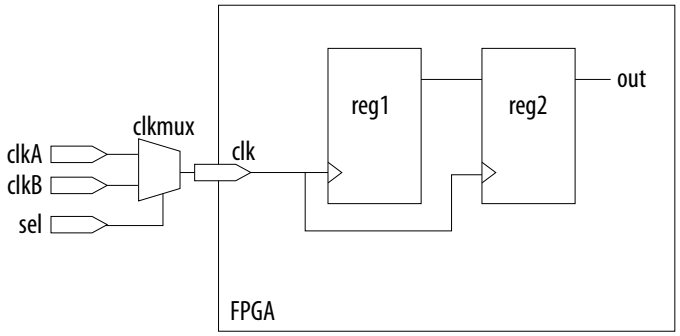
Figure 51. Synchronous Path with Clock Mux Internal to FPGA



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]
# Set the two clocks as exclusive clocks
set_clock_groups -exclusive -group {clk_100} -group {clk_125}
```

Figure 52. Synchronous Path with Clock Mux External to FPGA



Example SDC Constraints for External Clock Mux

```
# Create two clocks on the port clk
create_clock -name clkA -period 10 [get_ports clk]
create_clock -name clkB -period 8 [get_ports clk] -add
# Set the two clocks as exclusive clocks
set_clock_groups -exclusive -group {clkA} -group {clkB}
```

2.3.4.5.2. 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.

2.3.4.5.3. 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 SDC Constraints on page 32, 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. This characters escapes, 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. Usually most outputs of a PLL relate 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. Since the Timing Analyzer assumes that the clocks not appearing in the command relate to every clock, this conservatively groups the known clocks. If there are still failing paths in the design between unrelated clock domains, you can start add the new clock domains as necessary. In this case, a large number of the clocks are not in the `set_clock_groups` command, since they are either cut in the `.sdc` file for the IP core (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 core. Some common core constraints that this section does not describe in detail are:

- Adding multicycles between registers for analysis at a slower rate than the default analysis, increasing the time when data can be read. For example, a 10 ns clock period has a 10 ns setup relationship. If the data changes at a slower rate, or perhaps the registers switch at a slower rate due to a clock enable, then you can apply a multicycle that relaxes the setup relationship (opens the window so that valid data can pass). This is a multiple of the clock period, making the setup relationship 20 ns, 40 ns, and so on, while keeping the hold relationship at 0 ns. You generally apply these types of multicycles to paths.
- You can also use multicycle when you want to advance the cycle in which data is read, shifting the timing window. This generally occurs when your design performs a small phase-shift on a clock. For example, if your design has two 10 ns clocks exiting a PLL, but the second clock has a 0.5 ns phase-shift, the default setup relationship from the main clock to the phase-shift clock is 0.5 ns and the hold relationship is -9.5 ns. Meeting a 0.5 ns setup relationship is nearly impossible, and most likely you intend the data to transfer in the next window. By adding a multicycle from the main clock to the phase-shift clock, the setup relationship becomes 10.5 ns and the hold relationship becomes 0.5 ns. You generally apply this multicycle between clocks.
- Add a `create_generated_clock` to ripple clocks. When a register's output drives the `clk` port of another register, that is a ripple clock. Clocks do not propagate through registers, so you must apply the `create_generated_clock` constraint to all ripple clocks for correct analysis. Unconstrained ripple clocks appear in the **Report Unconstrained Paths** report, so you can easily recognize them. In general, avoid ripple clocks. Use a clock enable instead.
- Add a `create_generated_clock` to clock mux outputs. Without this clock, all clocks propagate through the mux and are related. The Timing Analyzer analyzes paths downstream from the mux where one clock input feeds the source register and the other clock input feeds the destination, and vice-versa. Although this behavior can be valid, this is typically not the behavior you want. By applying `create_generated_clock` constraints on the mux output, which relates them to the clocks coming into the mux, you can correctly group these clocks with other clocks.

2.3.4.6. 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.

2.3.4.6.1. 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.

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 setup analysis. The Timing Analyzer uses the early clock latency for the data arrival time, and the late clock latency for the clock arrival time, for hold analysis.

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 only can characterize source latency with the `set_clock_latency` command. You must use the `-source` option.

2.3.4.6.2. 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`.

2.3.5. Creating 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.

2.3.5.1. Input Constraints (set_input_delay)

Input constraints allow specify all the external delays feeding the device. 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 53. Input Delay Diagram

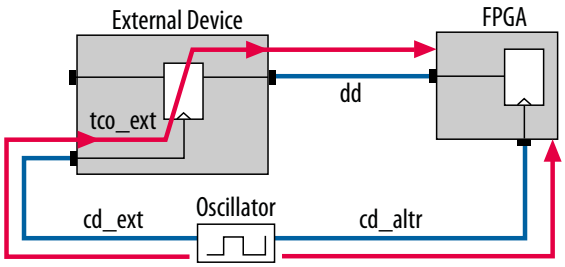


Figure 54. Input Delay Calculation

$$\text{input delay}_{MAX} = (\text{cd_ext}_{MAX} - \text{cd_altr}_{MIN}) + \text{tco_ext}_{MAX} + \text{dd}_{MAX}$$
$$\text{input delay}_{MIN} = (\text{cd_ext}_{MIN} - \text{cd_altr}_{MAX}) + \text{tco_ext}_{MIN} + \text{dd}_{MIN}$$

2.3.5.2. 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 55. Output Delay Diagram

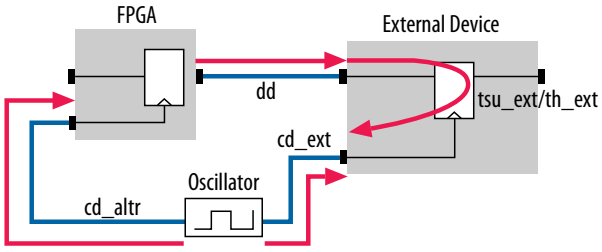


Figure 56. Output Delay Calculation

$$\text{output delay}_{MAX} = \text{dd}_{MAX} + \text{tsu_ext} + (\text{cd_altr}_{MAX} - \text{cd_ext}_{MIN})$$
$$\text{output delay}_{MIN} = (\text{dd}_{MIN} - \text{th_ext} + (\text{cd_altr}_{MIN} - \text{cd_ext}_{MAX}))$$

2.3.6. Creating Delay and Skew Constraints

You can specify skew and delays to model external device timing and board timing parameters.

2.3.6.1. 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
```

2.3.6.2. 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 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`, and `set_multicycle_path` do not affect `set_max_skew` timing constraints for this 18.1 version of the Timing Analyzer. However, `set_false_path` and `set_clock_groups` do impact `set_max_skew`. If your design targets an Intel Arria 10 device or Intel Cyclone 10 device, `set_clock_groups` does not affect `set_max_skew` constraints.

Table 12. set_max_skew Options

Arguments	Description
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-exclude <Tcl list>`	A Tcl list of parameters to exclude during skew analysis. This list includes one or more of the following: `utsu`, `uth`, `utco`, `from_clock`, `to_clock`, `clock_uncertainty`, `ccpp`, `input_delay`, `output_delay`, `odv`. *Note:* Intel Arria 10 devices do no support this argument.
`-fall_from_clock <names>`	Valid source clocks (Tcl matches string patterns).
names>	Valid destination clocks (Tcl matches string patterns).
`-from <-fall_to_clock <names>`[1]	Valid sources (Tcl matches string patterns).
`-fall_to_clock -from_clock <names>`	Valid source clocks (Tcl matches string patterns).

continued...

(1) Legal values for the -from and -to options are collections of clocks, registers, ports, pins, cells or partitions in a design.

Arguments	Description
`-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.
`-include <Tcl list>`	Tcl list of parameters to include during skew analysis. This list can include one or more of the following: `utsu`, `uth`, `utco`, `from_clock`, `to_clock`, `clock_uncertainty`, `ccpp`, `input_delay`, `output_delay`, `odv`. *Note:* Intel Arria 10 devices do not support this argument .
`-rise_from_clock <names>`	Valid source clocks (Tcl matches string patterns).
`-rise_to_clock <names>`	Valid destination clocks (Tcl matches string patterns).
`-skew_value_multiplier <multiplier>`	Value by which the clock period multiplies to compute skew requirement.
`-to <names>`(1)	Valid destinations (Tcl matches string patterns)
`-to_clock <names>`	Valid destination clocks (Tcl matches string patterns).
`<skew>`	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).

Use the `-include` and `-exclude` options to include or exclude one or more of the following: register micro parameters (`utsu`, `uth`, `utco`), clock arrival times (`from_clock`, `to_clock`), clock uncertainty (`clock_uncertainty`), common clock path pessimism removal (`ccpp`), input and output delays (`input_delay`, `output_delay`) and on-die variation (`odv`).

Max skew analysis can include data arrival times, clock arrival times, register micro parameters, clock uncertainty, on-die variation, and ccpp removal. Among these, only ccpp removal disables during the Fitter by default. When you use `-include` , the default analysis includes those in the inclusion list. Similarly, if you use `-exclude`, the default analysis excludes those in the exclusion list. When both the `-include` and `-exclude` options specify the same parameter, that parameter is excluded.

Note: If your design targets an Intel Arria 10 device, `-exclude` and `-include` are not supported.

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.

2.3.6.3. 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.

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 13. set_net_delay Options

Arguments	Description
`-h | -help`	Short help.
`-long_help`	Long help with examples and possible return values.
`-from <names>`	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>`[2]	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_skew_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 also set `-value_multiplier`, and 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 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 (that is, if the endpoints of the nets are not registers or constraint ports), then the Timing Analyzer ignores the net delay constraint.

2.3.6.4. Create Timing Netlist

You can configure or load the timing netlist that the Timing Analyzer uses to calculate path delay data.

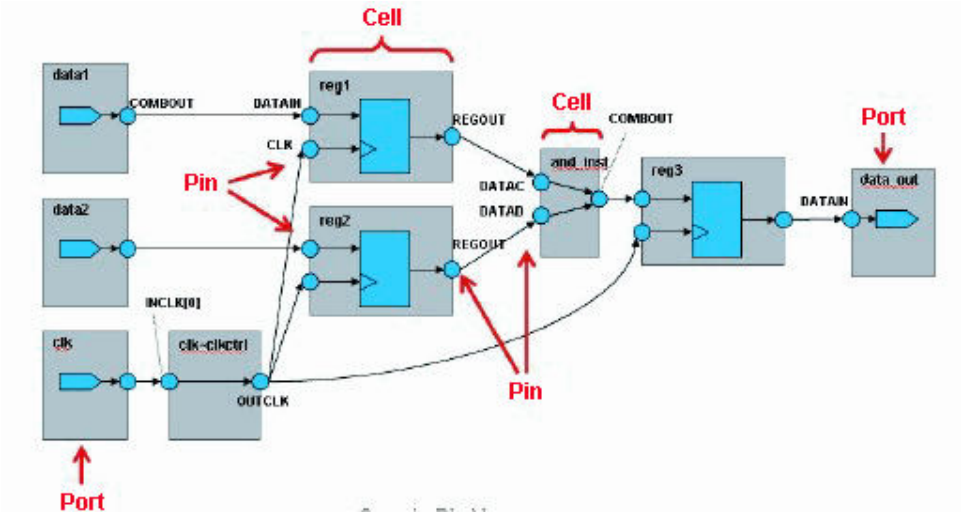
You must generate the timing netlist before running timing analysis. You can use the **Create Timing Netlist** dialog box or the **Create Timing Netlist** command in the **Tasks** pane. **Create Timing Netlist** also generates Advanced I/O Timing reports if you turn on **Enable Advanced I/O Timing** in the **Timing Analyzer** page of the **Settings** dialog box.

(2) If no -to option, or if `-to` is a wildcard ("*") character, all the output pins and registers on timing netlist become valid destination points.

Note: The Compiler creates the timing netlist during compilation. The timing netlist does not reflect any configuration changes that occur after the device enters user mode, such as dynamic transceiver reconfiguration. This applies to all device families except transceivers on Intel Arria 10 devices with the Multiple Reconfiguration Profiles feature.

The following diagram shows how the Timing Analyzer interprets and classifies timing netlist data for a sample design.

Figure 57. How Timing Analyzer Interprets the Timing Netlist



2.3.7. Creating Timing Exceptions

Timing exceptions modify (or provide exception to) the default timing analysis behavior to account for your specific design conditions. Specify timing exceptions after specifying clocks and input and output delay constraints, because timing exceptions modify the default analysis.

2.3.7.1. Timing Constraint Precedence

If the same clock or node names occur in multiple timing exceptions, the Timing Analyzer observes the following order of timing constraint precedence:

1. **Set False Path** (`set_false_path`) is the first priority
2. **Set Minimum Delay** (`set_min_delay`) and **Set Maximum Delay** (`set_max_delay`) are the second priority.
3. **Set Multicycle Path** (`set_multicycle_path`) is the third 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 doesn't) and specifies a `-to` (while #2 doesn't). 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.

`set_net_delay` or `set_max_skew` exceptions analyze independently of minimum or maximum delays, or multicycle path constraints.

- The `set_net_delay` exception applies regardless the existence of a `set_false_path` exception, or `set_clock_group` exception, on the same nodes.
- When targeting the Intel Arria 10 device or Intel Cyclone 10 device and using the 18.1 version of the Timing Analyzer, the `set_max_skew` exception applies regardless of any `set_clock_group` exception on the same nodes, but a `set_false_path` exception overrides a `set_max_skew` exception.

2.3.7.2. 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 beginning with A, to all registers beginning 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. For example, you can specify a false path for a static configuration register that is written 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.

Related Information

Creating Clock Groups (set_clock_groups) on page 44

2.3.7.3. 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.

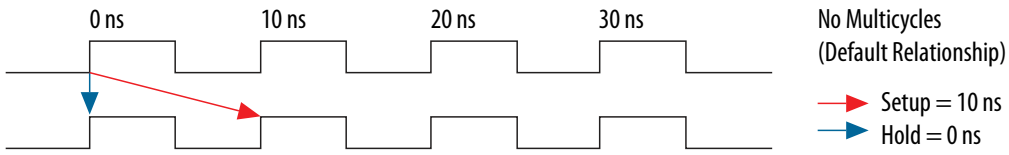
You can create a minimum or maximum delay exception for an output port that does not have an output delay constraint. You cannot report timing for the paths that relate to the output port; however, the Timing Analyzer reports any slack for the path in the setup summary and hold summary reports. Because 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.

2.3.7.4. 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.

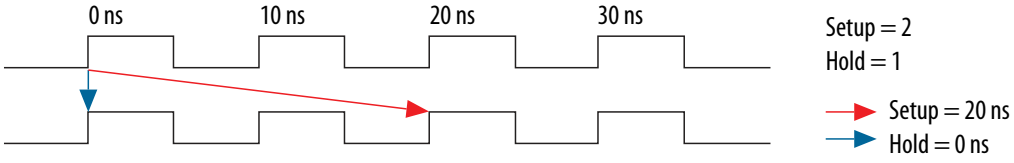
Figure 58. Default Setup and Hold Relationship (No Multicycle)



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. The Timing Analyzer does not report negative setup or hold relationships. When the Timing Analyzer detects either a negative setup or a negative hold relationship, the Timing Analyzer moves both the launch and latch edges until the setup and hold relationship becomes positive.

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 59. 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 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 14. 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.

2.3.7.4.1. 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.

2.3.7.4.2. 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.

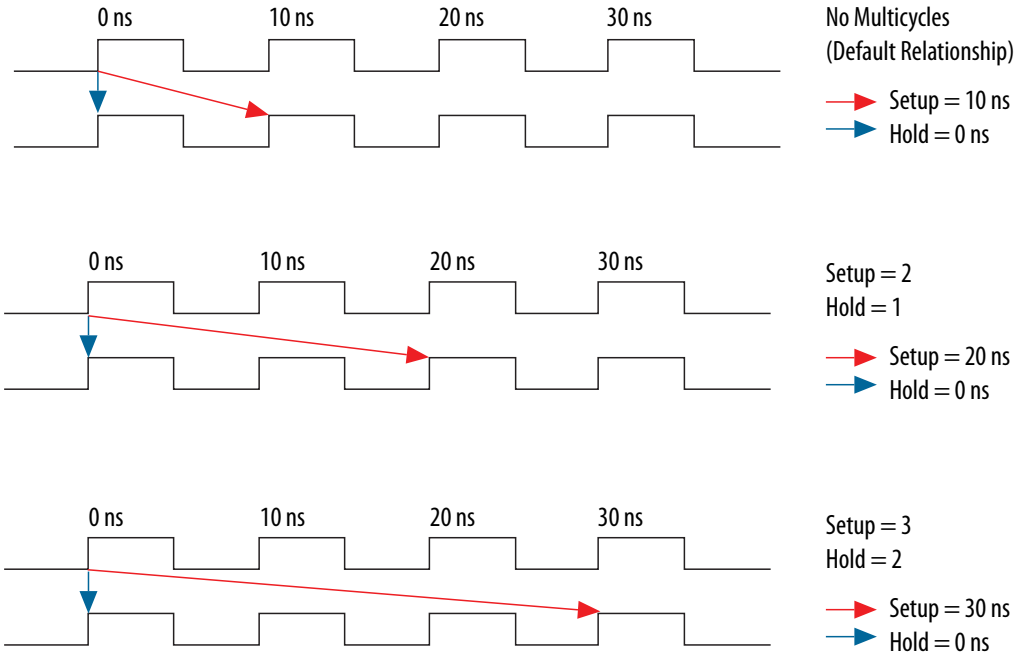
In the following example, the source clock has a period of 10 ns, but the clock enables a group of registers, so the registers only enable every other cycle. Since the registers are fed by a 10 ns clock, the Timing Analyzer reports a setup of 10 ns and a hold of 0 ns. However, since the data is transferring every other cycle, the Timing Analyzer must analyze the relationships as if the clock is operating at 20 ns. That results in 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 60. 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
```

2.3.7.4.3. 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.

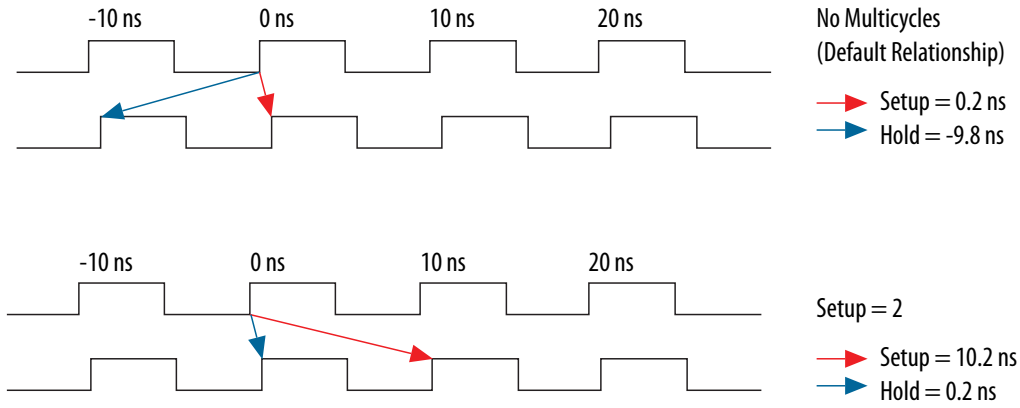
For example, the following code phase-shifts one output of a PLL forward by a small amount, in this case 0.2 ns.

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 61. 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 69

2.3.7.5. 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.

2.3.7.5.1. 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 62. Default Timing Diagram

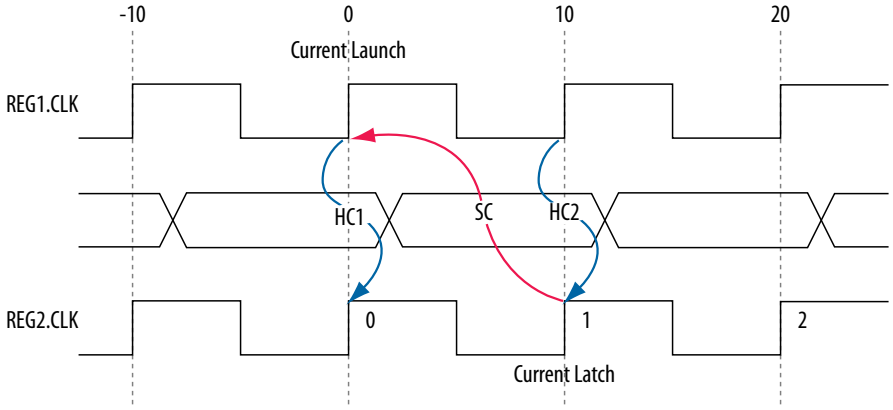


Figure 63. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 10\ \text{ns} - 0\ \text{ns} \\ &= 10\ \text{ns} \end{aligned}$$

The most restrictive setup relationship with the default single-cycle analysis, that is, a setup relationship with an end multicycle setup assignment of one, is 10 ns.

The setup report for the default setup in the Timing Analyzer with the launch and latch edges highlighted.

Figure 64. Setup Report

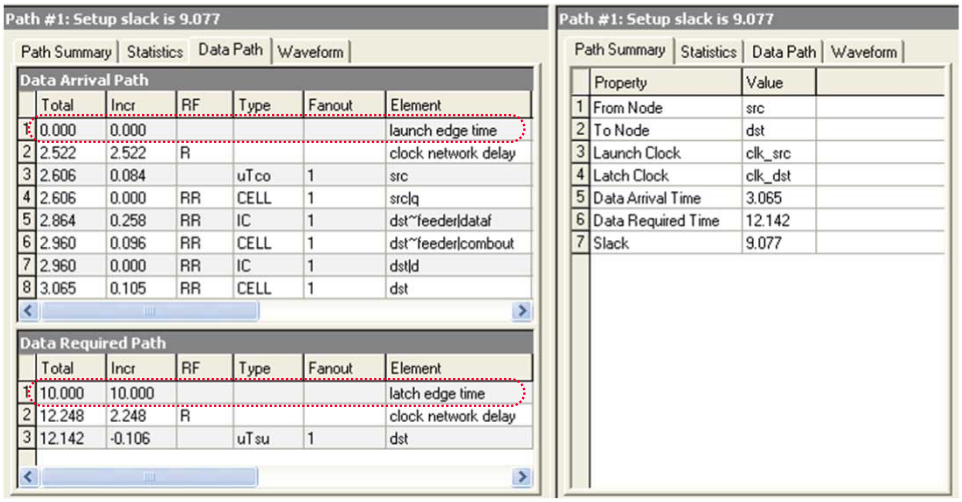


Figure 65. Hold Check Calculation

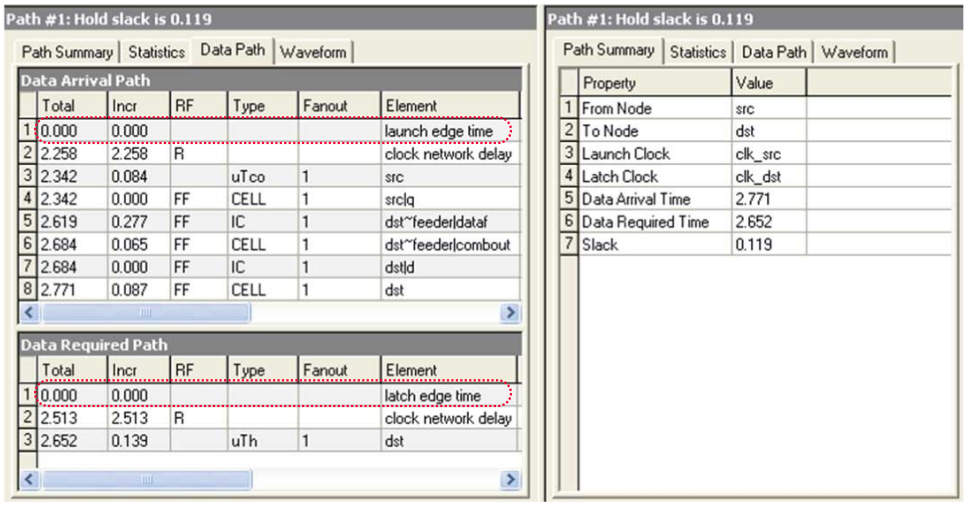
$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 0\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 10\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

The most restrictive hold relationship with the default single-cycle analysis, that a hold relationship with an end multicycle hold assignment of zero, is 0 ns.

The hold report for the default setup in the Timing Analyzer with the launch and latch edges highlighted.

Figure 66. Hold Report



2.3.7.5.2. 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 67. Setup Timing Diagram

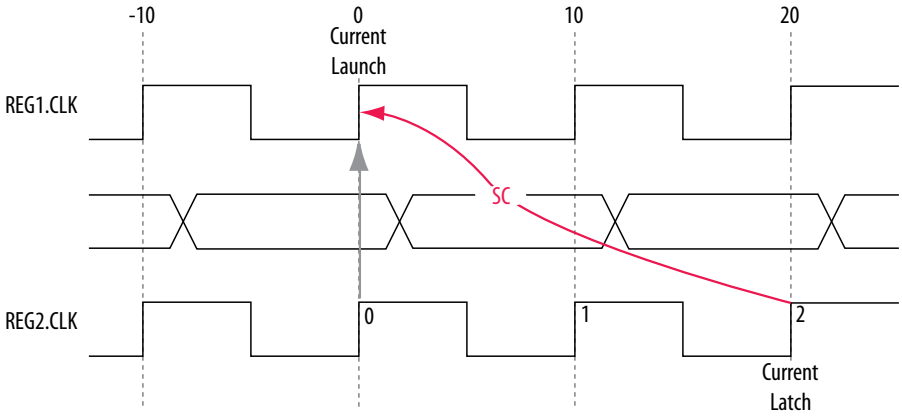


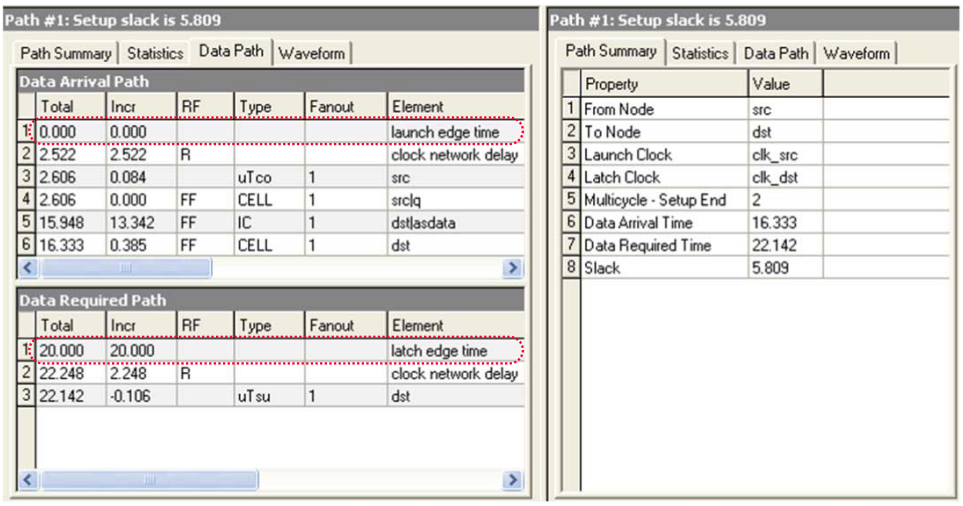
Figure 68. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 20\ \text{ns} - 0\ \text{ns} \\ &= 20\ \text{ns} \end{aligned}$$

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 69. Setup Report



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 70. Hold Timing Diagram

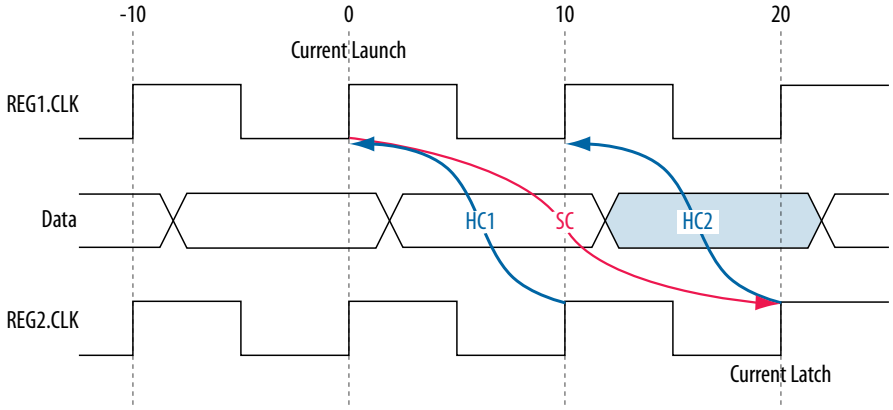


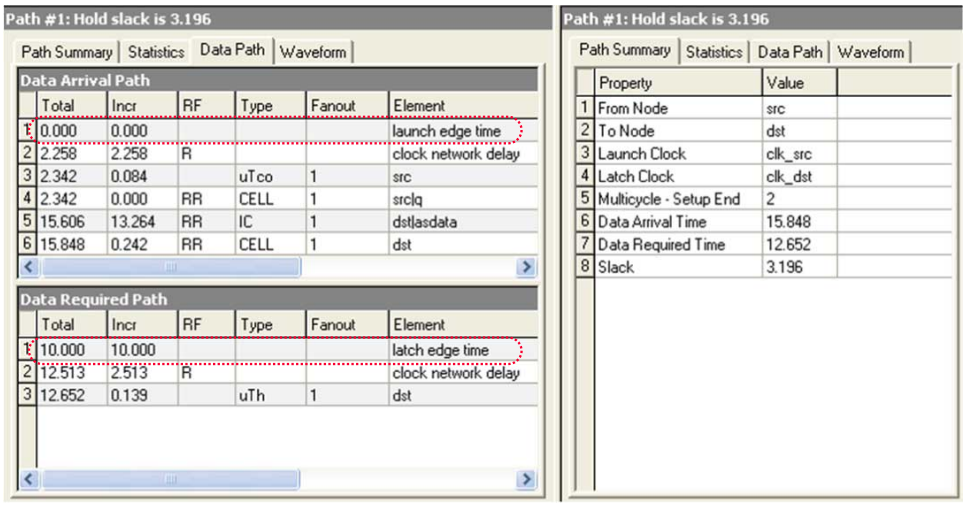
Figure 71. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 10\ \text{ns} \\ &= -10\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 20\ \text{ns} \\ &= -10\ \text{ns} \end{aligned}$$

This is the most restrictive hold relationship with an end multicycle setup assignment value of two and an end multicycle hold assignment value of zero is 10 ns.

Figure 72. Hold Report



2.3.7.5.3. 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 two clock periods by moving the latch edge to the left two clock periods. The hold relationship relaxes by a full period by moving the latch edge to the previous latch edge.

The following shows the setup timing diagram for the analysis that the Timing Analyzer performs:

Figure 73. Setup Timing Diagram

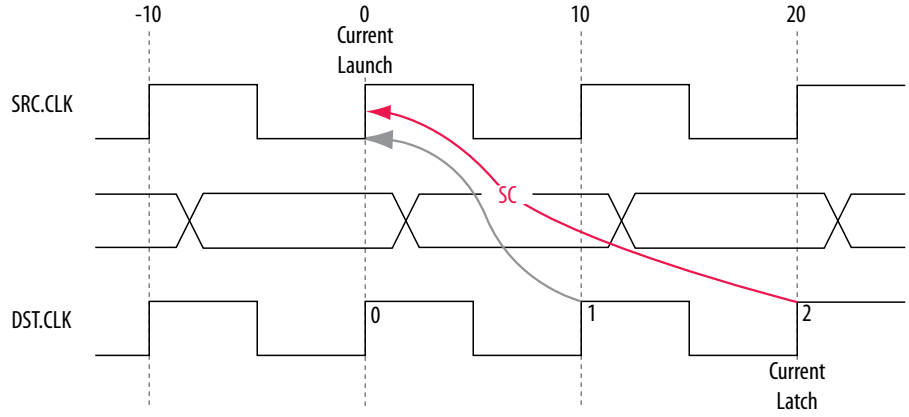


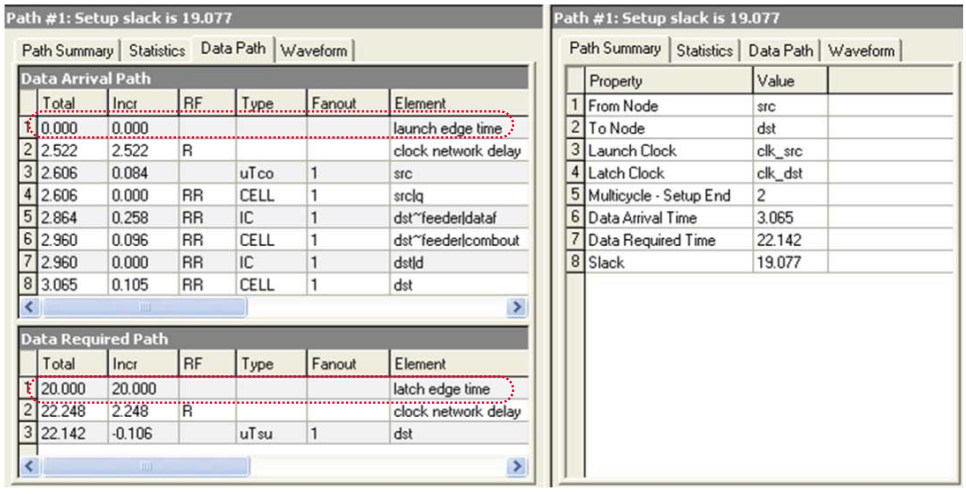
Figure 74. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 20\ \text{ns} - 0\ \text{ns} \\ &= 20\ \text{ns} \end{aligned}$$

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 75. Setup Report



The following shows the timing diagram for the hold checks for this example. The hold checks are relative to the setup check.

Figure 76. Hold Timing Diagram

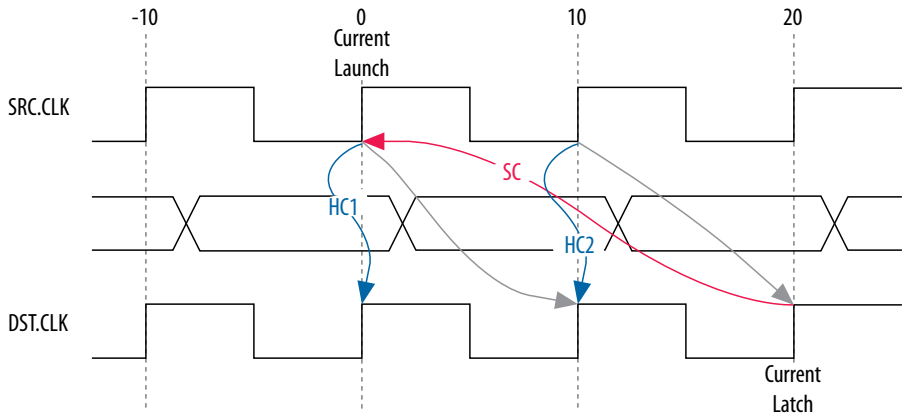


Figure 77. Hold Check Calculation

$$\text{hold check 1} = \text{current launch edge} - \text{previous latch edge}$$
$$= 0\text{ ns} - 0\text{ ns}$$
$$= 0\text{ ns}$$

$$\text{hold check 2} = \text{next launch edge} - \text{current latch edge}$$
$$= 10\text{ ns} - 10\text{ ns}$$
$$= 0\text{ ns}$$

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 78. Hold Report

Path #1: Hold slack is 0.119

Path Summary | Statistics | Data Path | Waveform

Data Arrival Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				launch edge time
2	2.258	2.258	R			clock network delay
3	2.342	0.084		uTco	1	src
4	2.342	0.000	FF	CELL	1	srclq
5	2.619	0.277	FF	IC	1	dst~feederldataf
6	2.684	0.065	FF	CELL	1	dst~feederlcombout
7	2.684	0.000	FF	IC	1	dstld
8	2.771	0.087	FF	CELL	1	dst

Data Required Path

	Total	Incr	RF	Type	Fanout	Element
1	0.000	0.000				latch edge time
2	2.513	2.513	R			clock network delay
3	2.652	0.139		uTh	1	dst

Path #1: Hold slack is 0.119

Path Summary | Statistics | Data Path | Waveform

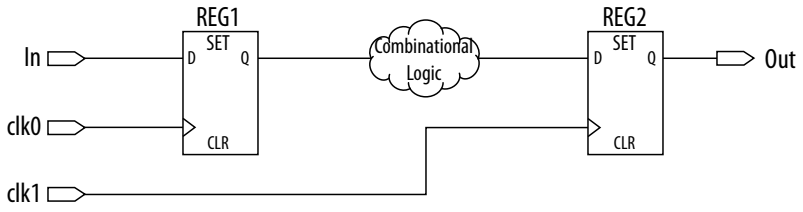
	Property	Value
1	From Node	src
2	To Node	dst
3	Launch Clock	clk_src
4	Latch Clock	clk_dst
5	Multicycle - Setup End	2
6	Multicycle - Hold End	1
7	Data Arrival Time	2.771
8	Data Required Time	2.652
9	Slack	0.119

2.3.7.5.4. 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 79. Same Frequency Clocks with Destination Clock Offset Diagram



The following timing diagram shows the default setup check analysis that the Timing Analyzer performs.

Figure 80. Setup Timing Diagram

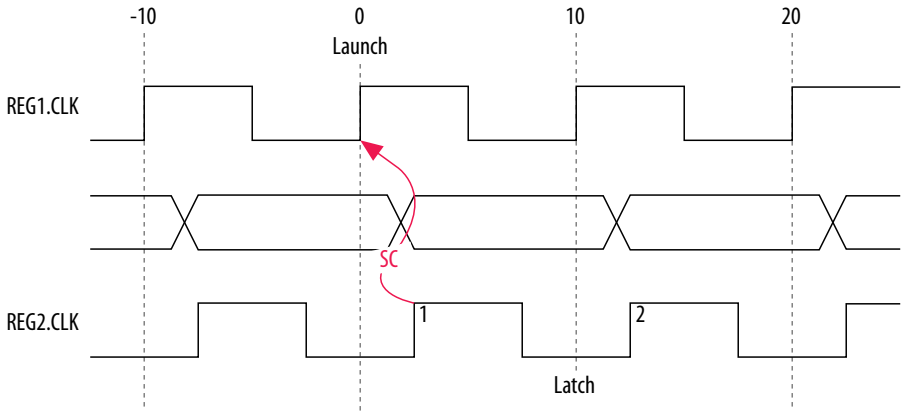


Figure 81. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 2\ \text{ns} - 0\ \text{ns} \\ &= 2\ \text{ns} \end{aligned}$$

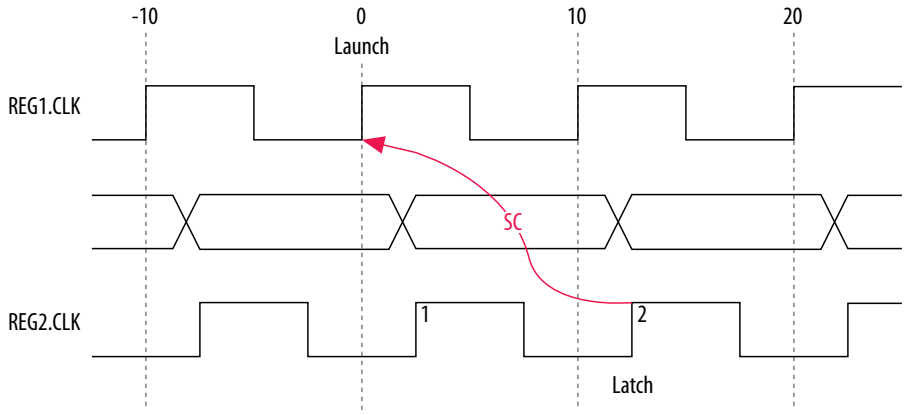
The setup relationship shown is too pessimistic and is not the setup relationship required for typical designs. To adjust the default analysis, you assign an end multicycle setup exception of two. The following shows a multicycle exception that adjusts the default analysis:

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 82. 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 83. Default Hold Check

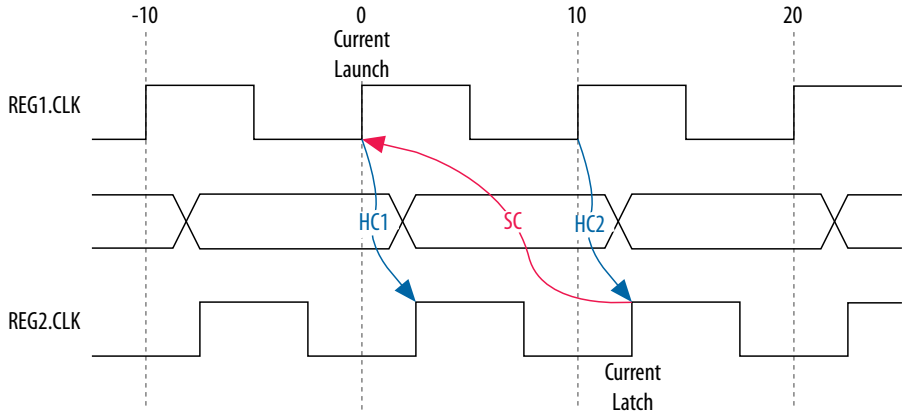


Figure 84. Hold Check Calculation

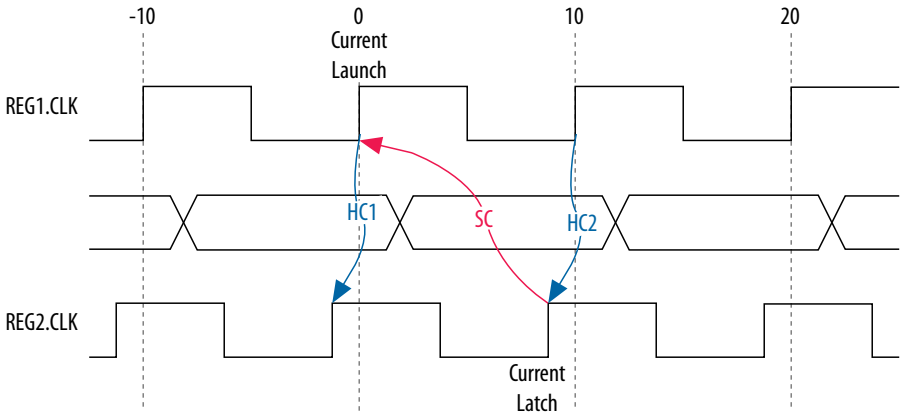
$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 2\ \text{ns} \\ &= -2\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 12\ \text{ns} \\ &= -2\ \text{ns} \end{aligned}$$

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 85. Negative Phase Shift

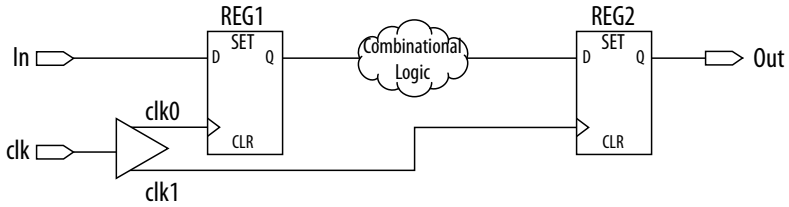


2.3.7.5.5. Destination Clock Frequency is a Multiple of the Source Clock Frequency

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 86. Destination Clock is Multiple of Source Clock



The following timing diagram shows the default setup check analysis that the Timing Analyzer performs:

Figure 87. Setup Timing Diagram

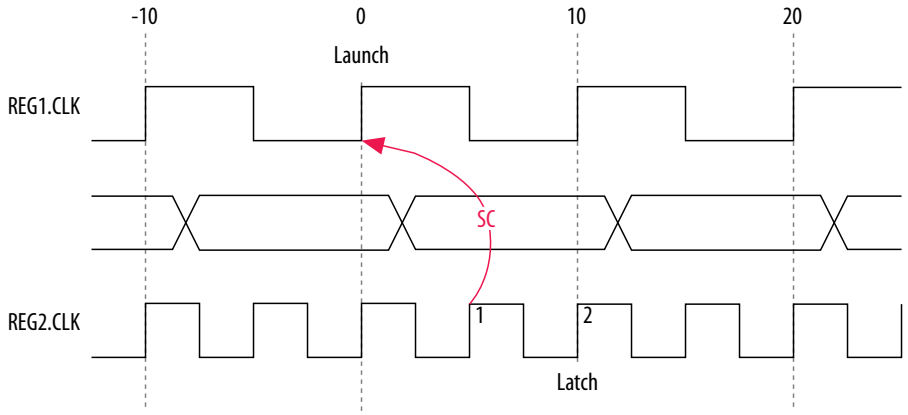


Figure 88. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 5\text{ ns} - 0\text{ ns} \\ &= 5\text{ ns} \end{aligned}$$

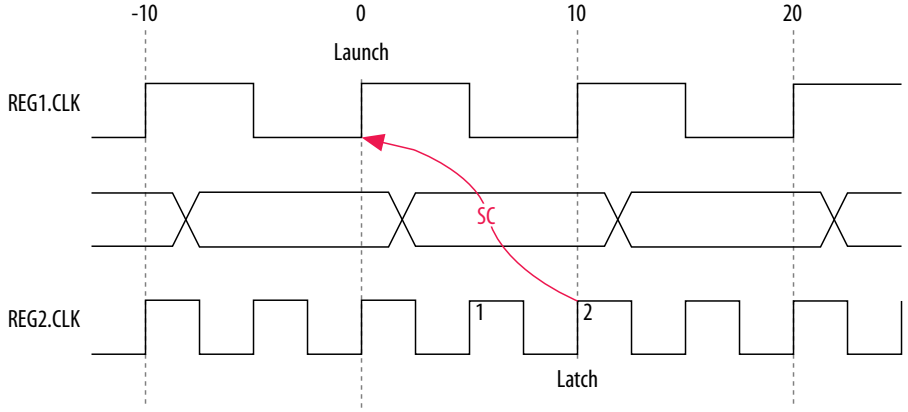
The setup relationship demonstrates that the data requires capture at edge two; therefore, you can relax the setup requirement. To correct the default analysis, you shift the latch edge by one clock period with an end multicycle setup exception of two. The following multicycle exception assignment adjusts 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 89. 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 90. Default Hold Check

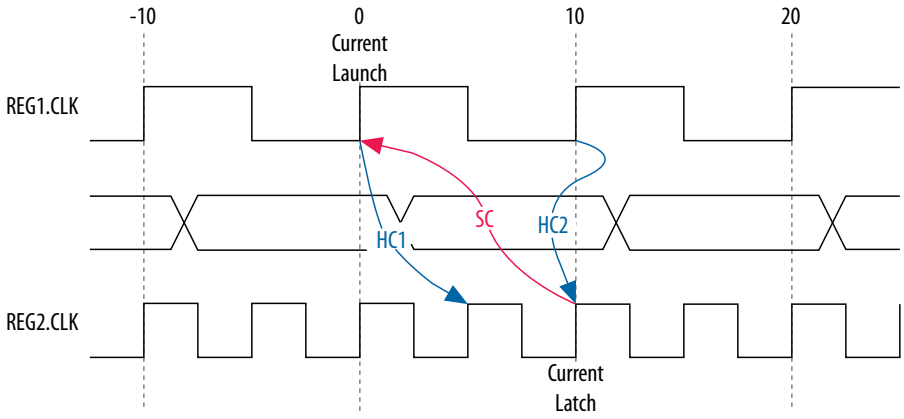


Figure 91. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 5\ \text{ns} \\ &= -5\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\ \text{ns} - 10\ \text{ns} \\ &= 0\ \text{ns} \end{aligned}$$

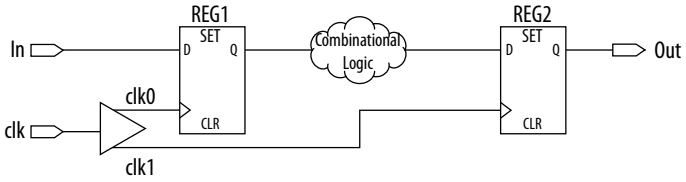
In this example, hold check one is too restrictive. The data is launched by the edge at 0 ns and must check against the data captured by the previous latch edge at 0 ns, which does not occur in hold check one. To correct the default analysis, you must use an end multicycle hold exception of one.

2.3.7.5.6. 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 92. Destination Clock is Multiple of Source Clock with Offset



The timing diagram for the default setup check analysis the Timing Analyzer performs.

Figure 93. Setup Timing Diagram

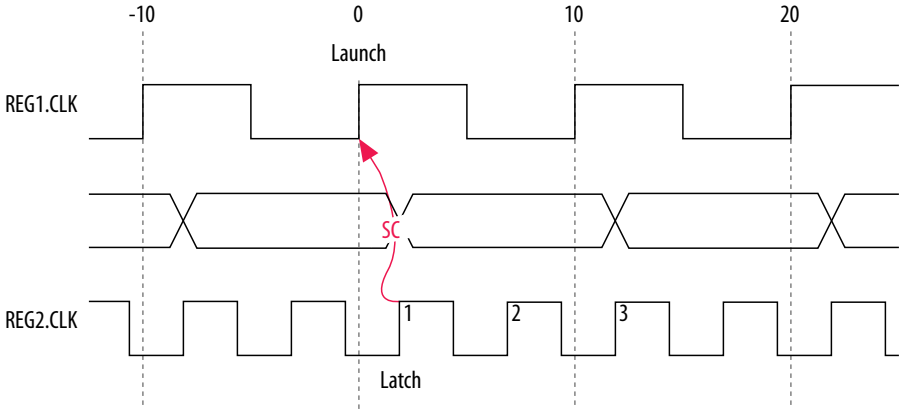


Figure 94. Hold Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 2\ \text{ns} - 0\ \text{ns} \\ &= 2\ \text{ns} \end{aligned}$$

The setup relationship in this example demonstrates that the data does not require capture at edge one, but rather requires capture at edge two; therefore, you can relax the setup requirement. To adjust the default analysis, you shift the latch edge by one clock period, and specify an end multicycle setup exception of three.

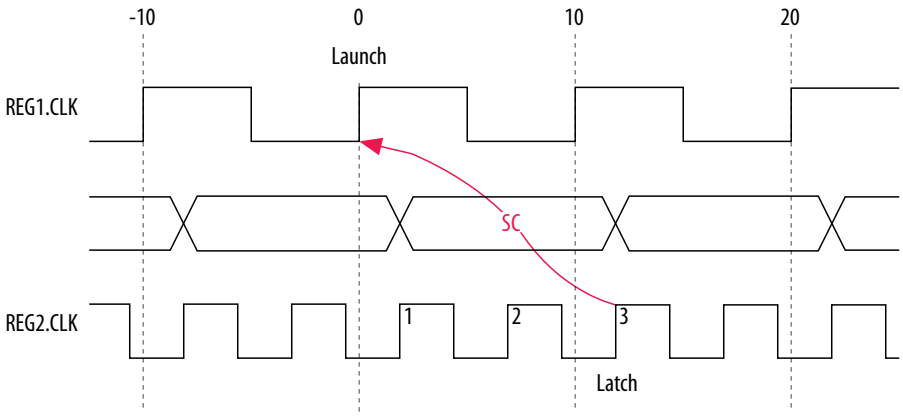
The multicycle exception adjusts the default analysis in this example:

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 95. 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 96. Default Hold Check

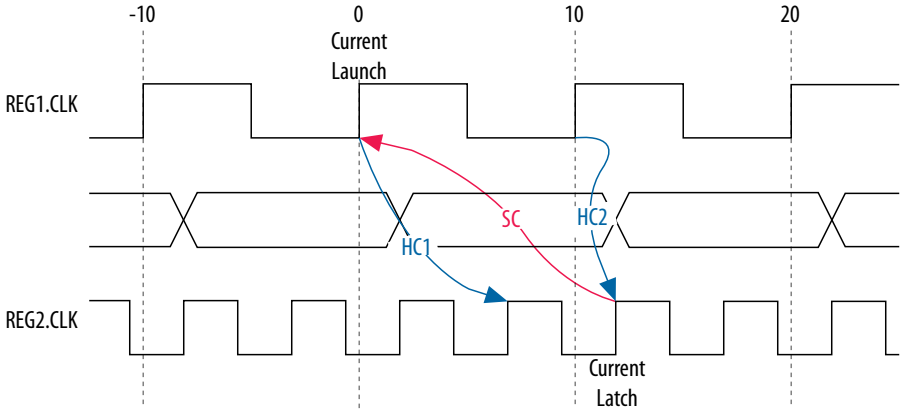


Figure 97. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 5\text{ ns} \\ &= -5\text{ ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 10\text{ ns} - 10\text{ ns} \\ &= 0\text{ ns} \end{aligned}$$

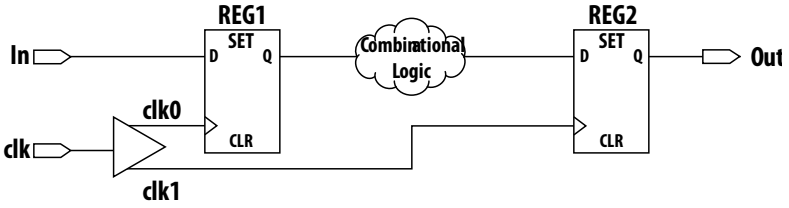
In this example, the hold check one is too restrictive. The data is launched by the edge at 0 ns, and must check against the data that the previous latch edge at 2 ns captures. This event does not occur in hold check one. To adjust the default analysis, you assign end multicycle hold exception of one.

2.3.7.5.7. 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 98. Source Clock Frequency is Multiple of Destination Clock Frequency:



The following timing diagram shows the default setup check analysis the Timing Analyzer performs:

Figure 99. Default Setup Check Analysis

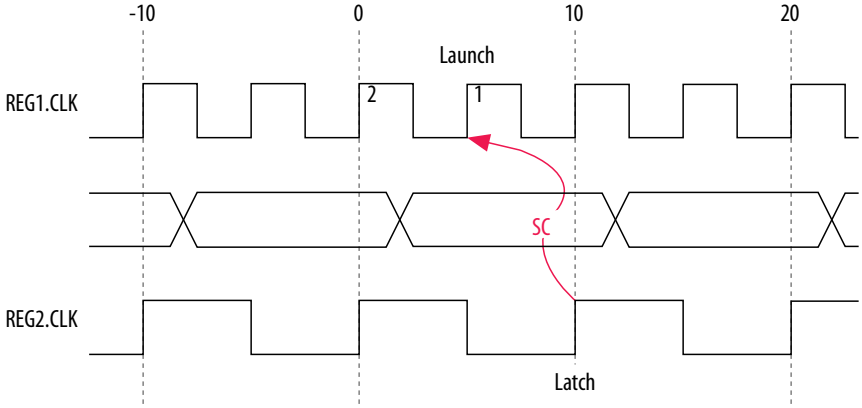


Figure 100. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 10\ \text{ns} - 5\ \text{ns} \\ &= 5\ \text{ns} \end{aligned}$$

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.

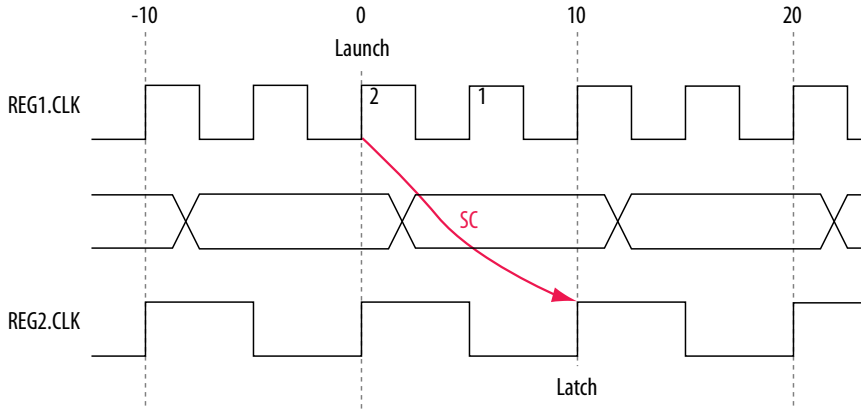
The following multicycle exception adjusts the default analysis in this example:

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 101. 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 102. Default Hold Check

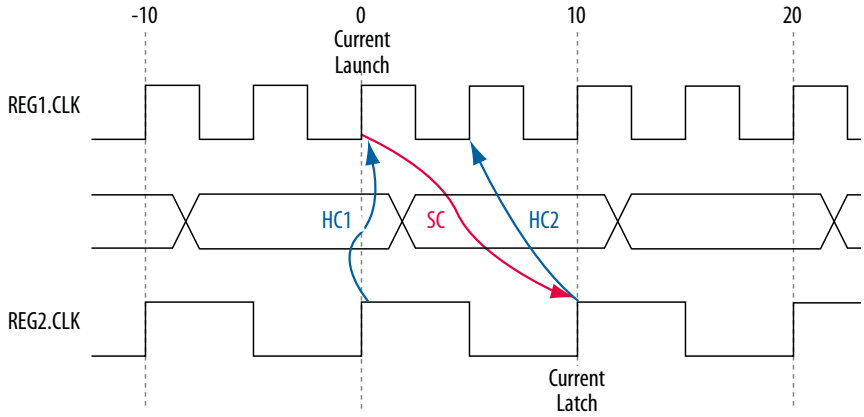


Figure 103. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\text{ ns} - 0\text{ ns} \\ &= 0\text{ ns} \end{aligned}$$

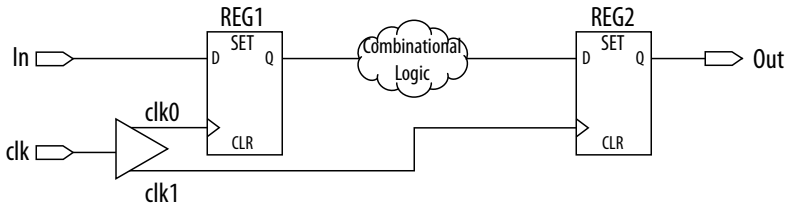
$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 5\text{ ns} - 10\text{ ns} \\ &= -5\text{ ns} \end{aligned}$$

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.

2.3.7.5.8. 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 104. 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 105. Setup Timing Diagram

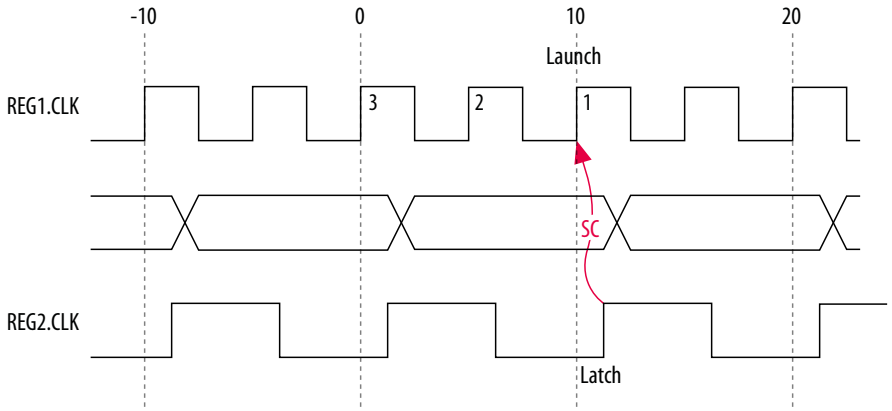


Figure 106. Setup Check Calculation

$$\begin{aligned} \text{setup check} &= \text{current latch edge} - \text{closest previous launch edge} \\ &= 12\text{ ns} - 10\text{ ns} \\ &= 2\text{ ns} \end{aligned}$$

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.

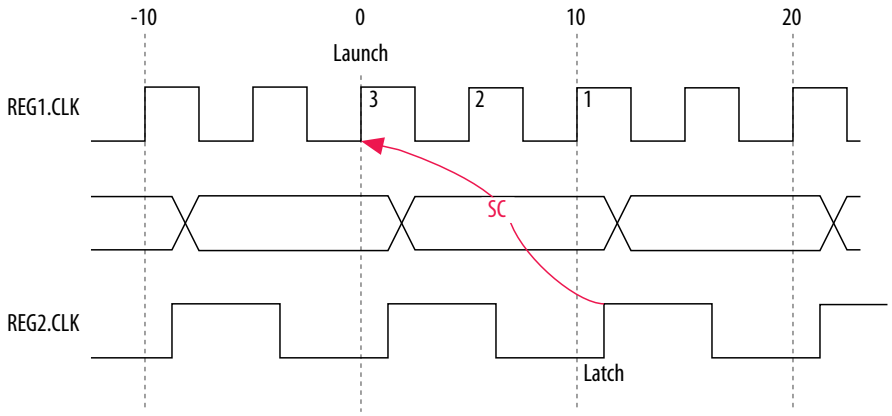
The following multicycle exception adjusts the default analysis in this example:

Multicycle Constraint

```
set_multicycle_path -from [get_clocks clk_src] -to [get_clocks clk_dst] \
     -setup -start 3
```

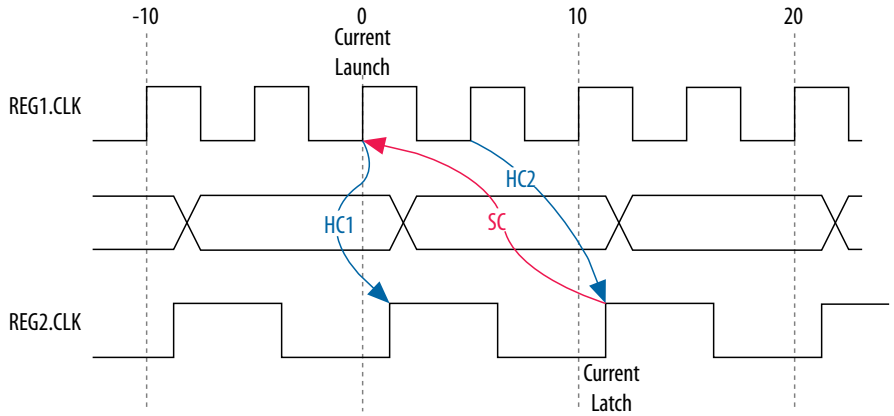
The following timing diagram shows the preferred setup relationship for this example:

Figure 107. 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 three:

Figure 108. Default Hold Check Analysis



The Timing Analyzer performs the following calculation to determine the hold check:

Figure 109. Hold Check Calculation

$$\begin{aligned} \text{hold check 1} &= \text{current launch edge} - \text{previous latch edge} \\ &= 0\ \text{ns} - 2\ \text{ns} \\ &= -2\ \text{ns} \end{aligned}$$

$$\begin{aligned} \text{hold check 2} &= \text{next launch edge} - \text{current latch edge} \\ &= 5\ \text{ns} - 12\ \text{ns} \\ &= -7\ \text{ns} \end{aligned}$$

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.

2.3.7.6. Delay Annotation

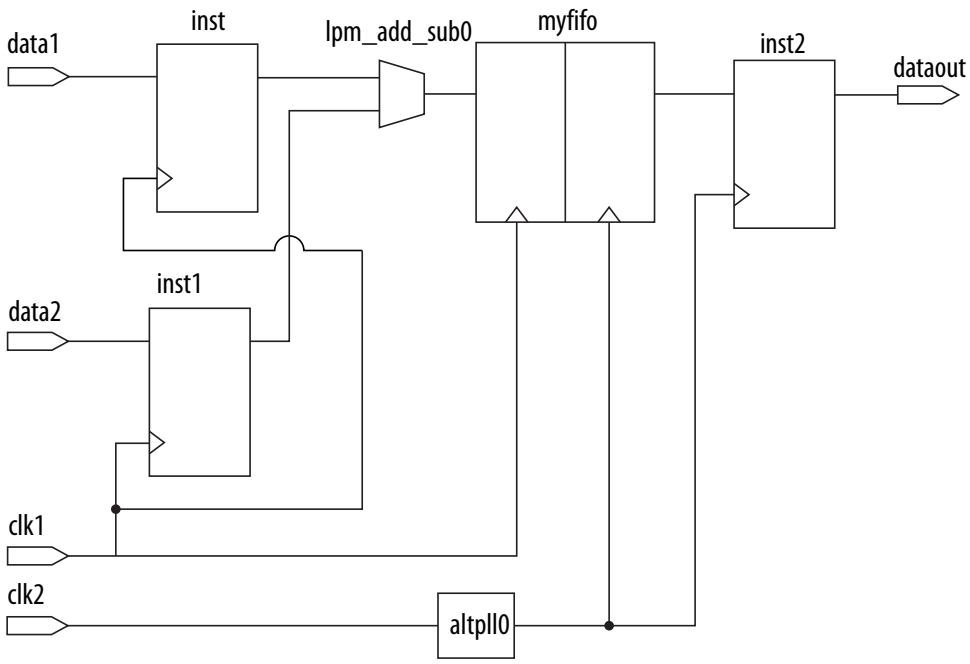
To modify the default delay values used during timing analysis, use the `set_annotated_delay` and `set_timing_derate` commands. You must update the timing netlist to apply these commands.

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.

2.3.8. Example Circuit and SDC File

The following circuit and corresponding `.sdc` file demonstrates constraining a design that includes two clocks, a phase-locked loop (PLL), and other common synchronous design elements.

Figure 110. Dual-Clock Design Constraint Example



The .sdc file contains basic constraints for the example circuit.

Example 7. Basic .sdc Constraints Example

```
# Create clock constraints
create_clock -name clockone -period 10.000 [get_ports {clk1}]
create_clock -name clocktwo -period 10.000 [get_ports {clk2}]
# Create virtual clocks for input and output delay constraints
create clock -name clockone_ext -period 10.000
create clock -name clocktwo_ext -period 10.000
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 `.sdc` file contains the following basic 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 its related clocks, and the third group containing 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.

Related Information

Asynchronous Clock Groups (-asynchronous) on page 46

2.4. Timing Analyzer Tcl Commands

You can optionally use Tcl commands from the Intel 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`

2.4.1. The quartus_sta Executable

The `quartus_sta` executable allows you to run timing analysis without running the full Intel Quartus Prime software GUI. 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
```

- To run the Timing Analyzer in interactive command-shell mode, type the following at the command prompt:

```
quartus_sta -s <options><project_name>
```

- To run multi-corner timing analysis from a system command prompt, type the following command:

```
quartus_sta <options><project_name>
```

Table 15. 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*>.

--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.
--tq2pt	Generates temporary files to convert the Timing Analyzer .sdc file to a PrimeTime .sdc file.
-f <*argument file*>	Specifies a file containing additional command-line arguments.
-c <*revision name*> | --rev=<*revision_name*>	Specifies which revision and its associated Intel 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.
--64bit	Enables 64-bit version of the executable.

2.4.2. 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 16. 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.
`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.

You can also examine collections and experiment with collections using wildcards in the Timing Analyzer by clicking **Name Finder** from the **View** menu.

2.4.2.1. 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`.

2.4.2.2. 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 Intel 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.

2.4.2.3. 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.

2.4.2.4. 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 17. Sample Search Strings and Search Results

Search String	Search Result
`get_pins *|dataa`	`lvl|dataa`
`get_pins *|datac`	*<empty>*[3]
`get_pins *|*|datac`	`lvl|cnod|datac`
`get_pins lvl*|*`	`lvl|dataa, lvl|datab`
`get_pins -hierarchical *|*|datac`	*<empty>*[3]
`get_pins -hierarchical lvl|*`	`lvl|dataa, lvl|datab`
`get_pins -hierarchical *|datac`	`lvl|cnod|datac`
`get_pins -hierarchical lvl|*|datac`	*<empty>*[3]
`get_pins -compatibility_mode *|datac`	`lvl|cnod|datac` [4]
`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 and 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`.

(3) 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.

(4) When you use `-compatibility_mode`, the Timing Analyzer does not treat pipe characters (|) as special characters when you use the characters with wildcards.

2.4.3. Identifying the Intel Quartus Prime Software Executable from the SDC File

To identify which Intel Quartus Prime software executable is currently running you can use the `$::TimingAnalyzerInfo(nameofexecutable)` variable from within an SDC file. This technique is most commonly used when you want to use an overconstraint to cause the Fitter to work harder on a particular path or set of paths in the design.

Identifying the Intel Quartus Prime Executable

```
#Identify which executable is running:
set current_exe $::TimingAnalyzerInfo(nameofexecutable)
if { [string equal $current_exe "quartus_fit"] } {
    #Apply .sdc assignments for Fitter executable here
} else {
    #Apply .sdc assignments for non-Fitter executables here
}
if { ! [string equal "quartus_sta" $::TimingAnalyzerInfo(nameofexecutable)] } {
    #Apply .sdc assignments for non-Timing Analyzer executables here
} else {
    #Apply .sdc assignments for Timing Analyzer executable here
}
```

Examples of different executable names are `quartus_map` for Analysis & Synthesis, `quartus_fit` for Fitter, and `quartus_sta` for the Timing Analyzer.

2.5. 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 Intel 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.

2.6. Using the Intel Quartus Prime Timing Analyzer Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2024.02.21	18.1.0	• Clarified constraint precedence in *Maximum Skew* and *Timing Constraint Precedence* topics.
2018.09.24	18.1.0	• Revised "Basic Timing Analysis Flow" section to add sequential step organization, update steps, and add supporting screenshots. • 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."
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*. • Updated information on using Intel Arria 10 devices with enhanced timing algorithms.

continued...

Document Version	Intel 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	14.1.0	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	14.0.0	• Minor updates. • Updated format.
November 2013	13.1.0	• Removed HardCopy device information.
June 2012	12.0.0	• Reorganized chapter. • Added "Creating a Constraint File from Intel Quartus Prime Templates with the Intel Quartus Prime Text Editor" section on creating an SDC constraints file with the **Insert Template** dialog box. • Added "Identifying the Intel Quartus Prime Software Executable from the SDC File" section. • Revised multicycle exceptions section.
November 2011	11.1.0	• Consolidated content from the Best Practices for the Intel Quartus Prime Timing Analyzer chapter. • Changed to new document template.
May 2011	11.0.0	• Updated to improve flow. Minor editorial updates.
December 2010	10.1.0	• Changed to new document template. • Revised and reorganized entire chapter. • Linked to Intel Quartus Prime Help.

continued...

Document Version	Intel Quartus Prime Version	Changes
July 2010	10.0.0	Updated to link to content on SDC commands and the Timing Analyzer GUI in Intel Quartus Prime Help.
November 2009	9.1.0	Updated for the Intel 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	8.1.0	Updated for the Intel 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

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Power Analysis and Optimization

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Online Version
Send Feedback

683506
UG-20184
2018.09.24

Contents

1. Power Analysis 4
1.1. Comparison of the EPE and the Intel Quartus Prime Power Analyzer 5
1.2. Power Estimations and Design Requirements 6
1.3. Power Analyzer Walkthrough 6
1.4. Inputs for the Power Analyzer 7
1.4.1. Operating Settings and Conditions 7
1.4.2. Sources for Signal Activity Data 8
1.5. Power Analysis in Modular Design Flows 11
1.5.1. Complete Design Simulation 13
1.5.2. Modular Design Simulation 13
1.5.3. Multiple Simulations on the Same Entity 13
1.5.4. Overlapping Simulations 14
1.5.5. Partial Simulations 14
1.5.6. Node Name Matching Considerations 15
1.5.7. Glitch Filtering 16
1.5.8. Node and Entity Assignments 17
1.5.9. Default Toggle Rate Assignment 18
1.5.10. Vectorless Estimation 18
1.6. Power Analyzer Compilation Report 18
1.7. Scripting Support 21
1.7.1. Running the Power Analyzer from the Command–Line 21
1.8. Power Analysis Revision History 22

2. Power Optimization 25
2.1. Factors Affecting Power Consumption 25
2.1.1. Design Activity and Power Analysis 25
2.1.2. Device Selection 26
2.1.3. Environmental Conditions 26
2.1.4. Device Resource Usage 27
2.1.5. Signal Activity 27
2.2. Power Dissipation 28
2.3. Design Space Explorer II for Power-Driven Optimization 29
2.4. Power-Driven Compilation 30
2.4.1. Power-Driven Synthesis 30
2.4.2. Power-Driven Fitter 33
2.4.3. Area-Driven Synthesis 34
2.4.4. Gate-Level Register Retiming 34
2.4.5. Intel Quartus Prime Compiler Settings 35
2.4.6. Assignment Editor Options 36
2.5. Design Guidelines 37
2.5.1. Clock Power Management 37
2.5.2. Pipelining and Retiming 43
2.5.3. Architectural Optimization 44
2.5.4. I/O Power Guidelines 45
2.5.5. Memory Optimization (M20K/MLAB) 46
2.5.6. DDR Memory Controller Settings 47
2.5.7. DSP Implementation 48

2.5.8. Reducing High-Speed Tile (HST) Usage....48
2.5.9. Unused Transceiver Channels....49
2.5.10. Periphery Power reduction XCVR Settings....50
2.6. Power Optimization Advisor....50
2.6.1. Set Realistic Timing Constraints....51
2.6.2. Appropriate Device Family....51
2.6.3. Dynamic Power....52
2.6.4. Static Power....52
2.6.5. Appropriate I/O Standards....53
2.6.6. Use RAM Blocks....53
2.6.7. Shut Down RAM Blocks....53
2.6.8. Clock Enables on Logic....53
2.6.9. Pipeline Logic to Reduce Glitching....53
2.7. Power Optimization Revision History....54
A. Intel Quartus Prime Standard Edition User Guides....56

1. Power Analysis

The Intel® Quartus® Prime Design Suite provides tools to estimate power consumption in a FPGA design at different stages of the design process. The Intel Quartus Prime Power Analyzer allows you to estimate power consumption for a post-fit design. To estimate the power consumption before you compile the design, use the Early Power Estimator (EPE) spreadsheet.

Note: Do not use the results of the Power Analyzer as design specifications.

As designs grow larger and process technology continues to shrink, power becomes an increasingly important design consideration. When designing a PCB, you must estimate the power consumption of a device accurately to develop an appropriate power budget, and to design the power supplies, voltage regulators, heat sink, and cooling system.

The Intel Quartus Prime Power Analyzer helps you establish guidelines for the power budget of your design. Make sure to verify the actual power during device operation, because this information is sensitive to the actual device design and the environmental operating conditions.

This chapter describes the Intel Quartus Prime Power Analyzer tool. For details about the EPE spreadsheets, refer to the Early Power Estimator page in the Altera website.

Note: The Intel Quartus Prime Power Analyzer does not support the Intel Arria® 10 HPS IP. You can obtain a power estimation for this Intel FPGA IP with the EPE spreadsheet.

Related Information

- Early Power Estimator Page
- Power Analyzer Support Resources

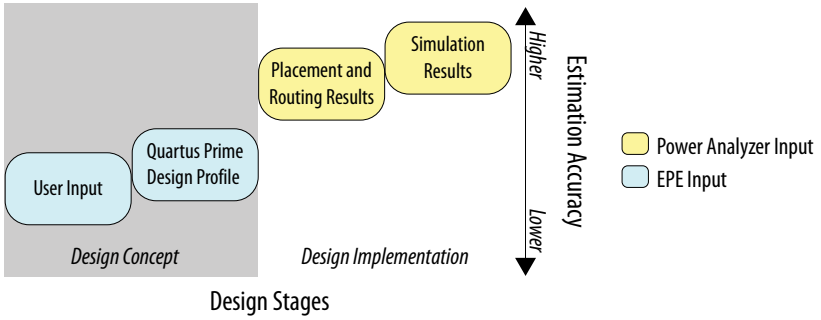
Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

1.1. Comparison of the EPE and the Intel Quartus Prime Power Analyzer

The following figure shows the design stages in which you use power analysis tools, and compares the accuracy of the estimations for different input types:

Figure 1. Estimation Accuracy for Different Inputs and Power Analysis Tools



The following table lists the differences between the EPE and the Intel Quartus Prime Power Analyzer.

Table 1. Comparison of the EPE and the Intel Quartus Prime Power Analyzer

Characteristic	EPE	Intel Quartus Prime Power Analyzer
When to use	Any time *Note:* For post-fit power analysis, you get better results with the Intel Quartus Prime Power Analyzer.	Post-fit
Software requirements	Spreadsheet program	The Intel 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 • Off-chip (non-thermal) power dissipation • Device supply currents
Estimation of transceiver power for dynamic reconfiguration features	Includes an estimation of the incremental power consumption by these features.	Not included

1.2. Power Estimations and Design Requirements

Power estimation and analysis helps you satisfy two important planning requirements:

- **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: For power supply planning, use the EPE at early stages of the design cycle. When the design is complete, you can use the Power Analyzer reports for an estimate of design power requirement.

Thermal and supply power requirements are similar, but not identical, because there are elements outside the device that also contribute to power dissipation, such as terminator resistors.

1.3. Power Analyzer Walkthrough

The Intel Quartus Prime Power Analyzer requires post-fit design.

You must either provide timing assignments for all clocks in the design, or generate activity data from a simulation-based flow. You must specify the I/O standard on each device input and output, and the board trace model on each output in the design.

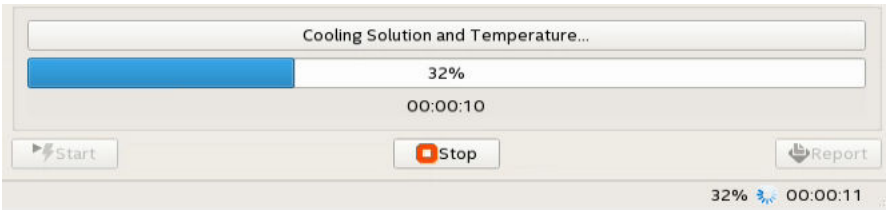
To run the Power Analyzer:

1. From the Intel Quartus Prime Software, open the Power Analyzer tool by clicking **Processing ➤** **Power Analyzer Tool**.
2. If you have signal activity information for the project, turn on **Use input files to initialize toggle rates and static probabilities during power analysis**, and then click **Add Power Input Files** to specify input files.
   For more information about generating those input files, refer to *Sources for Signal Activity Data*.
3. To direct the Power Analyzer to write a Signal Activity (`.saf`) output file, turn on **Write out signal activities used during power analysis**, and specify the file name.
4. To direct the Power Analyzer to generate an Early Power Estimation file, turn on **Write out Early Power Estimation file**, and specify the file name.

   With this file, you can import design information into the Early Power Estimator spreadsheet, and perform what-if analyses.
5. Specify the default toggle rate for input I/O signals.

   The Power Analyzer uses the default toggle rate when no other method specifies the signal-activity data.
6. Specify the default toggle rate for the remaining (non input) signals.
7. Define the cooling solution and temperature.
8. Click **Start**.

Figure 2. Progress Bar in Intel Quartus Prime Power Analyzer



9. When the tool finishes, click **Report** to open the Power Analyzer report.

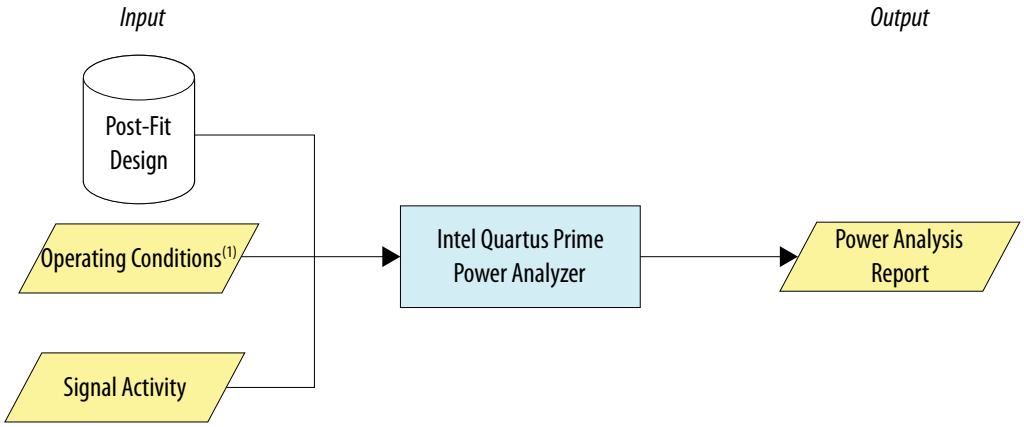
Related Information

Inputs for the Power Analyzer on page 7

1.4. Inputs for the Power Analyzer

The Power Analyzer supports accurate power estimations by allowing you to specify the important design factors affecting power consumption. The following figure shows the high-level flow of the Power Analyzer:

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 you to synthesize the design and then fit the design to the target device. Additionally, you must specify:

- 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.

1.4.1. Operating Settings and Conditions

You can specify device power characteristics, operating voltage conditions, and operating temperature conditions for power analysis in the Intel Quartus Prime software.

The **Operating Settings and Conditions** page of the **Settings** dialog box allows you to specify whether the device has typical power consumption characteristics or maximum power consumption characteristics.

The **Voltage** page of the **Settings** dialog box allows you to view the operating voltage conditions for each power rail in the device, and specify supply voltages for power rails with selectable supply voltages.

If the specifications in the **Voltage** page conflict with the voltage required for operation (for example, supply voltages for some transceiver depend on the data rate), the Fitter overrides the voltage for relevant rails with the value that the Fitter calculates. The Intel Quartus Prime Power Analyzer also uses the calculated value.

On the **Temperature** page of the **Settings** dialog box, you can specify the thermal operating conditions of the device.

Related Information

- Operating Settings and Conditions Page (Settings Dialog Box)
  In *Intel Quartus Prime Help*
- Voltage Page (Settings Dialog Box)
  In *Intel Quartus Prime Help*
- Temperature Page (Settings Dialog Box)
  In *Intel Quartus Prime Help*

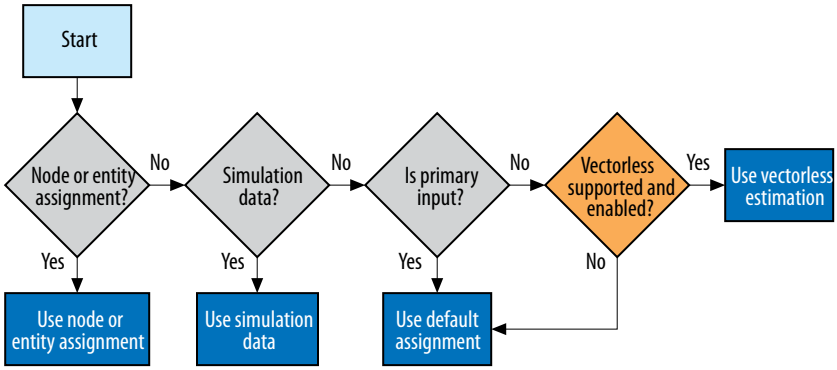
1.4.2. Sources for Signal Activity Data

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 activities from the following sources:

- Simulation results
- User-entered node, entity, and clock assignments
- User-entered default toggle rate assignment
- Vectorless estimation (selected devices)

You can mix and match the signal-activity data sources on a signal-by-signal basis. The following figure shows the priority scheme applied to each signal.

Figure 4. Signal-Activity Data Source Priority Scheme



1.4.2.1. Waveforms from Supported Simulators

The Power Analyzer can read waveforms generated by a supported design simulation. From the simulation waveform, the Power Analyzer calculates static probability and toggle rate can be calculated for each signal.

Note: Power analysis is most accurate when you use representative input stimuli to generate simulations.

The following simulators generate outputs that the Power Analyzer supports:

- ModelSim®
- ModelSim - Intel FPGA Edition
- QuestaSim
- Active-HDL
- NCSim
- VCS*
- VCS MX
- Riviera-PRO*

Related Information

Signal Activity on page 27

1.4.2.2. .vcd Files from Third-Party Simulation Tools

The Intel Quartus Prime Power Analyzer supports `.vcd` files generated by a simulation tool as the source of activity data.

A Verilog Value Change Dump File (`.vcd`) provides signal activity and static probability information. These files include all the routing resources and the exact logic array resource usage.

For third-party simulators, use the **EDA Tool Settings** to specify the Generate Value Change Dump (VCD) file script option in the **Simulation** page of the **Settings** dialog box. These scripts instruct the third-party simulators to generate a `.vcd` that encodes the simulated waveforms. The Intel Quartus Prime Power Analyzer reads this file directly to derive the toggle rate and static probability data for each signal.

Note: The Intel Quartus Prime software does not support a built-in simulator.

Other third-party EDA simulators can generate `.vcd` files that the Power Analyzer can read. For those simulators, you must manually create a simulation script to generate the `.vcd` file.

1.4.2.2.1. Generating a .vcd in a EDA Simulation Tool

To create a `.vcd` for the design, follow these steps:

1. On the **Assignments ➤ Settings**.
2. In the **Category** list, under **EDA Tool Settings**, click **Simulation**.
3. In the **Tool name** list, select the EDA simulator.

4. In the **Format for output netlist** list, select **Verilog HDL**, or **SystemVerilog HDL**, or **VHDL**.
5. Turn on **Generate Value Change Dump (VCD) file script**.

   This option turns on the **Map illegal HDL characters** and **Enable glitch filtering** options.

   The **Map illegal HDL characters** option ensures that all signals have legal names and makes signal toggle rates available to the Power Analyzer.
   The **Enable glitch filtering** option directs the EDA Netlist Writer to perform glitch filtering when generating VHDL Output Files, Verilog Output Files, and the corresponding Standard Delay Format Output Files for use with other EDA simulation tools. This option is available regardless of whether or not you want to generate `.vcd` scripts.

   Note: For ModelSim simulations , the `+nospecify` option in the `vsim` command disables the **Specify path delays and timing checks** option. By enabling glitch filtering on the **Simulation** page, the simulation models include specified path delays. Thus, ModelSim might fail to simulate a design. As a best practice, remove the `+nospecify` option from the ModelSim `vsim` command to ensure accurate simulation for power estimation.
6. Click **Script Settings**. Select the signals that you want to write to the `.vcd`.
   - If you choose **All signals**, the generated script instructs the third-party simulator to write all connected output signals to the `.vcd` file.
   - If you choose **All signals except combinational lcell outputs**, the generated script instructs the third-party simulator to write all connected output signals to the `.vcd`, except logic cell combinational outputs.

   Note: The file can become extremely large if you write all output signals to the file, because the file size depends on the number of output signals being monitored and the number of transitions that occur.
7. Click **OK**.
8. In the **Design instance name** box, type a name for the testbench.
9. Compile the design with the Intel Quartus Prime software, and generate the necessary EDA netlist and script that instructs the third-party simulator to generate a `.vcd`.
10. In the third-party EDA simulation tool, call the generated script in the simulation tool before running the simulation.
11. Perform the simulation.

The simulation tool generates the `.vcd` file in the project directory.

1.4.2.2.2. Generating a .vcd from ModelSim Software

To generate a `.vcd` with the ModelSim software, follow these steps:

1. In the Intel Quartus Prime software, on the Assignments menu, click **Settings**.
2. In the **Category** list, under **EDA Tool Settings**, click **Simulation**.
3. In the **Tool name** list, select your preferred EDA simulator.
4. In the **Format for output netlist** list, select **Verilog HDL**, or **SystemVerilog HDL**, or **VHDL**.
5. Turn on **Generate Value Change Dump (VCD) file script**.

6. To generate the `.vcd`, perform a full compilation.
7. In the ModelSim software, compile the files necessary for simulation.
8. Load your design by clicking **Start Simulation** on the Tools menu, or use the `vsim` command.
9. Use the `.vcd` script created in 6 on page 11 using the following command: `source <design>_dump_all_vcd_nodes.tcl`
10. Run the simulation (for example, run 2000ns or run -all).
11. Quit the simulation using the `quit -sim` command, if required.
12. Exit the ModelSim software.
    If you do not exit the software, the ModelSim software might end the writing process of the **.vcd** improperly, resulting in a corrupt `.vcd`.

1.4.2.3. 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.

1.4.2.3.1. 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.

1.4.2.4. 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.

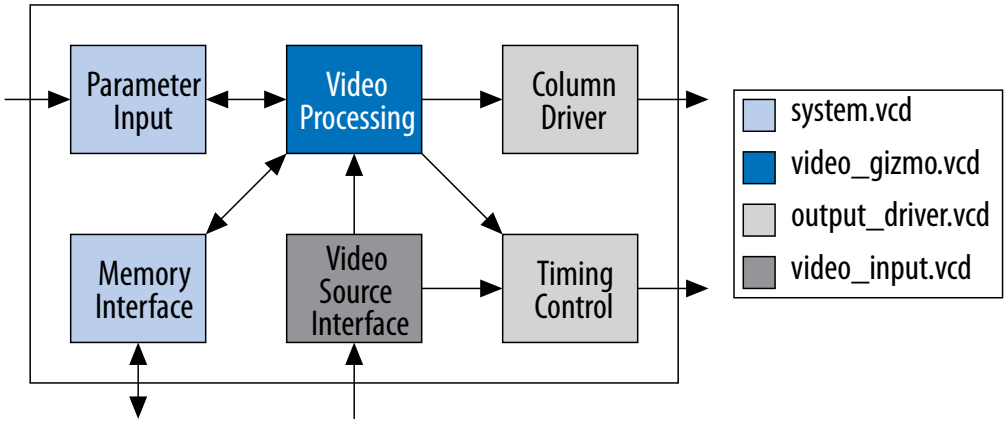
1.4.2.5. Signal Activities from User Defaults Only

The user defaults only flow provides the lowest degree of accuracy.

1.5. Power Analysis in Modular Design Flows

A common design practice is to create modular or hierarchical designs in which you develop each design entity separately, and then instantiate these modules in a higher-level entity to form a complete design. You can perform simulation on a complete design or on each module for verification. The Power Analyzer supports modular design flows when reading the signal activities from simulation files. The following figure shows an example of a modular design flow.

Figure 5. Modular Simulation Flow

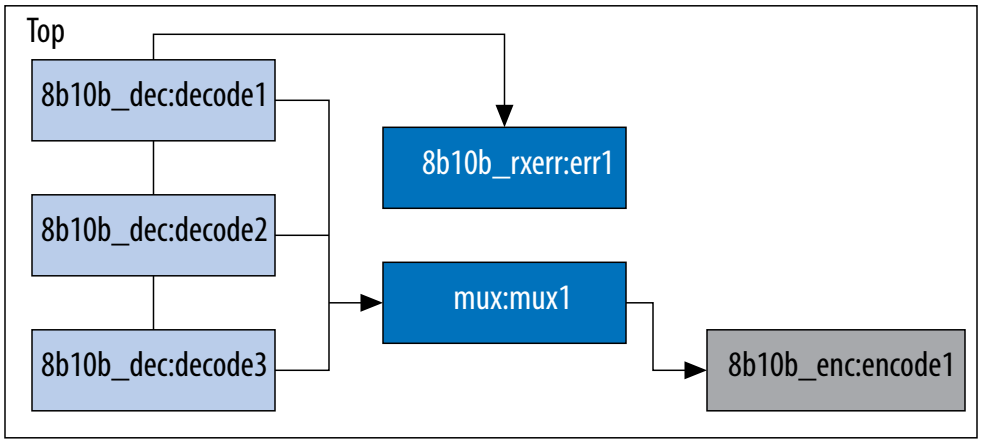


The Power Analyzer supports associating multiple `.vcd` simulation files to specific entity names, enabling the integration of partial design simulations into a complete design power analysis. When specifying multiple `.vcd` files for the design, more than one simulation file can contain signal-activity information for the same signal. In those cases, the Power analyzer follows these rules:

- When you apply multiple `.vcd` files to the same design entity, the Power Analyzer calculates the signal activity as the equal-weight arithmetic average of each `.vcd`.
- When you apply multiple simulation files to design entities 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 entity.

The following figure shows an example of a hierarchical design:

Figure 6. 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

You can simulate the entire 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

You can independently simulate of 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 2. 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 entity. Simulation provides signal activities for the pins and for the outputs of functional blocks. If the inputs to an entity 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 entity such as `mux1` has its signal activity specified at the output of one of the decode entities.

1.5.3. Multiple Simulations on the Same Entity

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 3. 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 Simulations

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 4. 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 Simulations

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 when Performing Signal-Activity Calculations using the Limit VCD Period Option

To specify a start and end time when performing signal-activity calculations using the **Limit VCD period** option, follow these steps:

1. In the Intel 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
```

Related Information

- set_power_file_assignment on page 0
  In *Intel Quartus Prime Help*
- Add/Edit Power Input File Dialog Box on page 0
  In *Intel Quartus Prime Help*

1.5.6. Node Name Matching Considerations

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 Intel Quartus Prime projects might not match their node names with the current Intel Quartus Prime project.

For example, you may have a file named `8b10b_enc.vcd`, which the Intel 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 Intel Quartus Prime software might name all the combinational nodes in the `8b10b_enc.vcd` differently than in the `Top` project.

You can avoid name mismatching with only RTL simulation data, in which register names do not change, or with an incremental compilation flow that preserves node names along with a gate-level simulation.

Note: To ensure accuracy, Intel FPGA recommends that you use an incremental compilation flow to preserve the node names of your design.

1.5.7. 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 Intel 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.

1.5.7.1. Enabling Tool Based Glitch Filtering

To enable the first level of glitch filtering in the Intel Quartus Prime software for supported third-party simulators, follow these steps:

1. In the Intel Quartus Prime software, click **Assignments ➤ Settings**.
2. In the **Category** list, select **Simulation under EDA Tool Settings**.
3. Select the **Tool name** to use for the simulation.
4. Turn on **Enable glitch filtering**.

1.5.7.2. Enabling Glitch Filtering During Power Analysis

The second level of glitch filtering occurs while the Power Analyzer is reading the `.vcd` generated by a third-party simulator. To enable the second level of glitch filtering, follow these steps:

1. In the Intel Quartus Prime software, click **Assignments ➤ Settings**.
2. In the **Category** list, select **Power Analyzer Settings**.
3. Under **Input File(s)**, turn on **Perform glitch filtering on VCD files**.

The `.vcd` file reader performs filtering complementary to the filtering performed during simulation and is often not as effective. While the `.vcd` file reader can remove glitches on logic blocks, the file reader cannot determine how a given glitch affects downstream logic and routing, and may eliminate the impact of the glitch completely. 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.

1.5.8. Node and Entity Assignments

You can assign toggle rates and static probabilities to individual nodes and entities 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 Intel Quartus Prime software issues a warning and ignores the assignment.

Assigning toggle rates and static probabilities to individual nodes and entities is appropriate for signals in which you have knowledge of the signal or entity 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.

1.5.8.1. Timing Assignments to Clock Nodes

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.

1.5.9. Default Toggle Rate Assignment

You can specify a default toggle rate for primary inputs and other nodes in your design. The Power Analyzer uses the default toggle rate when no other method specifies the signal-activity data.

The Power Analyzer specifies 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 either the Power Analyzer cannot determine a clock domain for the node, or the clock domain is ambiguous. For example, the Power Analyzer may not be able to determine a clock domain for a node if the user did not specify sufficient timing assignments. In these cases, the Power Analyzer substitutes and reports a toggle rate of zero.

Related Information

Toggle Rate on page 28

1.5.10. Vectorless Estimation

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. Power Analyzer Compilation Report

The Power Analyzer Compilation Report 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.

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 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.

Thermal Power Dissipation by Block Type (Device Resource Type)

This Thermal Power Dissipation by Block Type (Device Resource Type) section of the report shows the estimated thermal dynamic power and thermal static power consumption categorized by block types. This information is further categorized by estimated dynamic and static power and provides an average toggle rate by block type. Thermal power is the power dissipated as heat from the FPGA device.

Thermal Power Dissipation by Hierarchy

This Thermal Power Dissipation by Hierarchy section of the report shows estimated thermal dynamic power and thermal static power consumption categorized by design hierarchy. This information is further categorized by the dynamic and static power that was used by the blocks and routing in that hierarchy. This information is useful when locating modules with high power consumption in your design.

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 from Voltage Supplies

The Current Drawn from Voltage Supplies 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, such as the V_{CCPD} power rail in Stratix® III and Stratix IV devices, and the V_{CCIO} power rail in Stratix IV devices.

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 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.

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 Intel Quartus Prime software generates during the analysis.

1.7. 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 Intel 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

- Tcl Scripting
  In *Intel Quartus Prime Standard Edition Handbook Volume 2*
- API Functions for Tcl on page 0
  In *Intel Quartus Prime Help*

1.7.1. 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 Intel Quartus Prime project called *sample*.

- To instruct the Power Analyzer to generate a EPE File:

```
quartus_pow sample --output_epe=sample.csv ←
```

- To instruct the Power Analyzer to generate a EPE File without performing the power estimate:

```
quartus_pow sample --output_epe=sample.csv --estimate_power=off  ←
```

- 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 to not use an input file, a default input I/O toggle rate of 60%, no vectorless estimation, 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 **Power Analyzer Settings Operating Conditions** page. Use the Intel 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

Power Analyzer Compilation Report on page 18

1.8. Power Analysis Revision History

The following revision history applies to this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • General chapter reorganization. • Moved *Factors Affecting Power Consumption* to chapter: *Power Optimization*.

continued...

Document Version	Intel Quartus Prime Version	Changes
		• 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*
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.
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.
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.

continued...

Document Version	Intel Quartus Prime Version	Changes
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.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

2. Power Optimization

The Intel 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 describes the power-driven compilation feature and flow in detail, as well as low power design techniques that can further reduce power consumption in your design. The techniques primarily target Arria, Stratix, and Cyclone series of devices. These devices utilize a low-k dielectric material that dramatically reduces dynamic power and improves performance. Arria series, Stratix IV, and Stratix V device families include efficient logic structures called adaptive logic modules (ALMs) that obtain maximum performance while minimizing power consumption. Cyclone device families offer the optimal blend of high performance and low power in a low-cost FPGA.

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

- AN 514: Power Optimization in Stratix IV FPGAs
- Power Analysis on page 4
- AN 711: Power Reduction Features in Intel 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:

Design Activity and Power Analysis on page 25
Device Selection on page 26
Environmental Conditions on page 26
Device Resource Usage on page 27
Signal Activity on page 27

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

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

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 5. 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: $T_{Junction}=T_{Ambient}+P_{Thermal}\cdot\theta_{A}$

continued...

Environmental Condition	Description
	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. For example, Stratix V devices support global clocks and quadrant (regional) clocks. 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 Intel 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 Intel 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.

Related Information

Default Toggle Rate Assignment on page 18

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).

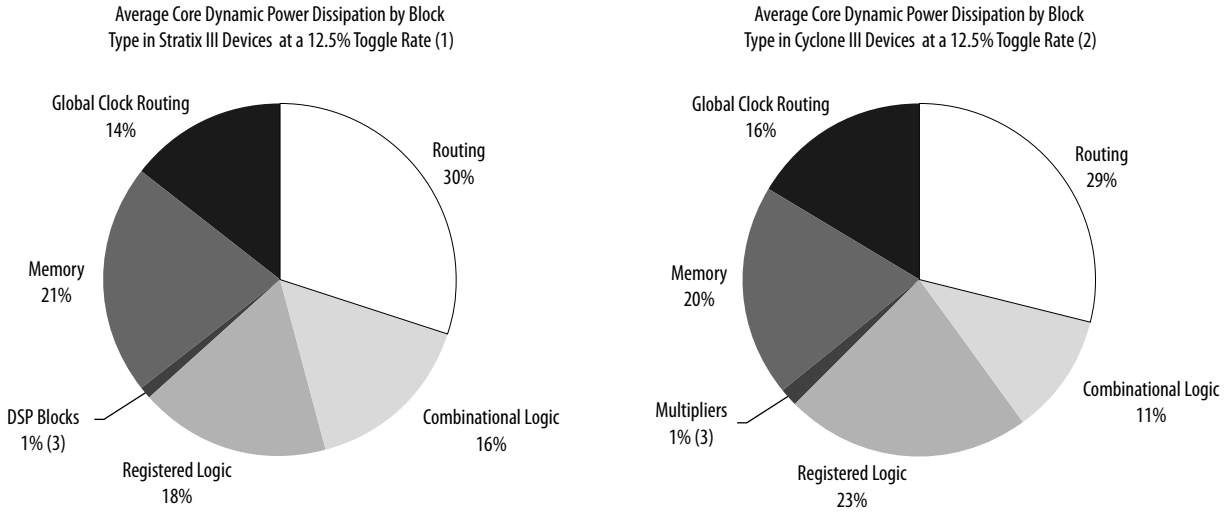
The static probability of input signals impacts the design's static power consumption, due to state-dependent leakage in routing and logic. This effect becomes more important for smaller geometries. In output I/O standards that drive termination resistors, the static power also depends on the static probability on I/O pins.

2.2. Power Dissipation

You can refine techniques that reduce power consumption in a design by understanding the sources of power dissipation.

The following figure shows the power dissipation of Stratix and Cyclone devices in different designs. The analysis considers a fixed clock rate of 100 MHz and exhibits varied logic resource utilization across available resources.

Figure 7. Average Core Dynamic Power Dissipation



Notes:

1. These results originate from 103 designs.
2. These results originate from 96 designs.
3. In designs using DSP blocks, DSPs consumed 5% of core dynamic power.

In Stratix and Cyclone device families, a series of column and row interconnect wires of varying lengths provide signal interconnections between logic array blocks (LABs), memory block structures, and digital signal processing (DSP) blocks or multiplier blocks. These interconnects dissipate the largest component of device power.

FPGA combinational logic is another source of power consumption. For more information about ALMs and LEs in Cyclone or Stratix devices, refer to the respective device handbook.

Memory and clock resources are other major consumers of power in FPGAs. Stratix devices feature the TriMatrix memory architecture. TriMatrix memory includes 512-bit M512 blocks, 4-Kbit M4K blocks, and 512-Kbit M-RAM blocks, which are configurable to support many features. Stratix IV TriMatrix on-chip memory is an enhancement based upon the Stratix II FPGA TriMatrix memory and includes three sizes of memory blocks: MLAB blocks, M9K blocks, and M144K blocks. Stratix IV and Stratix V devices feature Programmable Power Technology, an advanced architecture that enables a smooth trade-off between speed and power. The core of each Stratix IV and Stratix V device is divided into tiles, each of which may be put into a high-speed or low-power mode. The primary benefit of Programmable Power Technology is to reduce static power, with a secondary benefit being a small reduction in dynamic power. Cyclone IV GX devices have 9-Kbit M9K memory blocks.

2.3. 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 Intel 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 *Intel Quartus Prime Standard Edition User Guide: Design Optimization*
- Launch Design Space Explorer Command (Tools Menu)
  In *Intel Quartus Prime Help*

2.4. Power-Driven Compilation

The standard Intel 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.

Intel 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.

2.4.1. 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 6. 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 Intel 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 7. 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

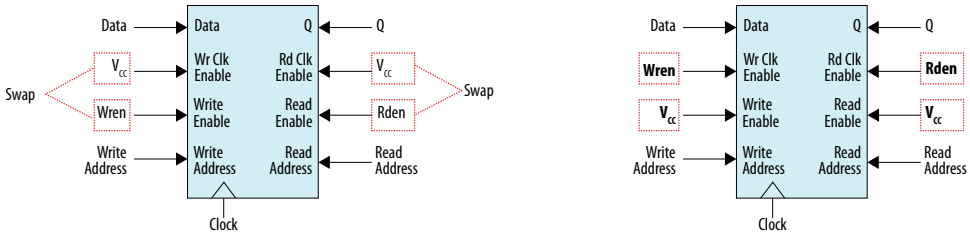
- Clock Enable in Memory Blocks on page 38
- Intel Quartus Prime Compiler Settings on page 35

2.4.1.1. 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 8. 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.

For Stratix IV and Stratix V devices, the memory transformation takes place at the Fitter level by selecting the **Normal compilation** settings for the power optimization option.

In Cyclone IV GX and StratixIV devices, the read-during-write behavior impacts the power of single-port and bidirectional dual-port RAMs. As a best practice, you can allow optimization by setting the read-during-write parameter to "Don't care" at the HDL level, and set 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.

2.4.1.2. Power-Aware Logic Mapping

Power-aware logic mapping reduces power by rearranging the logic during synthesis to eliminate nets with high switching rates.

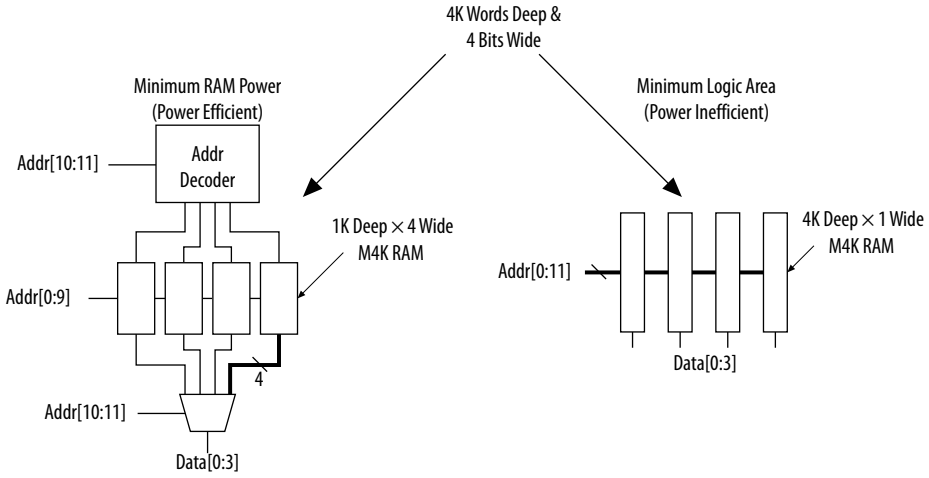
2.4.1.3. Power-Aware Memory Balancing

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**.

The following figure is an example of a 4k × 4 (4k deep and 4 bits wide) memory implementation in two different configurations using M4K memory blocks available in some Stratix devices.

Figure 9. 4K × 4 Memory Implementation Using Multiple M4K Blocks



The minimum logic area implementation configures M4K blocks as 4k × 1. The Intel Quartus Prime software uses this implementation as the default, because the resulting design has the minimum logic area (0 logic cells) and the highest speed. However, all four M4K blocks are active on each memory access, which increases RAM power.

The minimum RAM power implementation configures four M4K blocks as 1k × 4 for optimal power saving. The RAM IP core includes an address decoder to select which of the four M4K blocks is active on a given cycle, based on the state of the top two user address bits. The RAM IP core implements a multiplexer to feed the downstream logic

by choosing the appropriate M4K output. This implementation reduces RAM power because only one M4K block is active on any cycle, but it requires extra logic cells, costing logic area and potentially impacting design performance.

There is a trade-off between power saved by accessing fewer memories and power consumed by the extra decoder and multiplexor logic. The Intel 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

Maximum Block Depth Configuration
In *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*

2.4.2. Power-Driven Fitter

The Intel 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 8. 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.

For Stratix IV and Stratix V devices, the **Normal compilation** setting enables the Programmable Power Technology to configure tiles as high-speed mode or low-power mode. Programmable Power Technology is always turned **ON** even when the **OFF** setting is selected for the **Power optimization** option. Tiles are the combination of LAB and MLAB pairs (including the adjacent routing associated with LAB and MLAB), which can be configured to operate in high-speed or low-power mode. This level of power optimization does not have any affect on the fitting, timing results, or compile 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 Intel 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. The benchmark data shows that the power-driven Fitter technique can reduce power consumption by as much as 19% in Stratix devices. On average, you can reduce core dynamic power by 16% with the Extra effort synthesis and Extra effort fitting settings, as compared to the **Off** settings in both synthesis and Fitter options for power-driven compilation.

Related Information

- AN 514: Power Optimization in Stratix IV FPGAs
- Power Analyzer Settings Page (Settings Dialog Box) on page 0
  In *Intel Quartus Prime Help*
- Value Change Dump File (.vcd) Definition on page 0
  In *Intel Quartus Prime Help*
- Assignment Editor Options on page 36

2.4.3. 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 Intel 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.

The benchmark data shows that the area-driven technique can reduce power consumption by as much as 31% in Stratix devices and as much as 15% in Cyclone devices.

Related Information

Assignment Editor Options on page 36

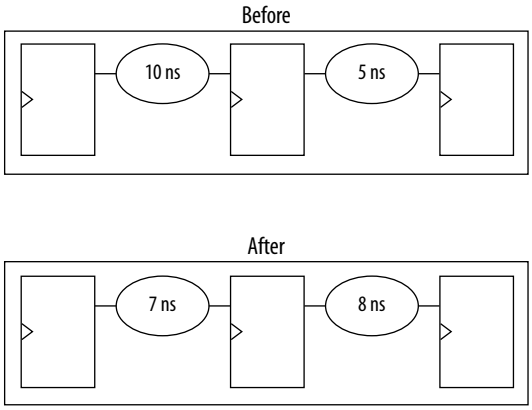
2.4.4. Gate-Level Register Retiming

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 Intel 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 10. 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.

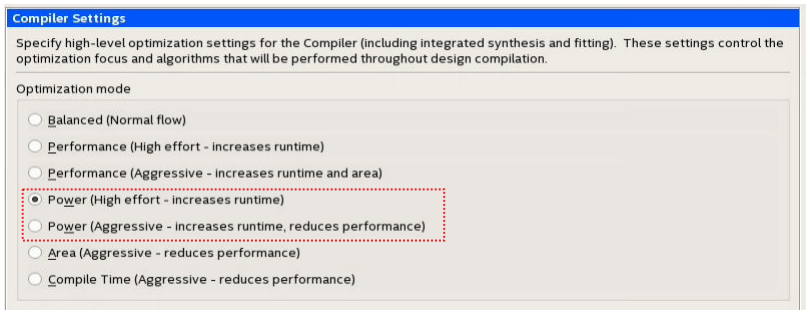
When using Intel Quartus Prime integrated synthesis, retiming occurs during synthesis before the design is mapped to Intel primitives. The benchmark data shows that the combination of WYSIWYG remapping and gate-level register retiming techniques can reduce power consumption by as much as 6% in Stratix devices and as much as 21% in Cyclone devices.

2.4.5. Intel Quartus Prime Compiler Settings

The Intel Quartus Prime software provides settings that optimize power for the full design.

To set the optimization mode on the Intel Quartus Prime software, click **Assignments ➤ Settings ➤ Compiler Settings**.

Figure 11. Compiler Settings for Power Optimization



The two power optimization modes direct the Compiler to prioritize one optimization metric.

Power (High effort—increases runtime)

High effort modes enable additional optimizations that increase compilation time and do not affect design performance. High Power Effort mode guides the Compiler to spend additional compilation time reducing routing utilization, which saves dynamic power.

Power (Aggressive—increases runtime, reduces performance)

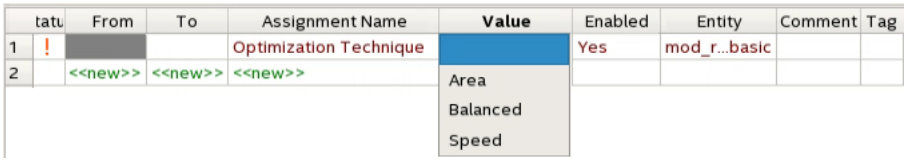
Aggressive modes increase compilation time and make trade-offs that may harm other optimization metrics (performance, area, etc.). In Aggressive Power mode, the Compiler attempts to reduce the routing usage of signals with the highest specified (via Signal Activity File) or estimated toggle rates, saving additional dynamic power but potentially affecting performance.

2.4.6. 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 12. Optimization Technique Options



The **Power Optimization During Synthesis** logic option determines how aggressively Analysis & Synthesis optimizes the design for power.

Figure 13. Power Optimization During Synthesis Options

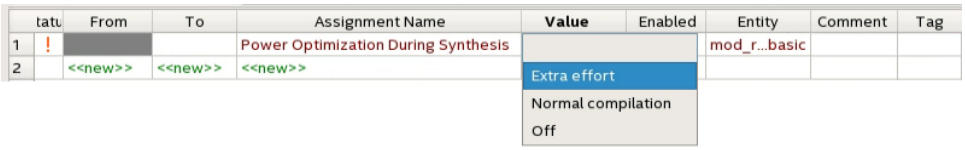


Table 9. Power Optimization During Synthesis Options

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

Related Information

- Area-Driven Synthesis on page 34
- Power-Driven Fitter on page 33

2.5. Design Guidelines

During FPGA design implementation, you can apply the following design techniques to reduce power consumption. This section provides detailed design techniques for Cyclone IV GX devices that affect overall design power. The results of these techniques are different from design to design.

2.5.1. 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 Intel Quartus Prime software optimizes clock routing power automatically, enabling only those portions of the clock network that are necessary to feed downstream registers.

Related Information

Clock Control Block IP Core User Guide (ALTCLKCTRL)

2.5.1.1. Clock Enable in Memory Blocks

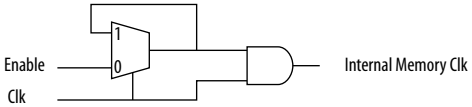
In memory blocks, power consumption is tied to the clock rate, and is insensitive to the toggle rate on the data and address lines. Memory consumes approximately 20% of the core dynamic power in typical designs.

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.

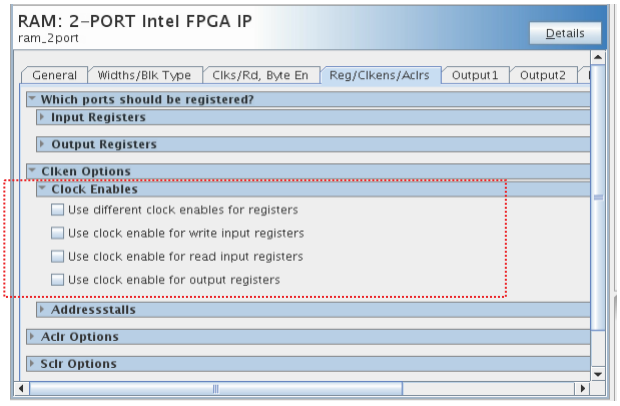
Figure 14. Memory Clock Enable Signal

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 15. Clock Enable in RAM 2-Port

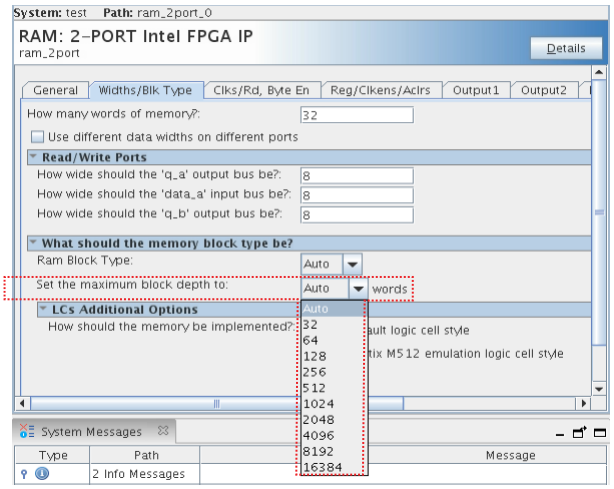


For example, consider a design that contains a 32-bit-wide M4K memory block in ROM mode that is running at 200 MHz. Assuming that the output of this block is only required approximately every four cycles, this memory block consumes 8.45 mW of dynamic power according to the demands of the downstream logic. By adding a small amount of control logic to generate a read clock enable signal for the memory block only on the relevant cycles, the power can be cut 75% to 2.15 mW.

You can also use the MAXIMUM_DEPTH parameter in your memory IP core to save power in Cyclone IV GX, Stratix IV, and Stratix V devices; however, this approach might increase the number of LEs required to implement the memory and affect design performance.

The Intel 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 16. RAM 2-Port Maximum Depth



Related Information

- Power-Driven Compilation on page 30
- Clock Power Management on page 37
- Clocking Modes and Clock Enable
  In *Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide*

2.5.1.1.1. Memory Power Reduction Example

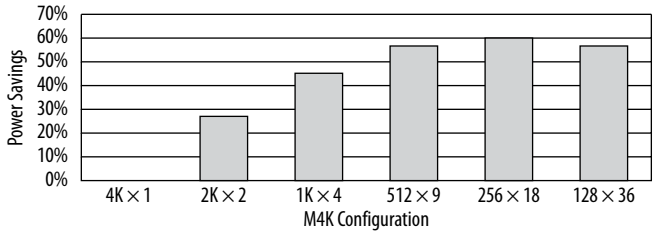
Power usage measurements for a 4K × 36 simple dual-port memory implemented using multiple M4K blocks in a Stratix device. For each implementation, the M4K blocks are configured with a different memory depth.

Table 10. 4K × 36 Simple Dual-Port Memory Implemented Using Multiple M4K Blocks

M4K Configuration	Number of M4K Blocks	ALUTs
4K × 1 (Default setting)	36	0
2K × 2	36	40
1K × 4	36	62
512 × 9	32	143
256 × 18	32	302
128 × 36	32	633

Using the MAXIMUM_DEPTH parameter can save power. For all implementations, a user-provided read enable signal is present to indicate when read data is required. Using this power-saving technique can reduce power consumption by as much as 60%.

Figure 17. Power Savings Using the MAXIMUM_DEPTH Parameter



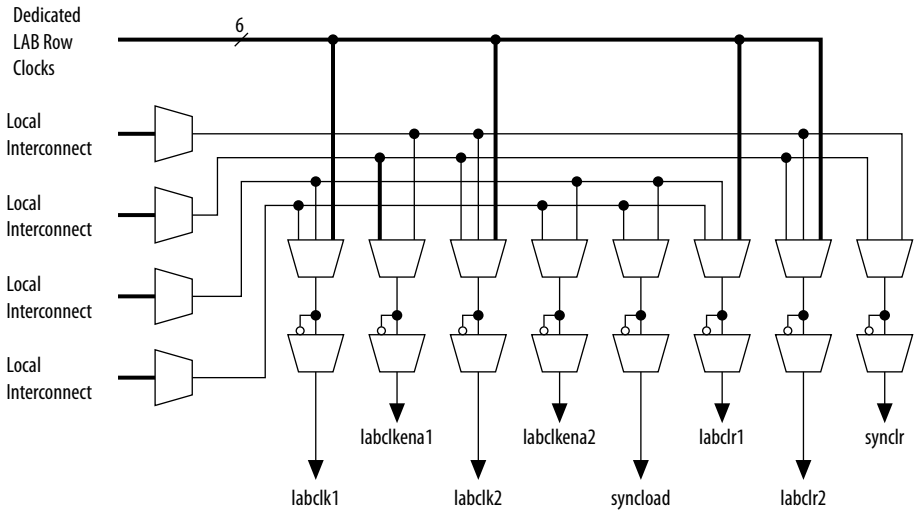
As the memory depth becomes more shallow, memory dynamic power decreases because unaddressed M4K blocks can be shut off using a decoded combination of address bits and the read enable signal. For a 128-deep memory block, power used by the extra LEs starts to outweigh the power gain achieved by using a more shallow memory block depth. The power consumption of the memory blocks and associated LEs depends on the memory configuration.

Note: The SOPC Builder and Platform Designer (Standard) system do not offer specific power savings control for on-chip memory block. There is no read enable, write enable, or clock enable that you can enable in the on-chip RAM megafunction to shut down the RAM block in the SOPC Builder and Platform Designer (Standard) system.

2.5.1.2. 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 18. 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 Intel 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.

2.5.1.2.1. 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.

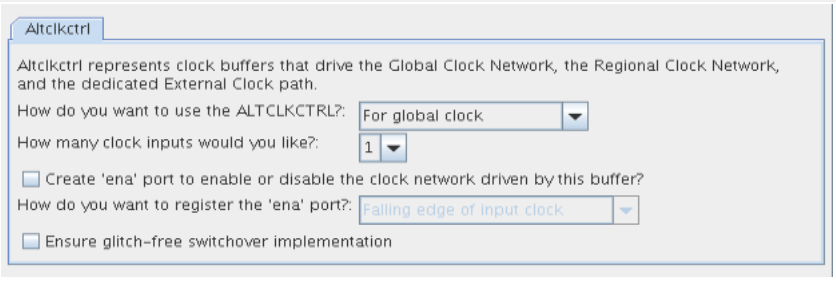
```
IF clk'event AND clock = '1' THEN
    IF logic_is_enabled = '1' THEN
        reg <= value;
    ELSE
        reg <= reg;
    END IF;
END IF;
```

2.5.1.3. Clock Enables

Use clock enables instead 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
```

```
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
```

2.5.1.4. 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.

Related Information

Device Page (Settings Dialog Box) on page 0
In *Intel Quartus Prime Help*

2.5.1.4.1. Viewing Clock Details in the Chip Planner

1. Open the Chip Planner (**Tools ➤ Chi Planner**).
2. In the **Task** pane, under **Clock Reports**, double-click **Report Clock Details**.

Figure 21. Chip Planner Task Pane

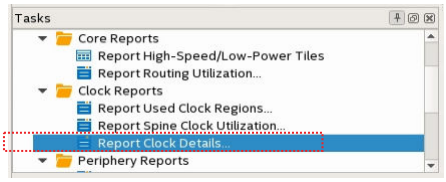
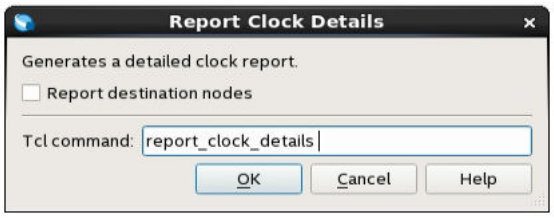


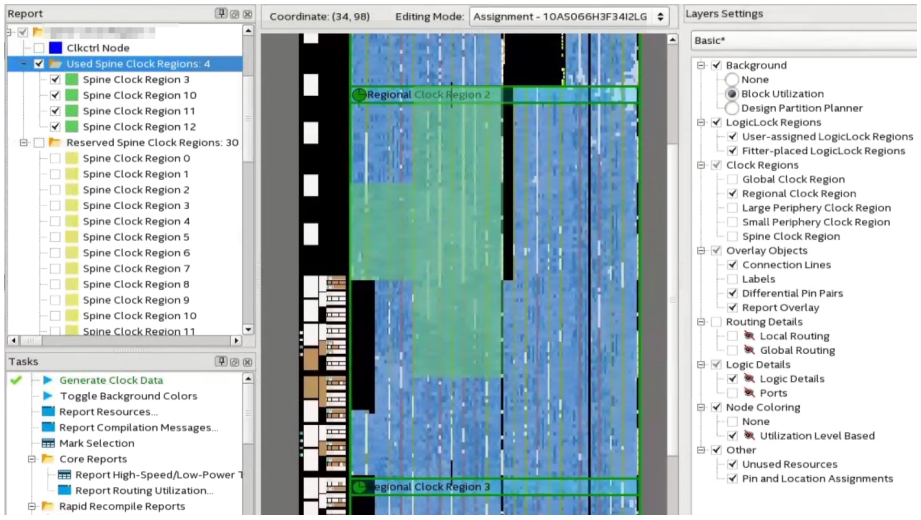
Figure 22. Report Clock Details



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 23. Clock Highlight in Chip Planner

This example uses a Regional clock Region instead of a global signal.



2.5.1.5. 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

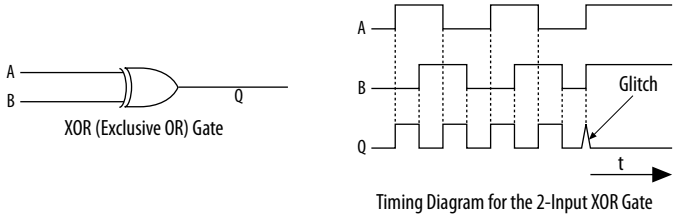
  clk: 365Mhz, Fanout 51277

2.5.2. 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 24. XOR Gate Showing Glitch at the Output

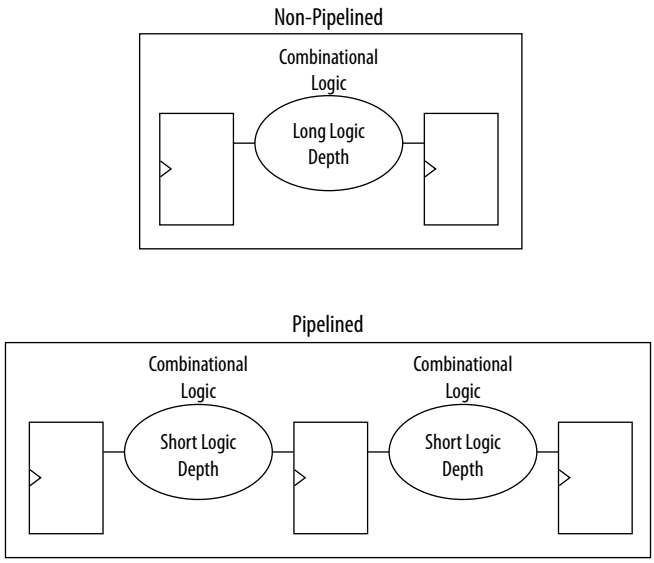


A glitch can propagate to subsequent logic and create unnecessary switching activity, increasing power consumption. Circuits with many XOR functions, such as arithmetic circuits or cyclic redundancy check (CRC) circuits, tend to have many glitches if there are several levels of combinational logic between 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 25. 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. Benchmark data shows that pipelining can reduce dynamic power consumption by as much as 30% in Cyclone and Stratix devices.

2.5.3. 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.

The Stratix device family allows you to efficiently target small, medium, and large memories with the TriMatrix memory architecture. Each TriMatrix memory block is optimized for a specific function. M512 memory blocks are more power-efficient than the distributed memory structures in some competing FPGAs. With M4K memory blocks you can implement buffers for a wide variety of applications, including processor code storage, large look-up table implementation, and large memory applications. The M-RAM blocks are useful in applications when storing a large volume of data on-chip is necessary. Effective utilization of these memory blocks can have a significant impact on power reduction in the design.

The latest Stratix and Cyclone device families have configurable M9K memory blocks that provide memory functions such as RAM, FIFO buffers, and ROM.

2.5.4. 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. For Cyclone IV GX, Stratix IV, and Stratix V devices, Advanced I/O Timing considers the full board trace model.

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

Stratix Series FPGA I/O Connectivity

2.5.5. 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 26. Fitter RAM Summary Report

Fitter RAM Summary

<<Filter>>

	Port A Depth	rt A Wid	rt B Dep	t 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.

2.5.5.1. Implementation

Table 11. Single-port Embedded Memory Configurations for 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 (1)	x8, x9, x10
M20K	512	x40, x32
	1K	x20, x16
	2K	x10, x8
	4K	x5, x4
	8K	x2
	16K	x1

(1) Supported through software emulation and consumes additional MLAB blocks.

Figure 27. 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

2.5.5.2. 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.

						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

2.5.6. DDR Memory Controller Settings

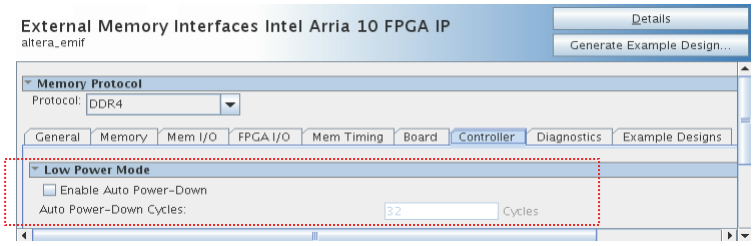
The Intel Arria 10 EMIF IP DDR3 controller provides low power mode options. These options put DDR in Power saving mode when the controller is idle, providing power savings on External Memory DDR. These options are **Enable Auto Power-Down** and **Auto Power-Down Cycles**.

Power-Down Mode

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 Number of cycles the controller must be IDLE before entering 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 Intel Arria 10 device family supports from 1 to 65534 cycles.

Figure 28. Intel Arria 10 EMIF Controller Parameters



Self-Refresh

Directs the Controller to self-refresh when not sending traffic for very long period. Self-refresh takes more time compared to power down, but the power saving is higher than power down.

Related Information

Intel Arria 10 EMIF IP DDR3 Parameters: Controller
In *External Memory Interfaces Intel Arria 10 FPGA IP User Guide*

2.5.7. 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 1. Implement Multiplier + Accumulator in 1 DSP

```
always @ (posedge clk)
begin
   if (ena)
   begin
      dataout <= dataa * datab + datac * datad;
   end
end
```

Example 2. 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

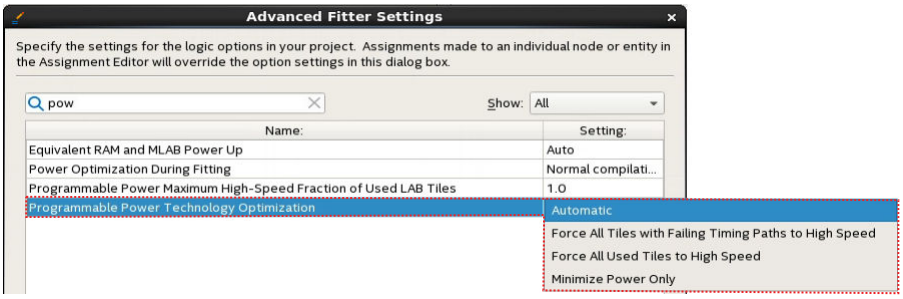
Inferring Multipliers and DSP Functions
In *Intel Quartus Prime Standard Edition User Guide: Design Recommendations*

2.5.8. Reducing High-Speed Tile (HST) Usage

High-Speed tiles are available in Stratix V and Intel Arria 10 device families.

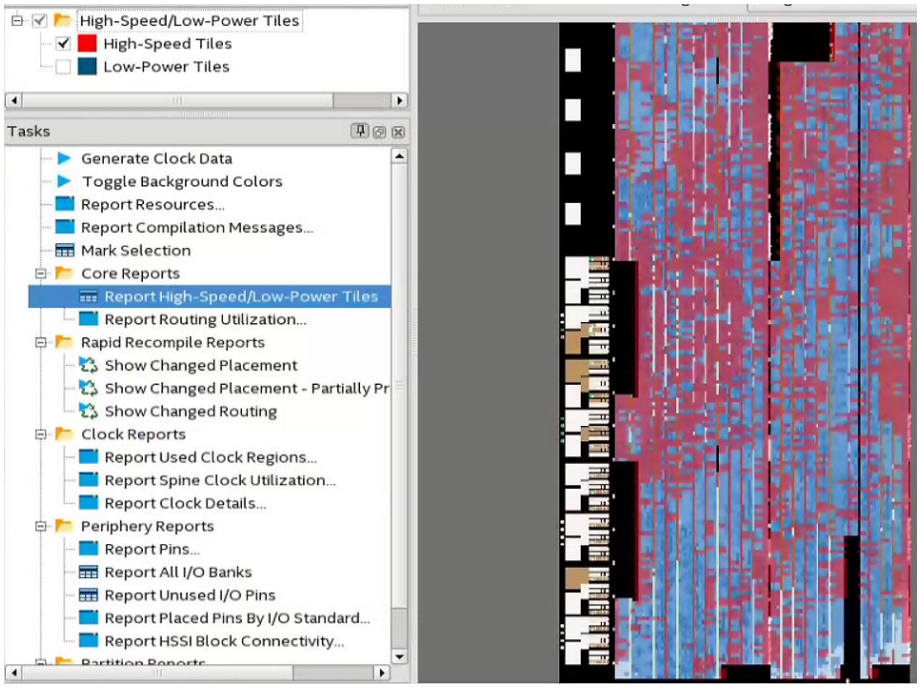
1. In the **Advanced Fitter Settings** pane, The **Programmable Power Technology Optimization** logic option controls how the fitter configures tiles to operate in high-speed mode or low-power mode. Select **Minimize Power Only**.

Figure 29. 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 30. Entity Modules and HST Heatmap on the Chip Planner



2.5.9. Unused Transceiver Channels

Transceivers in the device degrade over time unless you preserve them. The Intel 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.

2.5.10. Periphery Power reduction XCVR Settings

2.5.10.1. 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

2.5.10.2. I/O Current Strength

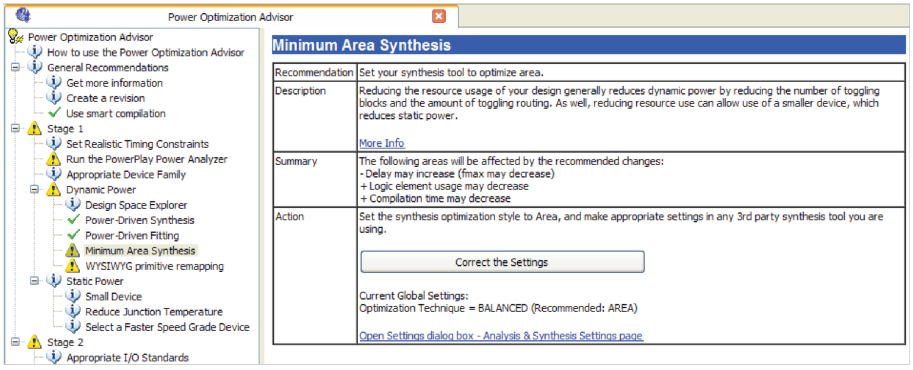
As a best practice, choose a low voltage I/O standard and the lowest drive strength that meets the speed requirements.

2.6. Power Optimization Advisor

The Intel 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 31. Power Optimization Advisor

Power Optimization Advisor after compiling a design that is not fully optimized for power.



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 Intel 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.

Related Information

Advisors in the Intel Quartus Prime Software on page 0
In *Intel Quartus Prime Help*

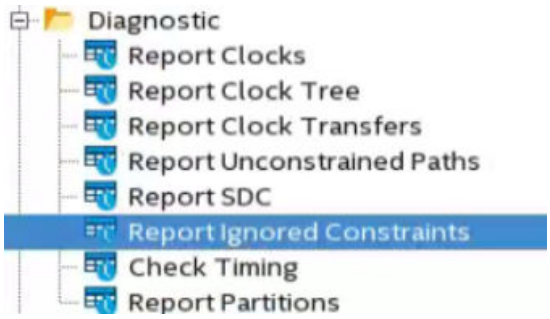
2.6.1. Set Realistic Timing Constraints

Timing requirements are too high, the Compiler increases HST Usage. In addition, the Fitter efforts focus more in timing than power optimization.

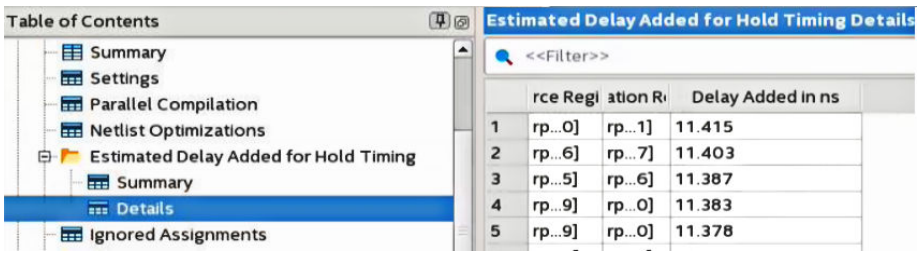
2.6.1.1. Find Timing Information

- To find False or Multi-Cycle Paths, click **Report Ignored Constraints** in the Timing Analyzer **Tasks** pane.

Figure 32. 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**.



2.6.2. Appropriate Device Family

Choose a device family with the dynamic and static power characteristics best suited to your application.

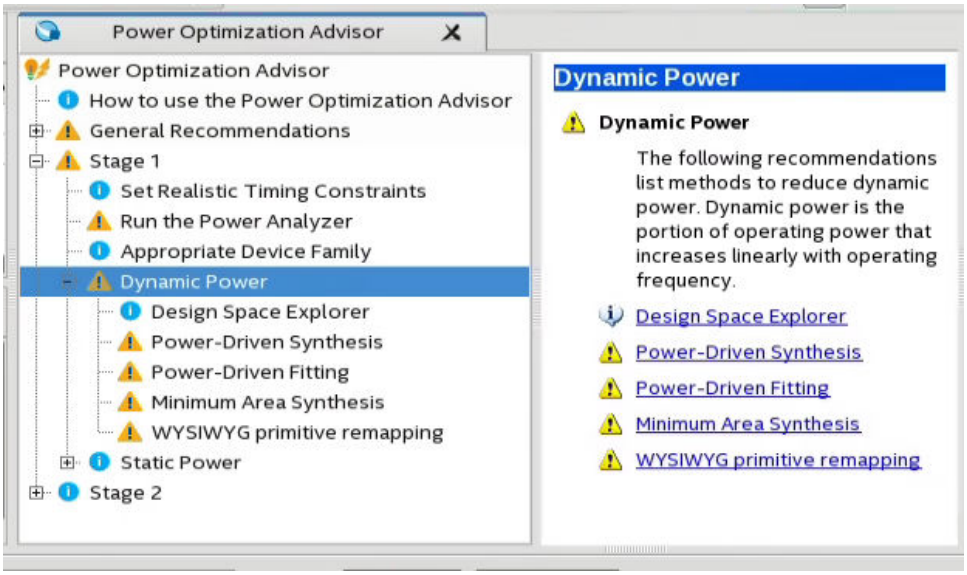
Related Information

- Device Selection on page 26
- Device Page (Settings Dialog Box) on page 0
  In *Intel Quartus Prime Help*

2.6.3. Dynamic Power

The recommendations in this section can reduce dynamic power.

Figure 33. Dynamic Power Recommendations in the Power Optimization Advisor



Related Information

- Design Space Explorer II for Power-Driven Optimization on page 29
- Power-Driven Synthesis on page 30
- Power-Driven Fitter on page 33
- Area-Driven Synthesis on page 34

2.6.4. 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 26
- Device Page (Settings Dialog Box) on page 0
  In *Intel Quartus Prime Help*

2.6.5. Appropriate I/O Standards

Choose appropriate I/O Standards to minimize design power.

Related Information

I/O Power Guidelines on page 45

2.6.6. 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 46

2.6.7. 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 38
- Memory Optimization (M20K/MLAB) on page 46

2.6.8. Clock Enables on Logic

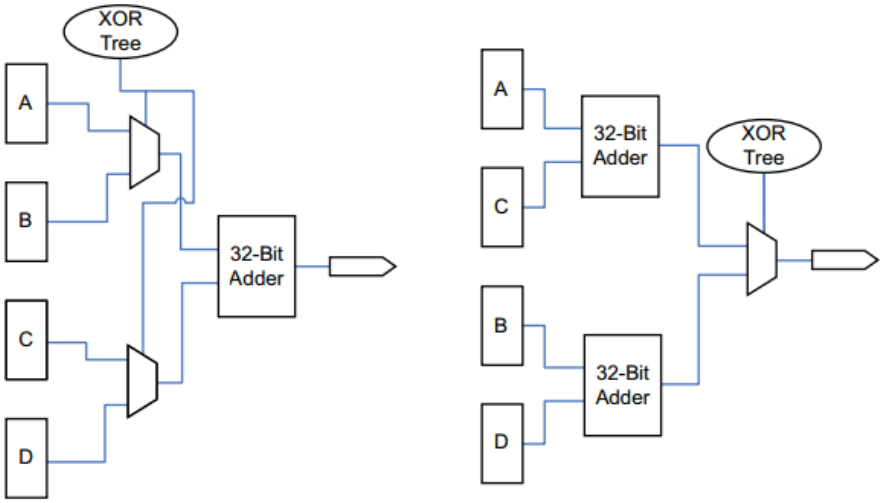
Another technique for power reduction is gating clocks when the logic does not require them. Even though you can build clock-gating logic, this approach can generate clock glitches in FPGAs using ALMs or LEs.

2.6.9. 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 3. Glitch Prone Design



Related Information

Pipelining and Retiming on page 43

2.7. Power Optimization Revision History

The following revision history applies to this chapter:

Table 12. Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Initial release in Intel Quartus Prime Standard Edition User Guide. • 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: *DDR Memory Controller Settings*, *DSP Implementation*, *Reducing High-Speed Tile (HST) Usage*, *Unused Transceiver Channels*, *Periphery Power reduction XCVR Settings*
2018.06.11	18.0.0	• Moved general information about the Design Space Explorer (DSE II) to the *Design Optimization Overview* chapter, 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 Overview* chapter, left a section about using DSE II for Power-Driven Optimization.
2016.10.31	16.1.0	• Removed statement of support for gate-level timing simulation.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel 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.

continued...

Document Version	Intel Quartus Prime Version	Changes
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.
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

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Design Constraints

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Constraining Designs 3
1.1. Specifying Design Constraints Designs in the GUI 3
1.1.1. Global Constraints and Assignments 4
1.1.2. Node, Entity, and Instance-Level Constraints 4
1.1.3. Probing Between Components of the Intel Quartus Prime GUI 7
1.1.4. Specifying Timing Constraints in the GUI 7
1.2. Constraining Designs with Tcl Scripts 9
1.2.1. Create a Project and Apply Constraints 9
1.2.2. Assigning a Pin 10
1.2.3. Generating Intel Quartus Prime Settings Files 10
1.2.4. Synopsys Design Constraint (.sdc) Files 12
1.2.5. Tcl-only Script Flows 13
1.3. A Fully Iterative Scripted Flow 16
1.4. Constraining Designs Revision History 16

2. Managing Device I/O Pins 18
2.1. I/O Planning Overview 19
2.1.1. Basic I/O Planning Flow 19
2.1.2. Integrating PCB Design Tools 20
2.1.3. Intel Device Terms 21
2.2. Assigning I/O Pins 21
2.2.1. Assigning to Exclusive Pin Groups 22
2.2.2. Assigning Slew Rate and Drive Strength 22
2.2.3. Assigning Differential Pins 22
2.2.4. Entering Pin Assignments with Tcl Commands 24
2.2.5. Entering Pin Assignments in HDL Code 24
2.3. Importing and Exporting I/O Pin Assignments 26
2.3.1. Importing and Exporting for PCB Tools 26
2.3.2. Migrating Assignments to Another Target Device 27
2.4. Validating Pin Assignments 28
2.4.1. I/O Assignment Validation Rules 28
2.4.2. Checking I/O Pin Assignments in Real-Time 29
2.4.3. I/O Assignment Analysis 30
2.4.4. Understanding I/O Analysis Reports 34
2.5. Verifying I/O Timing 34
2.5.1. Running Advanced I/O Timing 35
2.5.2. Adjusting I/O Timing and Power with Capacitive Loading 38
2.6. Viewing Routing and Timing Delays 39
2.7. Analyzing Simultaneous Switching Noise 39
2.8. Scripting API 39
2.8.1. Generate Mapped Netlist 39
2.8.2. Reserve Pins 40
2.8.3. Set Location 40
2.8.4. Exclusive I/O Group 40
2.8.5. Slew Rate and Current Strength 40
2.9. Managing Device I/O Pins Revision History 41

A. Intel Quartus Prime Standard Edition User Guides 42

1. Constraining Designs

The design constraints, assignments, and logic options that you specify influence how the Intel® 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 Intel Quartus Prime software preserves the constraints that you specify in the GUI in the following files:

- Intel 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.

Related Information

- Intel Quartus Prime Standard Edition Settings File Reference Manual
  For information about all settings and constraints in the Intel Quartus Prime software.
- Tcl Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*
- Command Line Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*

1.1. Specifying Design Constraints Designs in the GUI

Intel 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 Intel 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

1.1.1. Global Constraints and Assignments

Global constraints and project settings affect the entire Intel 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. Intel Quartus Prime software stores global constraints in `.qsf` files, one for each project revision.

Table 1. Intel Quartus Prime Tools to Set Global Constraints

Assignment Type	Example	New Project Wizard	Device Dialog Box	Settings Dialog Box	Options Dialog Box
Project-wide	Project files	X		X	
Synthesis	• Device Family • Top-level Entity	X	X	X	
Fitter	• Device • Fitter Effort • IO Standard		X	X	
Simulation	Vector input source			X	
Third-party Tools	External Logic Analyzer				X
IP Settings	Maximum Platform Designer (Standard) Memory Usage				X

Related Information

Managing Project Settings
In *Intel Quartus Prime Standard Edition Handbook Volume 1*

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.

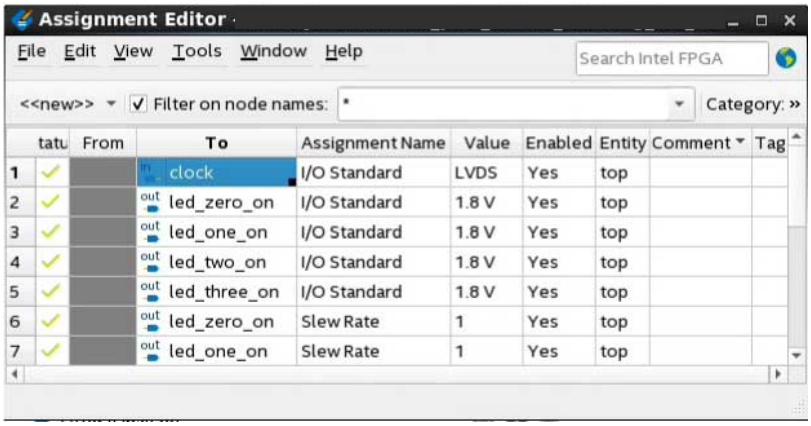
Table 2. Intel Quartus Prime Standard Edition Tools to Set Node, Entity and Instance Level Constraints

Assignment Type	Example	Assignment Editor	Chip Planner	Pin Planner
Pin	Project files	X		X
Location	• Device Family • Top-level Entity	X	X	
Routing	• Device • Fitter Effort • IO Standard	X	X	
Simulation	Vector input source	X	X	X

1.1.2.1. Specify Instance-Specific Constraints in Assignment Editor

Intel 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. Intel Quartus Prime Assignment Editor



Use the Assignment Editor to:

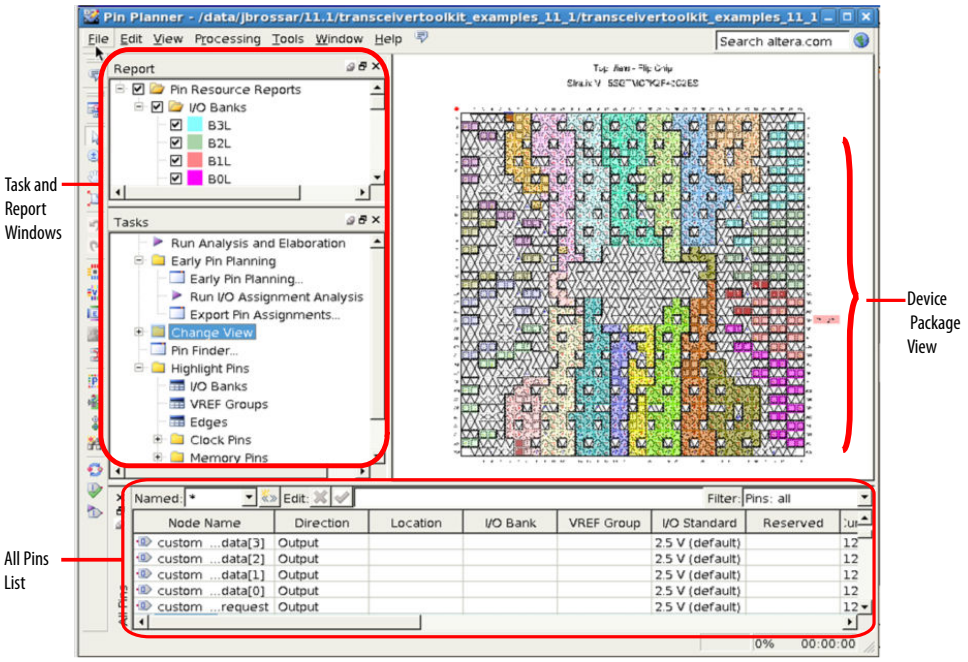
- Add, edit, or delete assignments for selected nodes
- Display information about specific assignments
- Enable or disable individual assignments
- Add comments to an assignment

Additionally, you can export assignments to a Comma-Separated Value File (`.csv`).

1.1.2.2. Specify I/O Constraints in Pin Planner

Intel 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 2. Pin Planner GUI



Related Information

Managing Device I/O Pins on page 18

1.1.2.3. 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 (Standard) regions in the **Logic Lock (Standard) Regions Window**.

Related Information

Design Floorplan Analysis in the Chip Planner
In *Intel Quartus Prime Standard Edition User Guide: Design Optimization*

1.1.2.4. 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 (Standard) Regions with the Design Partition Planner and the Chip Planner
In *Intel Quartus Prime Standard Edition User Guide: Design Optimization*

1.1.3. Probing Between Components of the Intel Quartus Prime GUI

Intel Quartus Prime software allows you to locate nodes and instances across windows and source files.

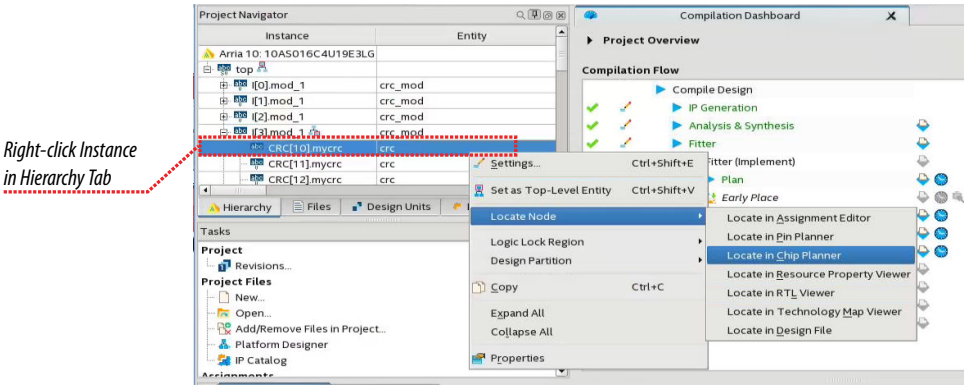
When you are in the Project Navigator, Assignment Editor, Chip Planner, or Pin Planner, and want to display a given resource in other Intel Quartus Prime tool:

1. Right-click the resource you want to display.
2. Click **Locate Node**, and then click one of the menu options.

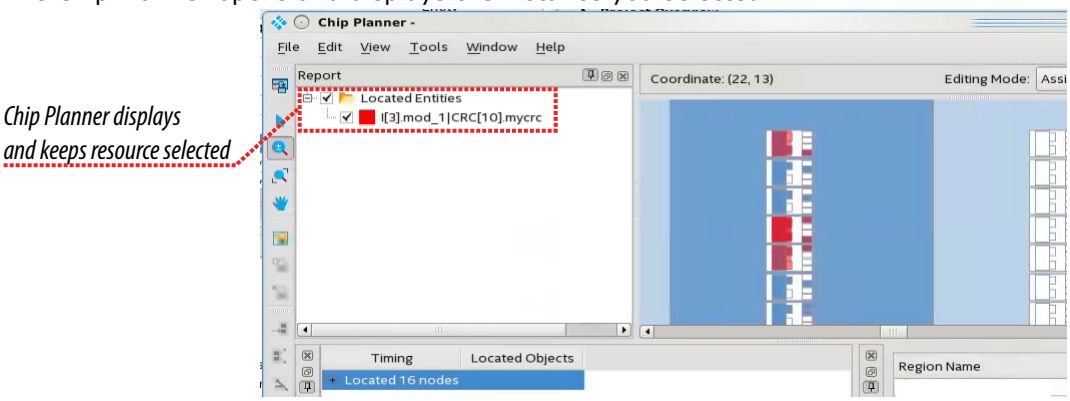
The corresponding window opens—or appears in the foreground if it is already open—and shows the element you clicked.

Example 1. Locate a Resource Selected in the Project Navigator

In the **Entity** list of the **Hierarchy** tab, right-click one object, and click **Locate ➤ Locate in Chip Planner**.



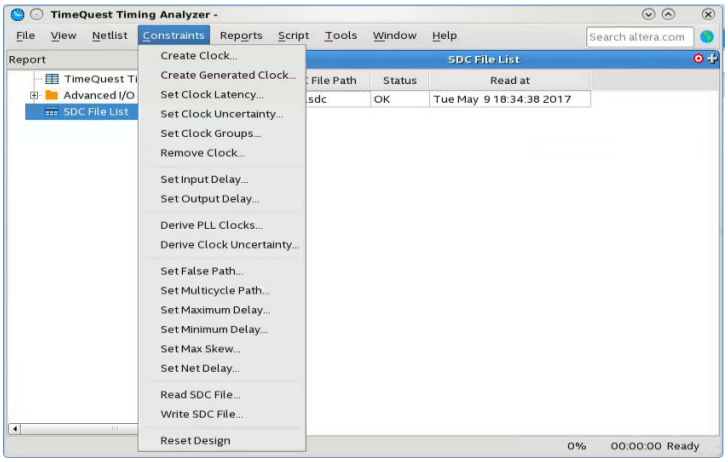
The Chip Planner opens and displays the instance you selected.



1.1.4. Specifying Timing Constraints in the GUI

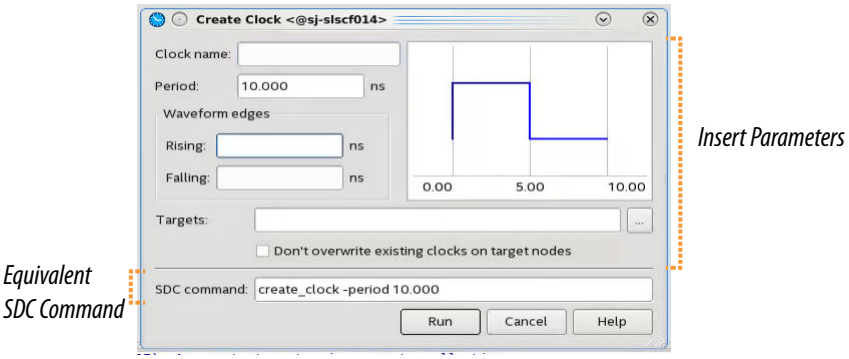
You can specify timing constraints in the Timing Analyzer GUI. Click the Constraints menu in the Timing Analyzer to specify timing constraints that you can apply to your project.

Figure 3. 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.

Related Information

Using the Timing Analyzer
In *Intel Quartus Prime Standard 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 Intel 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 <Intel Quartus Prime *installation directory*>`/qdesigns/fir_filter/` directory.

```
project_new filtref -overwrite
# Assign family, device, and top-level file
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C12F256C6
set_global_assignment -name BDF_FILE filtref.bdf
# 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_map 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 Intel Quartus Prime Settings Files

Intel 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 Intel Quartus Prime software, click **Assignments ➤ Export Assignments**.

To organize the .qsf in a human readable form, **Project ➤ Organize Intel 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 ORIGINAL_QUARTUS_VERSION 9.1
set_global_assignment -name PROJECT_CREATION_TIME_DATE "10:37:10  MAY 7, 2009"
set_global_assignment -name LAST_QUARTUS_VERSION "17.0.0 Standard Edition"
set_global_assignment -name VERILOG_FILE mult.v
set_global_assignment -name VERILOG_FILE accum.v
set_global_assignment -name BDF_FILE filtref.bdf
set_global_assignment -name VERILOG_FILE hvalues.v
set_global_assignment -name VERILOG_FILE taps.v
set_global_assignment -name VERILOG_FILE state_m.v
set_global_assignment -name VERILOG_FILE acc.v
set_global_assignment -name SMART_RECOMPILE ON
set_global_assignment -name VECTOR_WAVEFORM_FILE fir.vwf

# Pin & Location Assignments
# ==========================
set_location_assignment PIN_F13 -to reset
set_location_assignment PIN_G10 -to d[2]
set_location_assignment PIN_F12 -to clk
set_location_assignment PIN_A10 -to clkx2
set_location_assignment PIN_G9 -to d[1]
set_location_assignment PIN_C12 -to d[7]
set_location_assignment PIN_F10 -to follow
set_location_assignment PIN_F9 -to yvalid
set_location_assignment PIN_E13 -to yn_out[2]
set_location_assignment PIN_E10 -to yn_out[3]
set_location_assignment PIN_C11 -to d[4]
set_location_assignment PIN_F11 -to d[0]
set_location_assignment PIN_C13 -to d[6]
set_location_assignment PIN_C8 -to yn_out[6]
```

```
set_location_assignment PIN_B13 -to d[5]
set_location_assignment PIN_B11 -to d[3]
set_location_assignment PIN_B10 -to yn_out[5]
set_location_assignment PIN_B8 -to yn_out[0]
set_location_assignment PIN_A13 -to yn_out[7]
set_location_assignment PIN_A11 -to yn_out[4]
set_location_assignment PIN_A12 -to yn_out[1]
set_location_assignment PIN_A9 -to newt

# Classic Timing Assignments
# ==========================
set_global_assignment -name FMAX_REQUIREMENT "85 MHz"

# Analysis & Synthesis Assignments
# ================================
set_global_assignment -name FAMILY "Cyclone IV GX"
set_global_assignment -name TOP_LEVEL_ENTITY filtref
set_global_assignment -name DEVICE_FILTER_PACKAGE FBGA
set_global_assignment -name DEVICE_FILTER_PIN_COUNT 256
set_global_assignment -name DEVICE_FILTER_SPEED_GRADE 6
set_global_assignment -name CYCLONE_OPTIMIZATION_TECHNIQUE SPEED
set_global_assignment -name MUX_RESTRUCTURE OFF

# Fitter Assignments
# ==================
set_global_assignment -name DEVICE EP4CGX15BF14C6
set_global_assignment -name FITTER_EFFORT "STANDARD FIT"
set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_RETIMING ON
set_global_assignment -name PHYSICAL_SYNTHESIS_EFFORT EXTRA
set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "2.5 V"

# Simulator Assignments
# =====================
set_global_assignment -name VECTOR_INPUT_SOURCE fir.vwf

# start CLOCK(clockb)
# -------------------
# Classic Timing Assignments
    # ==========================
    set_global_assignment -name BASED_ON_CLOCK_SETTINGS clocka -section_id clockb
    set_global_assignment -name DIVIDE_BASE_CLOCK_PERIOD_BY 2 -section_id clockb
    set_global_assignment -name OFFSET_FROM_BASE_CLOCK "500 ps" -section_id
clockb

# end CLOCK(clockb)
# -----------------

# start CLOCK(clocka)
# -------------------
# Classic Timing Assignments
    # ==========================
    set_global_assignment -name FMAX_REQUIREMENT "100 MHz" -section_id clocka

# end CLOCK(clocka)
# -----------------

# ---------------------
# start ENTITY(filtref)
# Classic Timing Assignments
    # ==========================
    set_instance_assignment -name CLOCK_SETTINGS clocka -to clk
    set_instance_assignment -name CLOCK_SETTINGS clockb -to clkx2
    set_instance_assignment -name MULTICYCLE 2 -from clk -to clkx2
# Fitter Assignments
    # ==================
    set_instance_assignment -name SLEW_RATE 2 -to yvalid
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[0]
    set_instance_assignment -name SLEW_RATE 2 -to follow
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[7]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[6]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[5]
```

```
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[4]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[3]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[2]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[1]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
follow
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[7]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[6]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[5]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[4]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[3]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[2]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[1]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[0]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yvalid
# start DESIGN_PARTITION(Top)
    # ---------------------------
# Incremental Compilation Assignments
# ===================================
set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
PLACEMENT_AND_ROUTING -section_id Top
set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top
# end DESIGN_PARTITION(Top)
    # -------------------------

# end ENTITY(filtref)
# -------------------
set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -
section_id Top
```

Related Information

Intel Quartus Prime Standard Edition Settings File Reference Manual
For information about all settings and constraints in the Intel Quartus Prime software.

1.2.4. Synopsys Design Constraint (.sdc) Files

Intel 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 17.0.0 Build 595 04/25/2017 SJ Standard Edition"
## DATE    "Wed May 10 14:03:25 2017"
##
## DEVICE  "EP4CGX15BF14C6"
##
#**************************************************************
# Time Information
#**************************************************************
set_time_format -unit ns -decimal_places 3
```

```
#**************************************************************
# Create Clock
#**************************************************************
create_clock -name {clk} -period 4.000 -waveform { 0.000 2.000 } [get_ports
{clk}]
create_clock -name {clkx2} -period 4.000 -waveform { 0.000 2.000 } [get_ports
{clkx2}]
#**************************************************************
# Set Clock Uncertainty
#**************************************************************
set_clock_uncertainty -rise_from [get_clocks {clkx2}] -rise_to [get_clocks
{clkx2}]  0.020
set_clock_uncertainty -rise_from [get_clocks {clkx2}] -fall_to [get_clocks
{clkx2}]  0.020
set_clock_uncertainty -fall_from [get_clocks {clkx2}] -rise_to [get_clocks
{clkx2}]  0.020
set_clock_uncertainty -fall_from [get_clocks {clkx2}] -fall_to [get_clocks
{clkx2}]  0.020
set_clock_uncertainty -rise_from [get_clocks {clk}] -rise_to [get_clocks
{clkx2}]  0.040
set_clock_uncertainty -rise_from [get_clocks {clk}] -fall_to [get_clocks
{clkx2}]  0.040
set_clock_uncertainty -rise_from [get_clocks {clk}] -rise_to [get_clocks {clk}]
0.020
set_clock_uncertainty -rise_from [get_clocks {clk}] -fall_to [get_clocks {clk}]
0.020
set_clock_uncertainty -fall_from [get_clocks {clk}] -rise_to [get_clocks
{clkx2}]  0.040
set_clock_uncertainty -fall_from [get_clocks {clk}] -fall_to [get_clocks
{clkx2}]  0.040
set_clock_uncertainty -fall_from [get_clocks {clk}] -rise_to [get_clocks {clk}]
0.020
set_clock_uncertainty -fall_from [get_clocks {clk}] -fall_to [get_clocks {clk}]
0.020
#**************************************************************
# Set False Path
#**************************************************************
set_false_path -from [get_clocks {clk clkx2}] -through [get_pins -
compatibility_mode *]  -to [get_clocks {clk clkx2}]
```

Related Information

Constraining and Analyzing with Tcl Commands
In *Intel Quartus Prime Standard Edition User Guide: Timing Analyzer*

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. fir_filter_generated.tcl Tcl file

```
# Quartus Prime: Generate Tcl File for Project
# File: fir_filter_generated.tcl
# Generated on: Tue May  9 18:41:24 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) "fir_filter"]} {
      puts "Project fir_filter is not open"
      set make_assignments 0
      }
} else {
    # Only open if not already open
    if {[project_exists fir_filter]} {
      project_open -revision filtref fir_filter
    } else {
      project_new -revision filtref fir_filter
    }
    set need_to_close_project 1
}

# Make assignments
if {$make_assignments} {
    set_global_assignment -name ORIGINAL_QUARTUS_VERSION 9.1
    set_global_assignment -name PROJECT_CREATION_TIME_DATE "10:37:10  MAY 7,
2009"
    set_global_assignment -name LAST_QUARTUS_VERSION "17.0.0 Standard Edition"
    set_global_assignment -name VERILOG_FILE mult.v
    set_global_assignment -name VERILOG_FILE accum.v
    set_global_assignment -name BDF_FILE filtref.bdf
    set_global_assignment -name VERILOG_FILE hvalues.v
    set_global_assignment -name VERILOG_FILE taps.v
    set_global_assignment -name VERILOG_FILE state_m.v
    set_global_assignment -name VERILOG_FILE acc.v
    set_global_assignment -name SMART_RECOMPILE ON
    set_global_assignment -name VECTOR_WAVEFORM_FILE fir.vwf
    set_global_assignment -name FMAX_REQUIREMENT "85 MHz"
    set_global_assignment -name FAMILY "Cyclone IV GX"
    set_global_assignment -name DEVICE_FILTER_PACKAGE FBGA
    set_global_assignment -name DEVICE_FILTER_PIN_COUNT 256
    set_global_assignment -name DEVICE_FILTER_SPEED_GRADE 6
    set_global_assignment -name CYCLONE_OPTIMIZATION_TECHNIQUE SPEED
    set_global_assignment -name MUX_RESTRUCTURE OFF
    set_global_assignment -name DEVICE EP4CGX15BF14C6
    set_global_assignment -name FITTER_EFFORT "STANDARD FIT"
    set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_RETIMING ON
    set_global_assignment -name PHYSICAL_SYNTHESIS_EFFORT EXTRA
    set_global_assignment -name STRATIX_DEVICE_IO_STANDARD "2.5 V"
    set_global_assignment -name VECTOR_INPUT_SOURCE fir.vwf
    set_global_assignment -name BASED_ON_CLOCK_SETTINGS clocka -section_id clockb
    set_global_assignment -name DIVIDE_BASE_CLOCK_PERIOD_BY 2 -section_id clockb
    set_global_assignment -name OFFSET_FROM_BASE_CLOCK "500 ps" -section_id
clockb
    set_global_assignment -name FMAX_REQUIREMENT "100 MHz" -section_id clocka
    set_global_assignment -name PARTITION_NETLIST_TYPE SOURCE -section_id Top
    set_global_assignment -name PARTITION_FITTER_PRESERVATION_LEVEL
PLACEMENT_AND_ROUTING -section_id Top
    set_global_assignment -name PARTITION_COLOR 16764057 -section_id Top
    set_location_assignment PIN_F13 -to reset
    set_location_assignment PIN_G10 -to d[2]
    set_location_assignment PIN_F12 -to clk
    set_location_assignment PIN_A10 -to clkx2
    set_location_assignment PIN_G9 -to d[1]
    set_location_assignment PIN_C12 -to d[7]
    set_location_assignment PIN_F10 -to follow
    set_location_assignment PIN_F9 -to yvalid
    set_location_assignment PIN_E13 -to yn_out[2]
    set_location_assignment PIN_E10 -to yn_out[3]
    set_location_assignment PIN_C11 -to d[4]
    set_location_assignment PIN_F11 -to d[0]
    set_location_assignment PIN_C13 -to d[6]
    set_location_assignment PIN_C8 -to yn_out[6]
    set_location_assignment PIN_B13 -to d[5]
```

```
    set_location_assignment PIN_B11 -to d[3]
    set_location_assignment PIN_B10 -to yn_out[5]
    set_location_assignment PIN_B8 -to yn_out[0]
    set_location_assignment PIN_A13 -to yn_out[7]
    set_location_assignment PIN_A11 -to yn_out[4]
    set_location_assignment PIN_A12 -to yn_out[1]
    set_location_assignment PIN_A9 -to newt
    set_instance_assignment -name CLOCK_SETTINGS clocka -to clk
    set_instance_assignment -name CLOCK_SETTINGS clockb -to clkx2
    set_instance_assignment -name MULTICYCLE 2 -from clk -to clkx2
    set_instance_assignment -name SLEW_RATE 2 -to yvalid
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[0]
    set_instance_assignment -name SLEW_RATE 2 -to follow
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[7]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[6]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[5]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[4]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[3]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[2]
    set_instance_assignment -name SLEW_RATE 2 -to yn_out[1]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
follow
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[7]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[6]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[5]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[4]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[3]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[2]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[1]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yn_out[0]
    set_instance_assignment -name CURRENT_STRENGTH_NEW "MINIMUM CURRENT" -to
yvalid
    set_instance_assignment -name PARTITION_HIERARCHY root_partition -to | -
section_id Top

    # 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

1.2.5.1. Tcl-only Timing Analysis

To avoid using a separated file to keep your timing constraints, copy and paste the `.sdc` file into your executable timing analysis script.

1.3. A Fully Iterative Scripted Flow

The **::quartus::flow** Tcl package in the Intel Quartus Prime Tcl API allows you to modify design constraints and recompile in an iterative flow.

Related Information

- ::quartus::flow
  In *Intel Quartus Prime Help*
- Command Line Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*

1.4. Constraining Designs Revision History

Document Version	Intel Quartus Prime Version	Changes
2019.01.04	18.1.0	• Clarified default location of .sdc and .qsf files in "Constraining Designs" topic. • Added two new "Assigning a Pin" and "Creating a Project and Applying Constraints" topics showing Tcl examples.
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2017.11.06	17.1.0	• 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.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	Formatting updates.

continued...

Document Version	Intel Quartus Prime Version	Changes
November 2012	12.1.0	Update Pin Planner description for task and report windows.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Template update.
July 2010	10.0.0	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	9.1.0	• Added two notes. • Minor text edits.
March 2009	9.0.0	• 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	8.1.0	Changed to 8½" × 11" page size. No change to content.
May 2008	8.0.0	Updated Quartus II software 8.0 revision and date.

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

2. 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 4. Pin Planner GUI

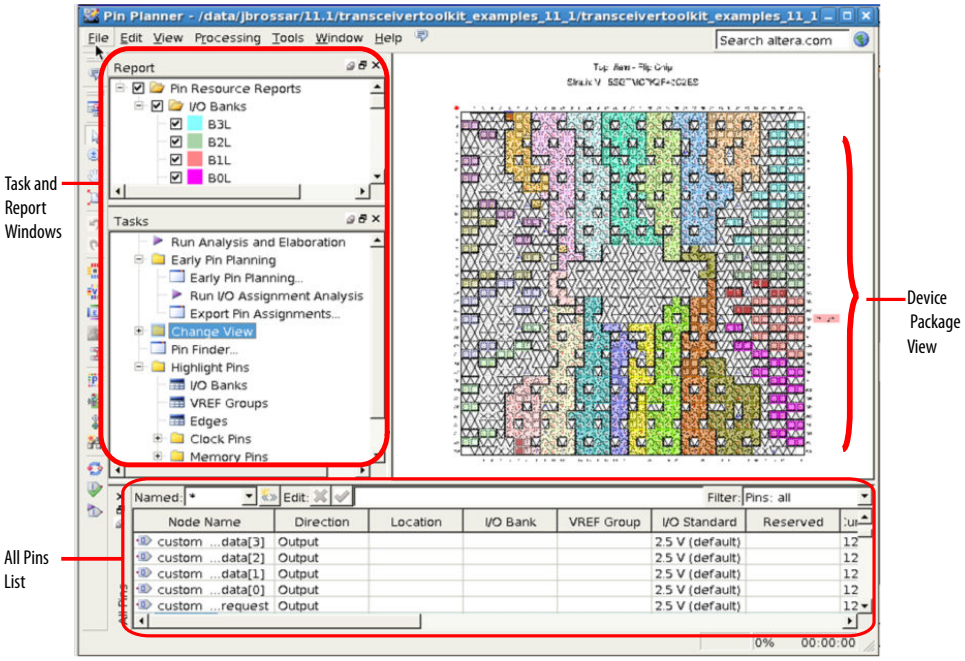


Table 3. Intel Quartus Prime I/O Pin Planning Tools

I/O Planning Task	Click to Access
Edit, validate, or export pin assignments	**Assignments ➤ Pin Planner**
View tailored pin planning advice	**Tools ➤ Advisors ➤ Pin Advisor**
Validate pin assignments against design rules	**Processing ➤ Start ➤ Start I/O Assignment Analysis**

For more information about special pin assignment features for the Intel Arria® 10 SoC devices, refer to *Instantiating the HPS Component* in the *Intel Arria 10 Hard Processor System Technical Reference Manual*.

Related Information

Instantiating the HPS Component
In *Intel Arria 10 Hard Processor System Technical Reference Manual*

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

2.1. 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.

Intel 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.

2.1.1. 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. To setup a top-level HDL wrapper file that defines early port and interface information for your design, click **Early Pin Planning** in the **Tasks** pane.
    a. Click **Import IP Core** to import any defined IP core, and then assign signals to the interface IP nodes.
    b. Click **Set Up Top-Level File** and assign user nodes to device pins. User nodes become virtual pins in the top-level file and are not assigned to device pins.
    c. Click **Generate Top-Level File**. Use top-level file to validate I/O assignments.
4. Assign I/O properties to match your device and PCB characteristics, including assigning logic, I/O standards, output loading, slew rate, and current strength.
5. Click **Run I/O Assignment Analysis** in the **Tasks** pane to validate assignments and generate a synthesized design netlist. Correct any problems reported.
6. Click **Processing ➤ Start Compilation**. During compilation, the Intel Quartus Prime software runs I/O assignment analysis.

2.1.2. Integrating PCB Design Tools

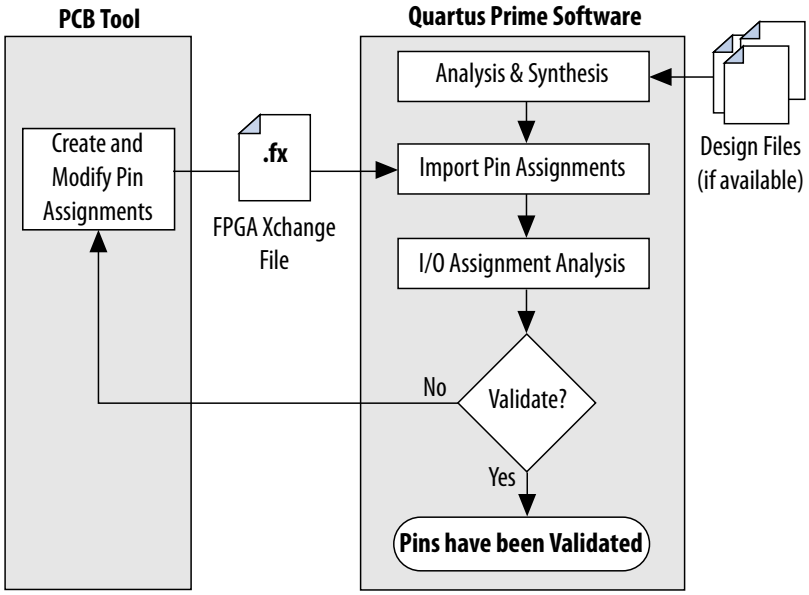
You can integrate PCB design tools into your work flow to map pin assignments to symbols in your system circuit schematics and board layout.

The Intel Quartus Prime software integrates with board layout tools by allowing import and export of pin assignment information in Intel Quartus Prime Settings Files (`.qsf`), Pin-Out File (`.pin`), and FPGA Xchange-Format File (`.fx`) files.

Table 4. 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	Mentor Graphics* I/O Designer Cadence Allegro
Define I/O assignments in your PCB tool, and then import the assignments into the Pin Planner for validation	Mentor Graphics I/O Designer Cadence Allegro

Figure 5. PCB Tool Integration

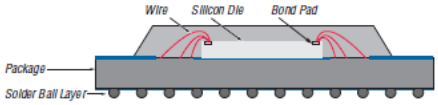
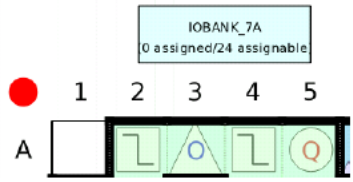
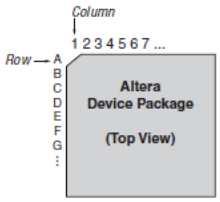
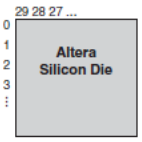
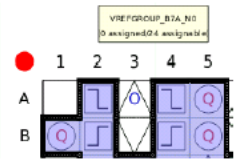


Related Information

Cadence PCB Design Tools Support
In *Intel Quartus Prime Standard Edition User Guide: PCB Design Tools*

2.1.3. Intel Device Terms

The following terms describe Intel 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. Click **View > Show > Show VREF Groups** in Pin Planner..	

2.2. 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 Intel 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.

2.2.1. 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.

2.2.2. 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.

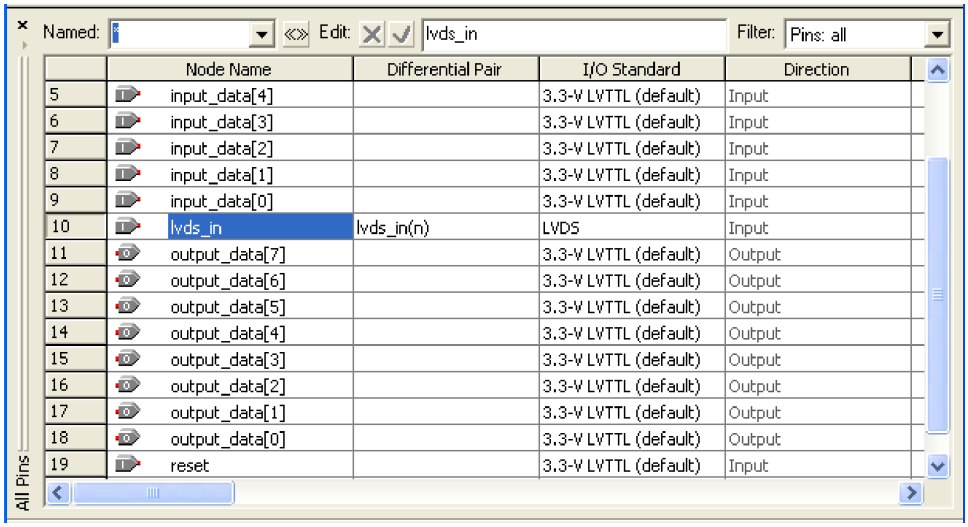
2.2.3. Assigning Differential Pins

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 Intel 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.

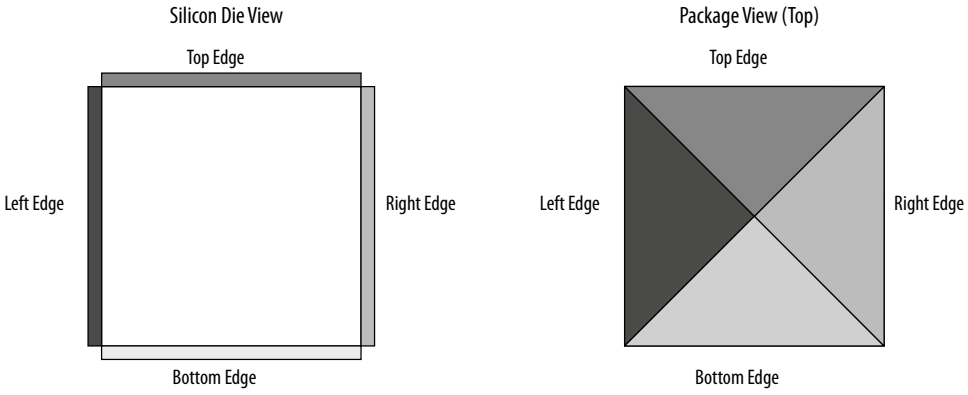
Figure 6. Creating a Differential Pin Pair in the Pin Planner



	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 7. Die View and Package View of the Four Edges on an Intel Device



2.2.3.1. 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.

2.2.4. 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

Tcl Scripting
In *Intel Quartus Prime Standard Edition User Guide: Scripting*

2.2.5. 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

2.2.5.1. Using Synthesis Attributes

The Intel Quartus Prime software translates synthesis attributes into standard assignments during compilation. These assignments appear in the Pin Planner. Intel Quartus Prime synthesis supports the `chip_pin`, `useioff`, and `altera_attribute` synthesis attributes.

If you modify or delete these assignments in the Pin Planner and then recompile your design, the Pin Planner changes override the synthesis attributes.

Use the `altera_attribute` synthesis attribute to create other pin-related assignments in your HDL code. The `altera_attribute` attribute supports all types of instance assignments. The following examples use the `altera_attribute` attribute to embed **Fast Input Register** logic option assignments and I/O standard assignments in both a Verilog HDL and a VHDL design file.

Example 8. altera_attribute Synthesis Attribute in Verilog HDL

```
input my_pin1 /* synthesis altera_attribute = "-name FAST_INPUT_REGISTER ON; -
name IO_STANDARD \"2.5 V\" " */ ;
```

Example 9. altera_attribute Synthesis Attribute in VHDL

```
entity my_entity is
        port(
                my_pin1: in std_logic
        );
end my_entity;
architecture rtl of my_entity is
begin

attribute altera_attribute : string;
attribute altera_attribute of my_pin1: signal is "-name FAST_INPUT_REGISTER ON;
-- The architecture body
end rtl;
```

Use the `chip_pin` and `useioff` synthesis attributes to create pin location assignments and to assign **Fast Input Register**, **Fast Output Register**, and **Fast Output Enable Register** logic options. The following examples use the `chip_pin` and `useioff` attributes to embed location and **Fast Input Register** logic option assignments in Verilog HDL and VHDL design files.

Example 10. useioff and chip_pin Synthesis Attributes in VHDL

```
entity my_entity is
        port(
                    my_pin1: in std_logic
        );
end my_entity;

architecture rtl of my_entity is
attribute useioff : boolean;
attribute useioff of my_pin1 : signal is true;
attribute chip_pin : string;
attribute chip_pin of my_pin1 : signal is "C1";
begin -- The architecture body
end rtl;
```

Example 11. chip_pin Synthesis Attribute in Verilog HDL

```
input my_pin1 /* synthesis chip_pin = "C1" useioff = 1 */;
```

2.2.5.2. 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**).

Related Information

Designing with Low Level Primitives User Guide

2.3. Importing and Exporting I/O Pin Assignments

The Intel 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 5. 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 Intel Quartus Prime project with common constraints	• From Intel Quartus Prime project for optimization in a PCB design tool • From Intel Quartus Prime project for spreadsheet analysis or use in scripting assignments • From Intel Quartus Prime project for import into another Intel Quartus Prime project with similar constraints
Command	**Assignments ➤ Import Assignments**	**Assignments ➤ Export Assignments**
File formats	`.qsf`, `.esf`, `.acf`, `.csv`, `.txt`,`.sdc`	`.pin`,`.fx`,`.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

2.3.1. 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.

Mentor Graphics I/O Designer requires you to generate and import both an **.fx** and a **.pin** file to transfer assignments. However, the Intel Quartus Prime software requires only the **.fx** to import pin assignments from I/O Designer.

Table 6. 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

- Pin-Out Files for Intel Devices
- PCB Design Tools Support
  In *Intel Quartus Prime Standard Edition User Guide: PCB Design Tools*

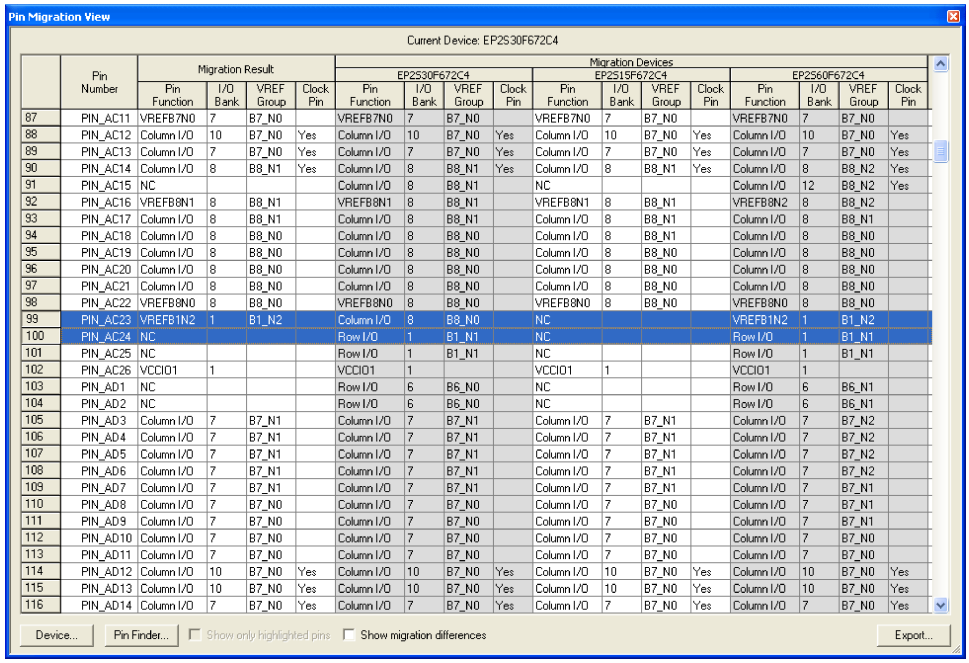
2.3.2. Migrating Assignments to Another Target Device

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 Intel 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 8. Device Migration Compatibility (AC24 does not exist in migration device)



	Pin Number	Migration Result				EP2S30F672C4				EP2S15F672C4				EP2S60F672C4			
		Pin Function	I/O Bank	VREF Group	Clock Pin	Pin Function	I/O Bank	VREF Group	Clock Pin	Pin Function	I/O Bank	VREF Group	Clock Pin	Pin Function	I/O Bank	VREF Group	Clock Pin
87	PIN_AC11	VREFB7N0	7	B7_N0		VREFB7N0	7	B7_N0		VREFB7N0	7	B7_N0		VREFB7N0	7	B7_N0	
88	PIN_AC12	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes
89	PIN_AC13	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes
90	PIN_AC14	Column I/O	8	B8_N1	Yes	Column I/O	8	B8_N1	Yes	Column I/O	8	B8_N1	Yes	Column I/O	8	B8_N2	Yes
91	PIN_AC15	NC				Column I/O	8	B8_N1		NC				Column I/O	12	B8_N2	Yes
92	PIN_AC16	VREFB8N1	8	B8_N1		VREFB8N1	8	B8_N1		VREFB8N1	8	B8_N1		VREFB8N2	8	B8_N2	
93	PIN_AC17	Column I/O	8	B8_N1		Column I/O	8	B8_N1		Column I/O	8	B8_N1		Column I/O	8	B8_N1	
94	PIN_AC18	Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N1		Column I/O	8	B8_N0	
95	PIN_AC19	Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0	
96	PIN_AC20	Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0	
97	PIN_AC21	Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0		Column I/O	8	B8_N0	
98	PIN_AC22	VREFB8N0	8	B8_N0		VREFB8N0	8	B8_N0		VREFB8N0	8	B8_N0		VREFB8N0	8	B8_N0	
99	PIN_AC23	VREFB1N2	1	B1_N2		Column I/O	8	B8_N0		NC				VREFB1N2	1	B1_N2	
100	PIN_AC24	NC				Row I/O	1	B1_N1		NC				Row I/O	1	B1_N1	
101	PIN_AC25	NC				Row I/O	1	B1_N1		NC				Row I/O	1	B1_N1	
102	PIN_AC26	VCCIO1	1			VCCIO1	1			VCCIO1	1			VCCIO1	1		
103	PIN_AD1	NC				Row I/O	6	B6_N0		NC				Row I/O	6	B6_N1	
104	PIN_AD2	NC				Row I/O	6	B6_N0		NC				Row I/O	6	B6_N1	
105	PIN_AD3	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N2	
106	PIN_AD4	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N2	
107	PIN_AD5	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N2	
108	PIN_AD6	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N2	
109	PIN_AD7	Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1		Column I/O	7	B7_N1	
110	PIN_AD8	Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N1	
111	PIN_AD9	Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N1	
112	PIN_AD10	Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0	
113	PIN_AD11	Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0		Column I/O	7	B7_N0	
114	PIN_AD12	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes
115	PIN_AD13	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes	Column I/O	10	B7_N0	Yes
116	PIN_AD14	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes	Column I/O	7	B7_N0	Yes

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 Intel 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.

Related Information

AN90: SameFrame PinOut Design for FineLine BGA Packages

2.4. Validating Pin Assignments

The Intel 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 7. I/O Validation Tools

I/O Validation Tool	Description	Click to Run
I/O Assignment Analysis	Verifies I/O assignment legality of synthesized design against full set of I/O rules for the target device	**Processing ➤ Start I/O Assignment Analysis**
Advanced I/O Timing	Fully validates I/O assignments against all I/O and timing checks during compilation	**Processing ➤ Start Compilation**

2.4.1. I/O Assignment Validation Rules

I/O Assignment Analysis validates your assignments against the following rules:

Table 8. 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

continued...

Rule	Description	HDL Required?
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 9. 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

2.4.2. Checking I/O Pin Assignments in Real-Time

Live I/O check validates I/O assignments against basic I/O buffer rules in real time. The Pin Planner immediately reports warnings or errors about assignments as you enter them. The Live I/O Check Status window displays the total number of errors and warnings. Use this analysis to quickly correct basic errors before proceeding. Run full I/O assignment analysis when you are ready to validate pin assignments against the complete set of I/O system rules.

Note: Live I/O check is supported only for Arria II, Cyclone® IV, MAX® II, and Stratix IV device families.

Live I/O check validates against the following basic I/O buffer rules:

- V_{CCIO} and V_{REF} voltage compatibility rules
- Electromigration (current density) rules
- Simultaneous Switching Output (SSO) rules
- I/O property compatibility rules, such as drive strength compatibility, I/O standard compatibility, PCI_IO clamp diode compatibility, and I/O direction compatibility
- Illegal location assignments:
  - An I/O bank or VREF group with no available pins
  - The negative pin of a differential pair if the positive pin of the differential pair is assigned with a node name with a differential I/O standard
  - Pin locations that do not support the I/O standard assigned to the selected node name
  - For HSTL- and SSTL-type I/O standards, VREF groups of a different V_{REF} voltage than the selected node name.

2.4.3. 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.

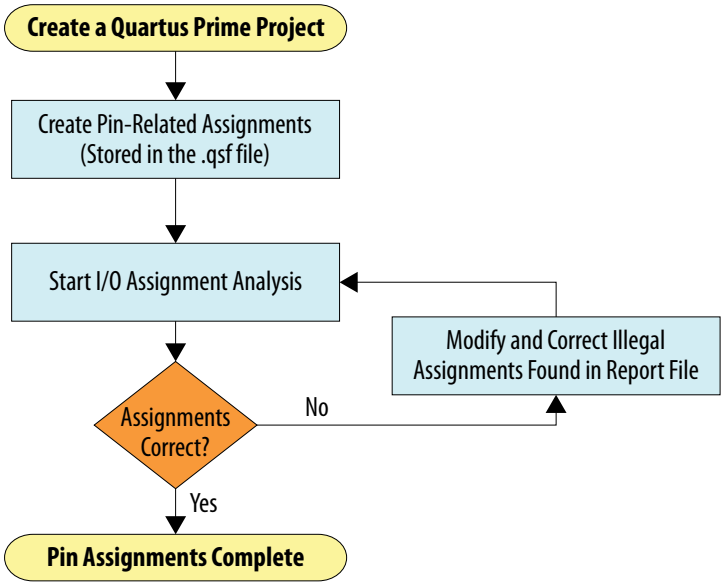
The Fitter assigns pins to accommodate your constraints. For example, if you assign an edge location to a group of LVDS pins, the Fitter assigns pin locations for each LVDS pin in the specified edge location and then performs legality checks. To display the Fitter-placed pins, click **Show Fitter Placements** 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.

2.4.3.1. 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 9. Assigning and Analyzing Pin-Outs without Design Files

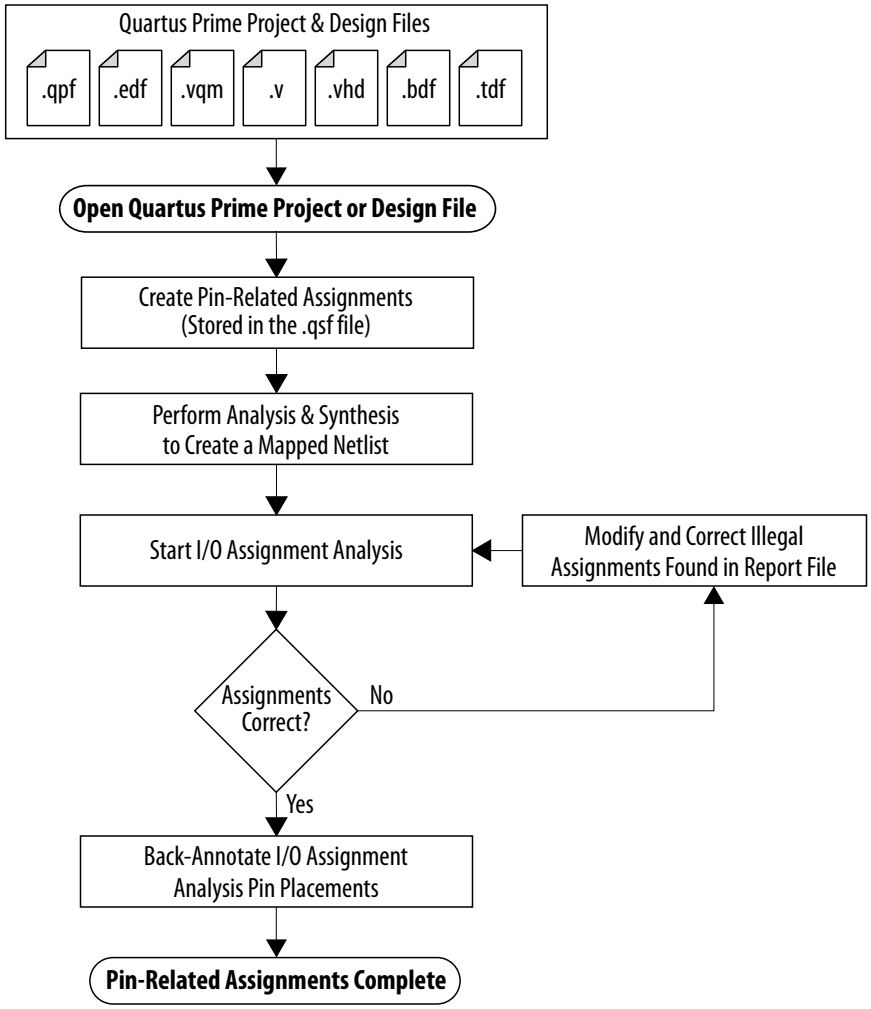


You must reserve all pins you intend to use as I/O pins, so that the Fitter can determine each pin type. After performing I/O assignment analysis, correct any errors reported by the Fitter and rerun I/O assignment analysis until all errors are corrected. A complete I/O assignment analysis requires all design files.

2.4.3.2. 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 10. I/O Assignment Analysis Flow

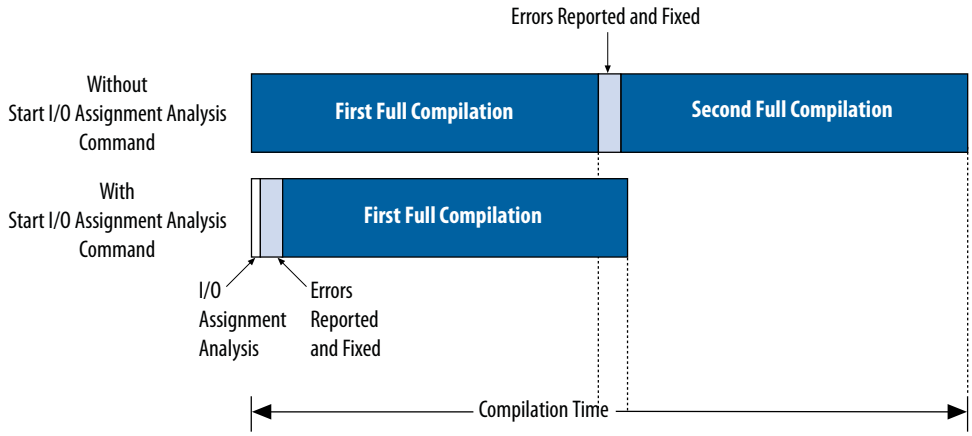


Even if I/O assignment analysis passes on incomplete design files, you may still encounter errors during full compilation. For example, you can assign a clock to a user I/O pin instead of assigning to a dedicated clock pin, or design the clock to drive a PLL that you have not yet instantiated in the 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 11. I/O Assignment Analysis Reduces Compilation Time



2.4.3.3. 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.

Related Information

Using the Timing Analyzer
In *Intel Quartus Prime Standard Edition User Guide: Timing Analyzer*

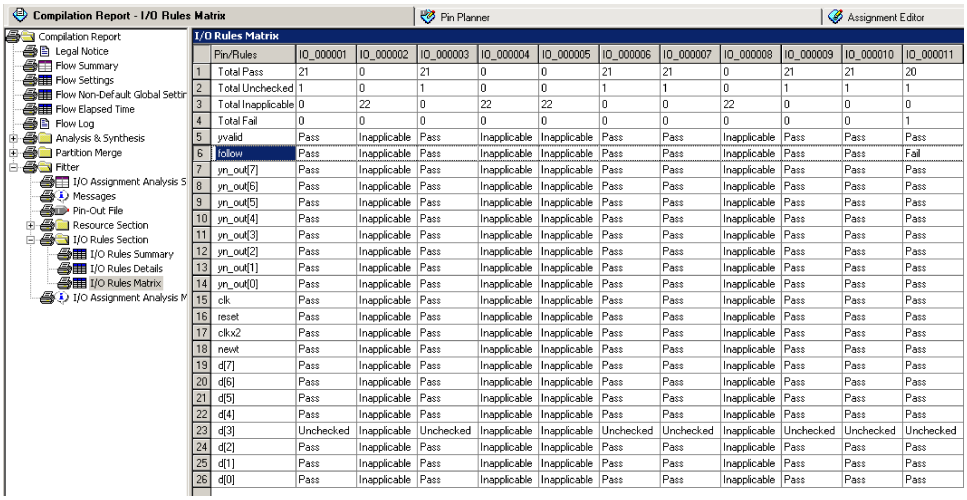
2.4.4. Understanding I/O Analysis Reports

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 12. I/O Rules Matrix



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	clkx2	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

2.5. 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 Intel Quartus Prime software supports three different methods of I/O timing analysis:

Table 10. 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 Intel Quartus Prime software-generated IBIS or HSPICE I/O models for simulation in Mentor Graphics HyperLynx* and Synopsys HSPICE.

Note: Advanced I/O timing analysis is supported only for .28nm and larger device families. For devices that support advanced I/O timing, it is the default method of I/O timing analysis. For all other devices, you must use a default or user-specified capacitive load assignment to determine t_{CO} and power measurements.

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 Altera Signal Integrity Center page of the Altera website.

For information about creating IBIS and HSPICE models with the Intel 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 Analysis with Third-Party Tools
  In *Intel Quartus Prime Standard Edition User Guide: PCB Design Tools*
- Literature and Technical Documentation
- Intel Signal & Power Integrity Center

2.5.1. 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.

2.5.1.1. Board Trace Models

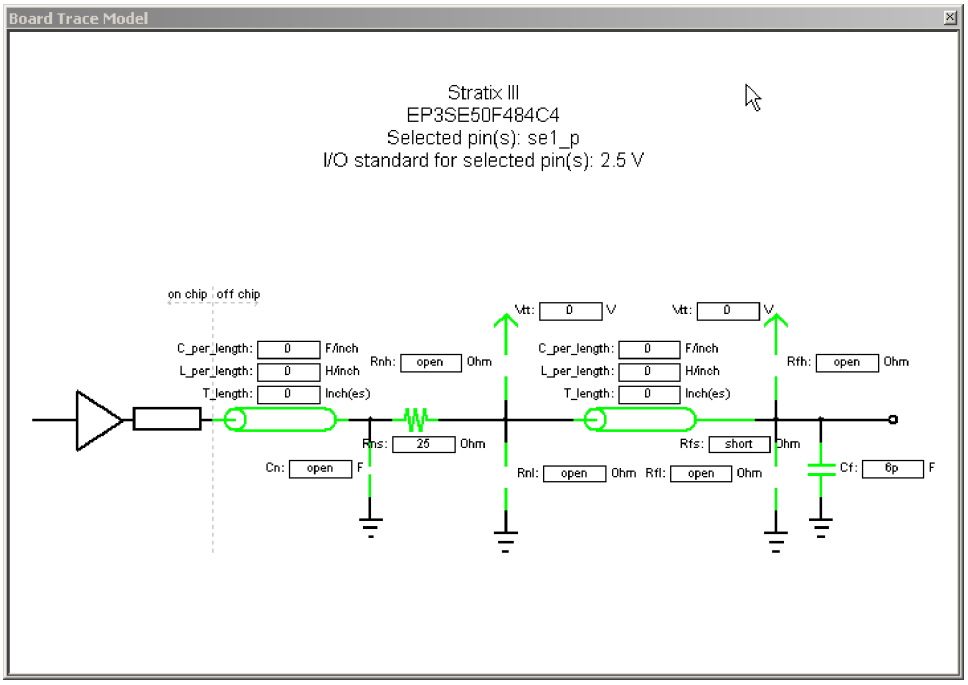
The Intel Quartus Prime software provides board trace model templates for various I/O standards.

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.

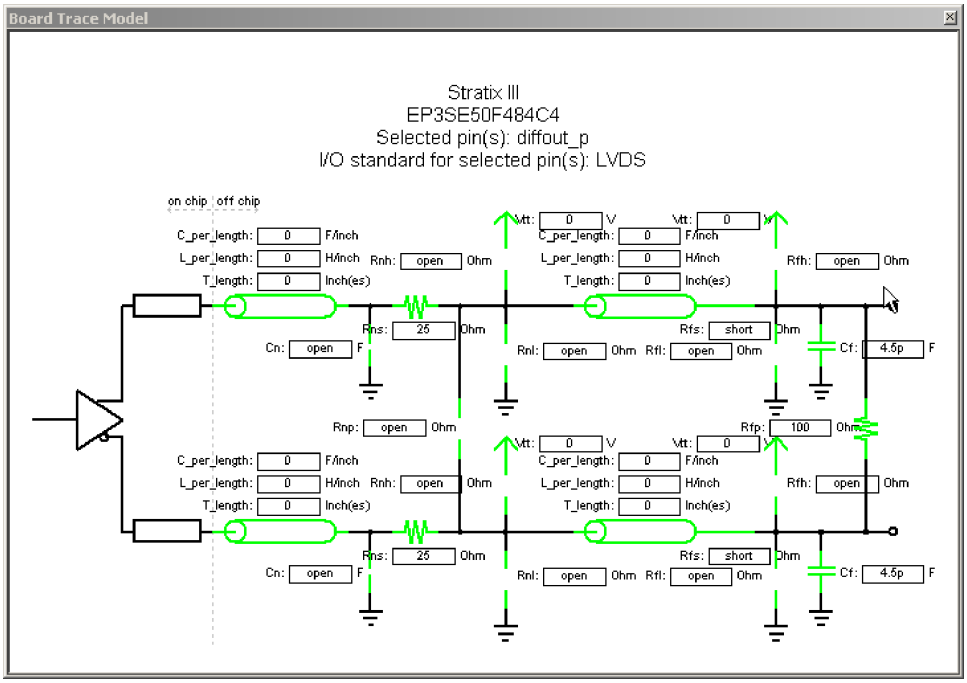
Figure 13. 2.5-V I/O Standard Board Trace Model



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 Intel 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 14. LVDS Differential Board Trace Model



2.5.1.2. 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 12. 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
```

2.5.1.3. 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.

2.5.1.4. 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.

2.5.1.5. Advanced I/O Timing Analysis Reports

The following reports show advanced I/O timing analysis information:

Table 11. 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**.

Related Information

Using the Timing Analyzer
In *Intel Quartus Prime Standard Edition User Guide: Timing Analyzer*

2.5.2. 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 Intel 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.

2.6. 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 (Standard) 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.

2.7. Analyzing Simultaneous Switching Noise

Click **Processing > Start > Start SSN Analyzer** to estimate the voltage noise for each pin in the design. The simultaneous switching noise (SSN) analysis accounts for the pin placement, I/O standard, board trace, output enable group, timing constraint, and PCB characteristics that you specify. The analysis produces a voltage noise estimate for each pin in the design. View the SSN results in the Pin Planner and adjust your I/O assignments to optimize signal integrity.

2.8. Scripting API

The Intel 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

- Tcl Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*
- Command Line Scripting
  In *Intel Quartus Prime Standard Edition User Guide: Scripting*

2.8.1. 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_map <project name>
```

2.8.2. 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.

2.8.3. 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

2.8.4. Exclusive I/O Group

The following Tcl command creates an exclusive I/O group assignment:

```
set_instance_assignment -name "EXCLUSIVE_IO_GROUP" -to pin
```

2.8.5. 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]
```

Related Information

Package Information Datasheet for Mature Altera Devices

2.9. Managing Device I/O Pins Revision History

The following table shows the revision history for this chapter:

Document Version	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	Initial release in Intel Quartus Prime Standard Edition User Guide.
2017.11.06	17.1.0	• 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.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Quartus Prime*.
2014.12.15	14.1.0	• Updated Live I/O check device support to include only limited device families.
2014.08.30	14.0a10.0	• Added link to information about special pin assignment features for Arria 10 SoC devices.
2014.06.30	14.0.0	• Replaced MegaWizard Plug-In Manager information with IP Catalog.
November 2013	13.1.0	• Reorganization and conversion to DITA.
May 2013	13.0.0	• Added information about overriding I/O placement rules.
November 2012	12.1.0	• Updated Pin Planner description for new task and report windows.
June 2012	12.0.0	• Removed survey link.
November 2011	11.1.0	• Minor updates and corrections. • Updated the document template.
December 2010	10.0.1	Template update
July 2010	10.0.0	• 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	9.1.0	• 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	9.0.0	• 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

Related Information

Documentation Archive

For previous versions of the *Intel Quartus Prime Handbook*, search the documentation archives.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

PCB Design Tools

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Simultaneous Switching Noise (SSN) Analysis and Optimizations.....5
1.1. Simultaneous Switching Noise (SSN) Analysis and Optimizations.....5
1.2. Definitions.....5
1.3. Understanding SSN.....6
1.4. SSN Estimation Tools.....8
1.5. SSN Analysis Overview.....9
1.5.1. Performing Early Pin-Out SSN Analysis.....10
1.5.2. Performing Final Pin-Out SSN Analysis.....11
1.6. Design Factors Affecting SSN Results.....11
1.7. Optimizing Your Design for SSN Analysis.....11
1.7.1. Optimizing Pin Placements for Signal Integrity.....12
1.7.2. Specifying Board Trace Model Settings.....13
1.7.3. Defining PCB Layers and PCB Layer Thickness.....14
1.7.4. Specifying Signal Breakout Layers.....16
1.7.5. Creating I/O Assignments.....17
1.7.6. Decreasing Pessimism in SSN Analysis.....17
1.7.7. Excluding Pins as Aggressor Signals.....18
1.8. Performing SSN Analysis and Viewing Results.....18
1.8.1. Understanding the SSN Reports.....18
1.8.2. Viewing SSN Analysis Results in the Pin Planner.....19
1.9. Decreasing Processing Time for SSN Analysis.....20
1.10. Scripting Support.....20
1.10.1. Optimizing Pin Placements for Signal Integrity.....21
1.10.2. Defining PCB Layers and PCB Layer Thickness.....21
1.10.3. Specifying Signal Breakout Layers.....22
1.10.4. Decreasing Pessimism in SSN Analysis.....22
1.10.5. Performing SSN Analysis.....22
1.11. Document Revision History.....23

2. Signal Integrity Analysis with Third-Party Tools.....24
2.1. Signal Integrity Analysis with Third-Party Tools.....24
2.1.1. Signal Integrity Simulations with HSPICE and IBIS Models.....25
2.2. I/O Model Selection: IBIS or HSPICE.....26
2.3. FPGA to Board Signal Integrity Analysis Flow.....26
2.3.1. Create I/O and Board Trace Model Assignments.....29
2.3.2. Output File Generation.....29
2.3.3. Customize the Output Files.....29
2.3.4. Set Up and Run Simulations in Third-Party Tools.....30
2.3.5. Interpret Simulation Results.....30
2.4. Simulation with IBIS Models.....30
2.4.1. Elements of an IBIS Model.....31
2.4.2. Creating Accurate IBIS Models.....31
2.4.3. Design Simulation Using the Mentor Graphics HyperLynx Software.....33
2.4.4. Configuring LineSim to Use Intel IBIS Models.....35
2.4.5. Integrating Intel IBIS Models into LineSim Simulations.....37
2.4.6. Running and Interpreting LineSim Simulations.....38
2.5. Simulation with HSPICE Models.....40

2.5.1. Supported Devices and Signaling..... 40
2.5.2. Accessing HSPICE Simulation Kits..... 40
2.5.3. The Double Counting Problem in HSPICE Simulations..... 41
2.5.4. HSPICE Writer Tool Flow..... 43
2.5.5. Running an HSPICE Simulation..... 45
2.5.6. Interpreting the Results of an Output Simulation..... 46
2.5.7. Interpreting the Results of an Input Simulation..... 46
2.5.8. Viewing and Interpreting Tabular Simulation Results..... 47
2.5.9. Viewing Graphical Simulation Results..... 47
2.5.10. Making Design Adjustments Based on HSPICE Simulations..... 48
2.5.11. Sample Input for I/O HSPICE Simulation Deck..... 50
2.5.12. Sample Output for I/O HSPICE Simulation Deck..... 54
2.5.13. Advanced Topics..... 60
2.6. Document Revision History..... 61

3. Mentor Graphics PCB Design Tools Support..... 62
3.1. FPGA-to-PCB Design Flow..... 63
3.2. Integrating with I/O Designer..... 65
3.2.1. Generating Pin Assignment Files..... 66
3.2.2. I/O Designer Settings..... 67
3.2.3. Transferring I/O Assignments..... 68
3.2.4. Updating I/O Designer with Intel Quartus Prime Pin Assignments..... 70
3.2.5. Updating Intel Quartus Prime with I/O Designer Pin Assignments..... 71
3.2.6. Generating Schematic Symbols in I/O Designer..... 71
3.2.7. Exporting Schematic Symbols to DxDesigner..... 73
3.3. Integrating with DxDesigner..... 73
3.3.1. DxDesigner Project Settings..... 73
3.3.2. Creating Schematic Symbols in DxDesigner..... 74
3.4. Analyzing FPGA Simultaneous Switching Noise (SSN)..... 74
3.5. Scripting API..... 74
3.6. Document Revision History..... 75

4. Cadence PCB Design Tools Support..... 76
4.1. Cadence PCB Design Tools Support..... 76
4.2. Product Comparison..... 77
4.3. FPGA-to-PCB Design Flow..... 77
4.3.1. Integrating Intel FPGA Design..... 79
4.3.2. Performing Simultaneous Switching Noise (SSN) Analysis of Your FPGA..... 79
4.4. Setting Up the Intel Quartus Prime Software..... 79
4.4.1. Generating a .pin File..... 80
4.5. FPGA-to-Board Integration with the Cadence Allegro Design Entry HDL Software..... 80
4.5.1. Creating Symbols..... 81
4.5.2. Instantiating the Symbol in the Cadence Allegro Design Entry HDL Software..... 86
4.6. FPGA-to-Board Integration with Cadence Allegro Design Entry CIS Software..... 87
4.6.1. Creating a Cadence Allegro Design Entry CIS Project..... 88
4.6.2. Generating a Part..... 88
4.6.3. Generating Schematic Symbol..... 89
4.6.4. Splitting a Part..... 89
4.6.5. Instantiating a Symbol in a Design Entry CIS Schematic..... 91
4.6.6. Intel Libraries for the Cadence Allegro Design Entry CIS Software..... 91
4.7. Document Revision History..... 93

5. Reviewing Printed Circuit Board Schematics with the Intel Quartus Prime Software.... 94
5.1. Reviewing Intel Quartus Prime Software Settings..... 94
5.1.1. Device and Pins Options Dialog Box Settings.....95
5.2. Reviewing Device Pin-Out Information in the Fitter Report.....96
5.3. Reviewing Compilation Error and Warning Messages..... 98
5.4. Using Additional Intel Quartus Prime Software Features..... 98
5.5. Using Additional Intel Quartus Prime Software Tools..... 98
5.5.1. Pin Planner.....99
5.5.2. SSN Analyzer..... 99
5.6. Document Revision History.....99

A. Intel Quartus Prime Standard Edition User Guides.....100

1. Simultaneous Switching Noise (SSN) Analysis and Optimizations

1.1. Simultaneous Switching Noise (SSN) Analysis and Optimizations

FPGA design has evolved from small programmable circuits to designs that compete with multimillion-gate ASICs. At the same time, the I/O counts on FPGAs and logic density requirements of designs have increased exponentially.

The higher-speed interfaces in FPGAs, including high-speed serial interfaces and memory interfaces, require careful interface design on the PCB. Designers must address the timing and signal integrity requirements of these interfaces early in the design cycle. Simultaneous switching noise (SSN) often leads to the degradation of signal integrity by causing signal distortion, thereby reducing the noise margin of a system.

Today's complex FPGA system design is incomplete without addressing the integrity of signals coming in to and out of the FPGA. Altera recommends that you perform SSN analysis early in your FPGA design and prior to the layout of your PCB with complete SSN analysis of your FPGA in the Intel® Quartus® Prime software. This chapter describes the Intel Quartus Prime SSN Analyzer tool and covers the following topics:

1.2. Definitions

The terminology used in this chapter includes the following terms:

- **Aggressor**: An output or bidirectional signal that contributes to the noise for a victim I/O pin
- **PDN**: Power distribution network
- **QH**: Quiet high signal level on a pin
- **QHN**: Quiet high noise on a pin, measured in volts
- **QL**: Quiet low signal level on a pin
- **QLN**: Quiet low noise on a pin, measured in volts
- **SI**: Signal integrity (a superset of SSN, covering all noise sources)
- **SSN**: Simultaneous switching noise
- **SSO**: Simultaneous switching output (which are either the output or bidirectional pins)
- **Victim**: An input, output, or bidirectional pin that is analyzed during SSN analysis. During SSN analysis, each pin is analyzed as a victim. If a pin is an output or bidirectional pin, the same pin acts as an aggressor signal for other pins.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

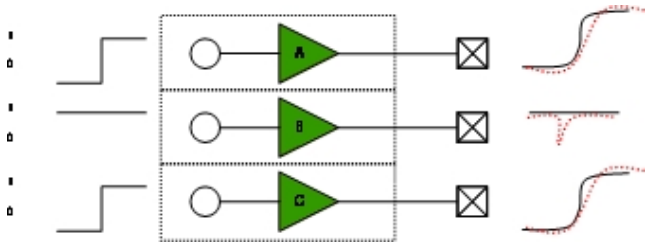
1.3. Understanding SSN

SSN is defined as a noise voltage induced onto a single victim I/O pin on a device due to the switching behavior of other aggressor I/O pins on the device. SSN can be divided into two types of noise: voltage noise and timing noise.

In a sample system with three pins, two of the pins (A and C) are switching, while one pin (B) is quiet. If the pins are driven in isolation, the voltage waveforms at the output of the buffers appear without noise interference, as shown by the solid curves at the left of the figure. However, when pins A and C are switching simultaneously, the noise generated by the switching is injected onto other pins. This noise manifests itself as a voltage noise on pin B and timing noise on pins A and C.

Figure 1. System with Three Pins

In this figure, the dotted curves show the voltage noise on pin B and timing noise on pins A and C.



Voltage noise is measured as the change in voltage of a signal due to SSN. When a signal is QH, it is measured as the change in voltage toward 0 V. When a signal is QL, it is measured as the change in voltage toward V_{CC}.

In the Intel Quartus Prime software, only voltage noise is analyzed. Voltage noise can be caused by SSOs under two worst-case conditions:

- The victim pin is high and the aggressor pins (SSOs) are switching from low to high
- The victim pin is low and the aggressor pins (SSOs) are switching from high to low

Figure 2. Quiet High Output Noise Estimation on Pin B

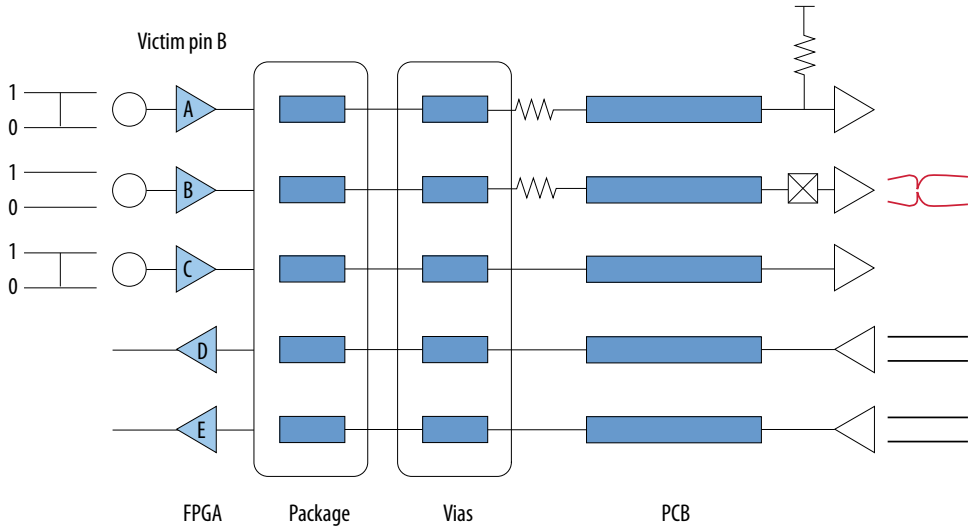
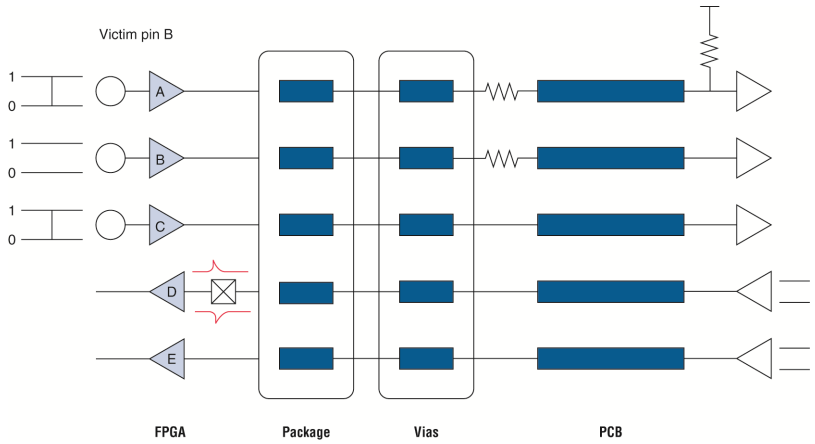
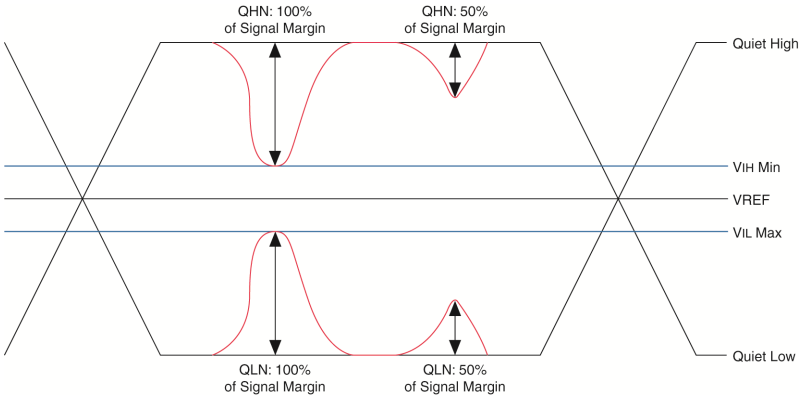


Figure 3. Quiet Low Input Noise Estimation for Pin D



SSN can occur in any system, but the induced noise does not always result in failures. Voltage functional errors are caused by SSN on quiet victim pins only when the voltage values on the quiet pins change by a large voltage that the logic listening to that signal reads a change in the logic value. For QH signals, a voltage functional error occurs when noise events cause the voltage to fall below V_{IH}. Similarly, for QL signals, a voltage functional error occurs when noise events cause the voltage to rise above V_{IL}. Because V_{IH} and V_{IL} of the Altera device are different for different I/O standards, and because signals have different quiet voltage values, the absolute amount of SSN, measured in volts, cannot be used to determine if a voltage failure occurs. Instead, to assess the level of impact by SSN in the SSN analysis, the Intel Quartus Prime software quantifies the SSN in terms of the percentage of signal margin in Intel devices.

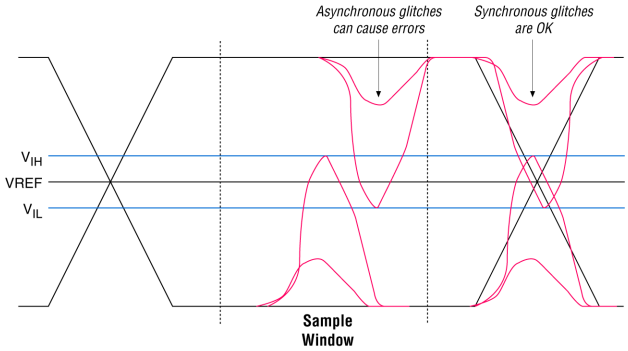
Figure 4. Reporting Noise Margins



The figure shows four noise events, two on QH signals and two on QL signals. The two noise events on the right-side of the figure consume 50 percent of the signal margin and do not cause voltage functional errors. However, the two noise events on the left side of the figure consume 100 percent of the signal margin, which can cause a voltage functional error.

Noise caused by aggressor signals are synchronously related to the victim pin outside of the sampling window of a receiver. This noise affects the switching time of a victim pin, but are not considered an input threshold violation failure.

Figure 5. Synchronous Voltage Noise with No Functional Error



Related Information

SSN Analysis Overview on page 9

1.4. SSN Estimation Tools

Addressing SSN early in your FPGA design and PCB layout can help you avoid costly board respins and lost time, both of which can impact your time-to-market.

Intel provides many tools for SSN analysis and estimation, including the following tools:

- SSN characterization reports
- An early SSN estimation (ESE) tool
- The SSN Analyzer in the Intel Quartus Prime software

The ESE tool is useful for preliminary SSN analysis of your FPGA design; for more accurate results, however, you must use the SSN Analyzer in the Intel Quartus Prime software.

Table 1. Comparison of ESE Tool and SSN Analyzer Tool

ESE Tool	SSN Analyzer
Is not integrated with the Intel Quartus Prime software.	Integrated with the Intel Quartus Prime software, allowing you to perform preliminary SSN analysis while making I/O assignment changes in the Intel Quartus Prime software.
QL and QH levels are computed assuming a worst-case pattern of I/O placements.	QL and QH levels are computed based on the I/O placements in your design.
No support for entering board information.	Supports board trace models and board layer information, resulting in a more accurate SSN analysis.
No graphical representation.	Integrated with the Intel Quartus Prime Pin Planner, in which an SSN map shows the QL and QH levels on victim pins.
Good for doing an early SSN estimate. Does not require you to use the Intel Quartus Prime software.	Requires you to create a Intel Quartus Prime software project and provide the top-level port information.

Related Information

Altera Signal Integrity Center

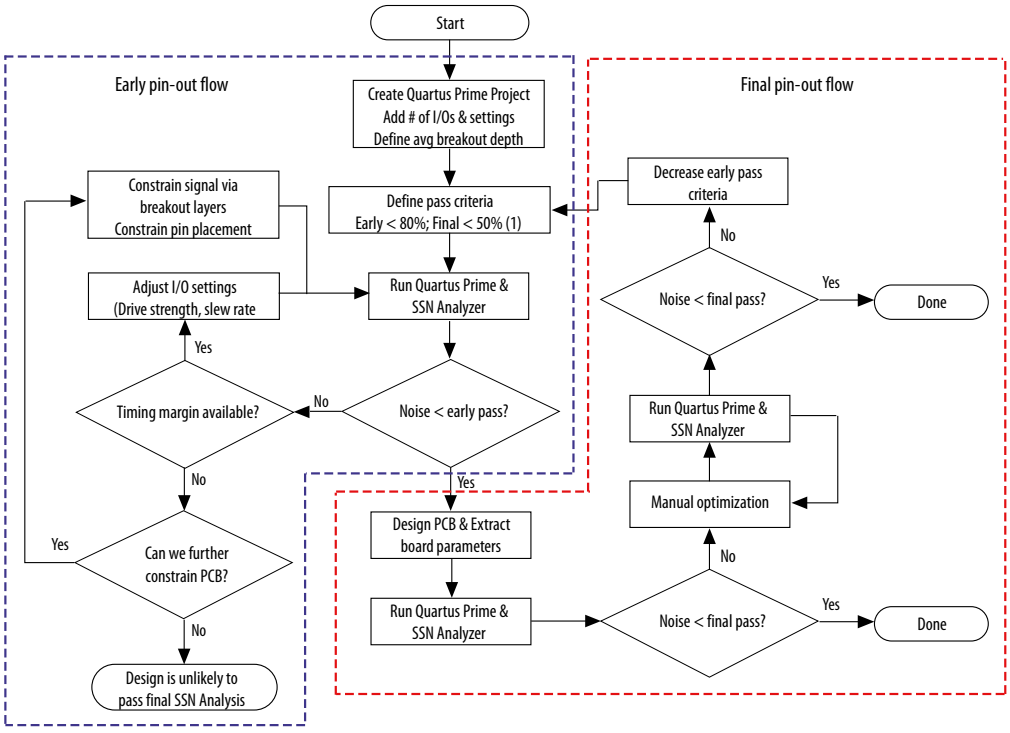
1.5. SSN Analysis Overview

You can run the SSN Analyzer at different stages in your design cycle to obtain SSN results. The accuracy of the results depends on the completeness of your design information.

Start SSN analysis early in the design cycle to obtain preliminary results and adjust your I/O assignments. Iterate through the design cycle to finally perform a fully constrained SSN analysis with complete information about your board.

The early pin-out flow assumes conservative design rules initially, and then lets you analyze the design and iteratively apply tighter design rules until SSN analysis indicates your design meets SSN constraints. You must define pass criteria for SSN analysis as a percentage of signal margin in both the early pin-out flow and the final pin-out flow. The pass criteria you define is specific to your design requirements. For example, a pass criterion you might define is a condition that verifies you have sufficient SSN margins in your design. You may require that the acceptable voltage noise on a pin must be below 70% of the voltage level for that pin. The pass criteria for the early-pin out flow may be higher than the final pin-out flow criteria, so that you do not spend too much time optimizing the on-FPGA portions of your design when the SSN metrics for the design may improve after the design is fully specified.

Figure 6. Early Flow and Final Pin-Out SSN Analysis



Note:

1. Pass criteria determined by customer requirements.

1.5.1. Performing Early Pin-Out SSN Analysis

In the early stages of your design cycle, before you create pin location for your design, use the early pin-out flow to obtain preliminary SSN analysis results.

To obtain useful SSN results, you must define the top-level ports of your design, but your design files do not have to be complete.

Performing Early Pin-Out SSN Analysis with the ESE Tool

If you know the I/O standards and signaling standards for your design, you can use the ESE tool to perform an initial SSN evaluation.

1.5.1.1. Performing Early Pin-Out SSN Analysis with the SSN Analyzer

In the early stages of your design cycle, you may not have complete board information, such as board trace parameters, layer information, and the signal breakout layers. If you run the SSN Analyzer without this specific information, it uses default board trace models and board layer information for SSN analysis, and as a result the SSN Analyzer confidence level is low. If the noise amounts are larger than the pass criteria for early pin-out SSN analysis, verify whether the SSN noise violations are true failures or false failures. For example, sometimes the SSN Analyzer can determine whether pins are switching synchronously and use that information to filter false positives; however, it may not be able to determine all the synchronous groups. You can improve the SSN analysis results by adjusting your I/O assignments and other design settings. After you optimize your design such that it meets the pass criteria for the early pin-out flow, you can then begin to design your PCB.

If you have complete information for the top-level ports of your design, you can use the SSN Analyzer to perform an initial SSN evaluation. Use the following steps to perform early pin-out SSN analysis:

1. Create a project in the Intel Quartus Prime software.
2. Specify your top-level design information either in schematic form or in HDL code.
3. Perform Analysis and Synthesis.
4. Create I/O assignments, such as I/O standard assignments, for the top-level ports in your design.

   Note: Do not create pin location assignments. The Fitter automatically creates optimized pin location assignments.
5. If you do not have completed design files and timing constraints, run I/O assignment analysis.

   Note: During I/O assignment analysis, the Fitter places all the unplaced pins on the device, and checks all the I/O placement rules.
6. Run the SSN Analyzer.

Related Information

- Optimizing Your Design for SSN Analysis on page 11
- Managing Intel Quartus Prime Projects
  In *Intel Quartus Prime Standard Edition Handbook Volume 1*

1.5.2. Performing Final Pin-Out SSN Analysis

You perform final pin-out SSN analysis after you place all the pins in your design, or the Fitter places them for you, and you have complete information about the board trace models and PCB layers.

Even if your design achieves sufficient SSN results during early pin-out SSN analysis, you should run SSN analysis with the complete PCB information to ensure that SSN does not cause failures in your final design. You must specify the board parameters in the Intel Quartus Prime software, including the PCB layer thicknesses, the signal breakout layers, and the board trace models, before you can run SSN analysis on your final assignments.

If the SSN analysis results meet the pass criteria for final pin-out SSN analysis, SSN analysis is complete. If the SSN analysis results do not meet the pass criteria, you must further optimize your design by changing the board and design parameters and then rerun the SSN Analyzer. If the design still does not meet the pass criteria, reduce the pass criteria for early pin-out SSN analysis, and restart the process. By reducing the pass criteria for early pin-out SSN analysis, you place a greater emphasis on reducing SSN through I/O settings and I/O placement. Changing the drive strength and slew rate of output and bidirectional pins, as well as adjusting the placement of different SSOs, can affect SSN results. Adjusting I/O settings and placement allows the design to meet the pass criteria for final pin-out SSN analysis after you specify the actual PCB board parameters.

1.6. Design Factors Affecting SSN Results

There are many factors that affect the SSN levels in your design. The two main factors are the drive strength and slew rate settings of the output and bidirectional pins in your design.

Related Information

Altera Signal Integrity Center

1.7. Optimizing Your Design for SSN Analysis

The SSN Analyzer gives you flexibility to precisely define your system to obtain accurate SSN results.

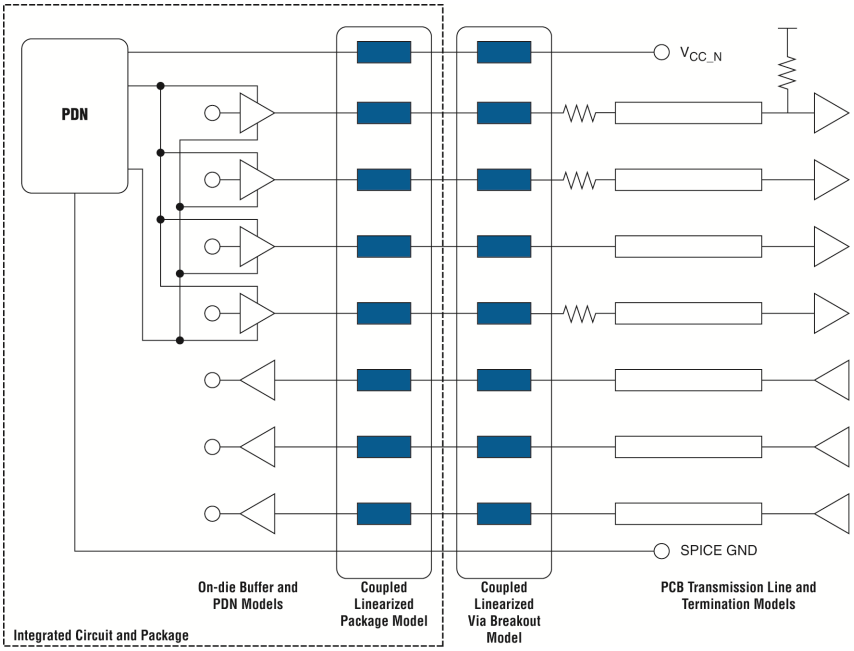
The SSN Analyzer produces a voltage noise estimate for each input, output, and bidirectional pin in the design. It allows you to estimate the SSN levels, comprised of QLN and QHN levels, for your FPGA pins. Performing SSN analysis helps you optimize your design for SSN during compilation.

Because the SSN Analyzer is integrated into the Intel Quartus Prime software, it can automatically set up a system topology that matches your design. The SSN Analyzer accounts for different I/O standards and slew rate settings for each buffer in the design and models different board traces for each signal. Also, it correctly models the state of the unused pins in the design. The SSN Analyzer leverages any custom board trace assignments you set up for use by the advanced I/O timing feature.

The SSN Analyzer also models the package and vias in the design. Models for the different packages that Altera devices support are integrated into the Intel Quartus Prime software. In the Intel Quartus Prime software, you can specify different layers on which signals break out, each with its own thickness, and then specify which signal breaks out on which layer.

After constructing the circuit topology, the SSN Analyzer uses a simulation-based methodology to determine the SSN for each victim pin in the design.

Figure 7. Circuit Topology for SSN Analysis



1.7.1. Optimizing Pin Placements for Signal Integrity

The **SSN Optimization** logic option tells the Fitter to adjust the pin placement to reduce the SSN in the design.

The **SSN Optimization** logic option has three possible values: **Off**, **Normal compilation**, and **Extra Effort**.

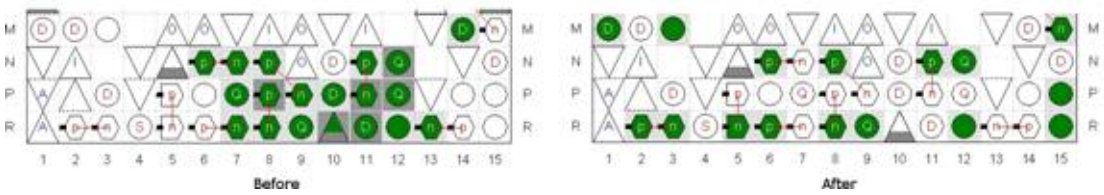
If you use the **SSN Optimization** logic option, avoid creating location assignments for your pins. Instead, let the Fitter place the pins during compilation to meet the timing performance of your design.

To display the Fitter-placed pins use the **Show Fitter Placements** feature in the Pin Planner. To accept these suggested pin locations, you must back-annotate your pin assignments.

Note: Setting the **SSN Optimization** option to **Extra effort** may impact your design f_{MAX}.

Figure 8. SSN Analysis Results Before and After Using the SSN Optimization Logic Option

The image shows the pin placement before and after turning on the **SSN Optimization** logic option.



Related Information

- Show Commands (View Menu/Task Window) (Pin Planner)
  In Intel Quartus Prime Help
- link/zov1529446404644/mwh1410471288350
- link/zov1529446404644/mwh1410471203263

1.7.2. Specifying Board Trace Model Settings

The SSN Analyzer uses circuit models to determine voltage noise during SSN analysis. The circuit topology is incomplete without board trace information and PCB layer information.

To accurately compute the SSN in your FPGA device, you must describe the board trace and PCB layer parameters in your design. If you do not specify some or all the board trace parameters and PCB layer information, the SSN Analyzer uses default parameters, which set the SSN confidence level to low.

The SSN Analyzer requires board trace models such as termination resistors, pin loads (capacitance), and transmission line parameters. You can define the board circuit models—or board trace models, in the Intel Quartus Prime software.

You can define an overall board trace model for each I/O standard in your design. This overall model is the default model for all pins that use a particular I/O standard. After configuring the overall board trace model, you can customize the model for specific pins.

Intel Quartus Prime software uses the parameters you specify for the board trace model during advanced I/O timing analysis with the Timing Analyzer.

If you already specified the board trace models as part of your advanced I/O timing assignments, Intel Quartus Prime software uses the same parameters during SSN analysis.

All the assignments for board trace models you specify are saved to the `.qsf` file. You can also use Tcl commands to create board trace model assignments.

Tcl Commands for Specifying Board Trace Models

```
set_instance_assignment -name BOARD_MODEL_TLINE_L_PER_LENGTH "3.041E-7" -to e[0]
set_instance_assignment -name BOARD_MODEL_TLINE_LENGTH 0.1391 -to e[0]
set_instance_assignment -name BOARD_MODEL_TLINE_C_PER_LENGTH "1.463E-10" -to e[0]
```

The best way to calculate transmission line parameters is to use a two-dimensional solver to estimate the inductance per inch and capacitance per inch for the transmission line. You can obtain the termination resistor topology information from the PCB schematics. The near-end and far-end pin load (capacitance) values can be obtained from the PCB schematic and other device data sheets. For example, if you know that an FPGA pin is driving a DIMM, you can obtain the far-end loading information in the data sheet for your target device.

Related Information

- link/grc1529967026944/mwh1410471036713
- Board Trace Model
  In Intel Quartus Prime Help.
- Literature and Technical Documentation

1.7.3. Defining PCB Layers and PCB Layer Thickness

Every PCB is fabricated using a number of layers. To remove some of the pessimism from your SSN results, Altera recommends that you create assignments describing your PCB layers in the Intel Quartus Prime software.

You can specify the number of layers on you PCB, and their thickness. The PCB layer information is used only during SSN analysis and is not used in other processes run by the Intel Quartus Prime software. If a custom PCB breakout region is not described you can select the default thickness, which directs the SSN Analyzer to use a single-layer PCB breakout region during SSN analysis.

All the assignments you create for the PCB layers are saved to the `.qsf`. You can also use Tcl commands to create PCB layer assignments. You can create any number of PCB layers, however, the layers must be consecutive.

Tcl Commands for Specifying PCB Layer Assignments

```
set_global_assignment -name PCB_LAYER_THICKNESS 0.00099822M -section_id 1
set_global_assignment -name PCB_LAYER_THICKNESS 0.00034036M -section_id 2
set_global_assignment -name PCB_LAYER_THICKNESS 0.00034036M -section_id 3
```

The cross-section shows the stackup information of a PCB, which tells you the number of layers used in your PCB. The PCB shown in this example consists of various signal and circuit layers on which FPGA pins are routed, as well as the power and ground layers.

Figure 9. Snapshot of Stackup of a PCB Shown in the Allegro Board Design Environment

Layout Cross Section

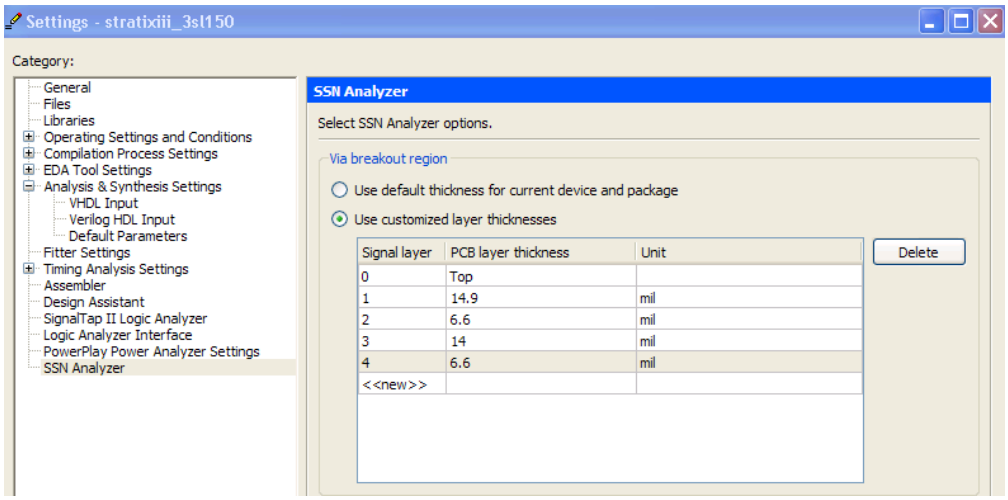
Cross Section

	Subclass Name	Type	Material	Thickness (MIL)
1		SURFACE	AIR	
2	TOP	CONDUCTOR	PLATED_COPPER_FOIL	1.9
3		DIELECTRIC	FR-4	3.6
4	L2-PWR	PLANE	COPPER	1.2
5		DIELECTRIC	FR-4	3
6	L3-GND	PLANE	COPPER	1.2
7		DIELECTRIC	FR-4	4
8	L4-SIGNAL	CONDUCTOR	COPPER	0.6
9		DIELECTRIC	FR-4	6
10	L5-SIGNAL	CONDUCTOR	COPPER	0.6
11		DIELECTRIC	FR-4	4
12	L6 GND	PLANE	COPPER	1.2
13		DIELECTRIC	FR-4	3
14	L7-PWR	PLANE	COPPER	1.2
15		DIELECTRIC	FR-4	4
16	L8-SIGNAL	CONDUCTOR	COPPER	0.6
17		DIELECTRIC	FR-4	6
18	L9-SIGNAL	CONDUCTOR	COPPER	0.6
19		DIELECTRIC	FR-4	4
20	L10-GND	PLANE	COPPER	1.2
21		DIELECTRIC	FR-4	3

In this example, each of the four signal layers are a different thickness, with the depths shown in the **Thickness (MIL)** column. The layer thickness for each signal layer is computed as follows:

- Signal Layer 1 is the L4-SIGNAL, at thickness (1.9+3.6+1.2+3+1.2+4=) 14.9 mils
- Signal Layer 2 is the L5-SIGNAL, at thickness (0.6+6=) 6.6 mils
- Signal Layer 3 is the L8-SIGNAL, at thickness (0.6+4+1.2+3+1.2+4=) 14 mils
- Signal Layer 4 is the L9-SIGNAL, at thickness (0.6+6=) 6.6 mils

Figure 10. PCB Layers and Thickness Assignments Specified in the Intel Quartus Prime Software



1.7.4. Specifying Signal Breakout Layers

Each user I/O pin in your FPGA device can break out at different layers on your PCB. In the Pin Planner, you can specify on which layers the I/O pins in your design break out.

The breakout layer information is used only during SSN analysis and is not used in other processes run by the Intel Quartus Prime software. To assign a pin to PCB layer, follow these steps:

1. On the Assignments menu, click **Pin Planner**.
2. If necessary, perform Analysis & Elaboration, Analysis & Synthesis, or fully compile the design to populate the Pin Planner with the node names in the design.
3. Right-click anywhere in the **All Pins** or **Groups** list, and then click **Customize Columns**.
4. Select the **PCB layer** column and move it from the **Available columns** list to the **Show these columns in this order** list.
5. Click **OK**.
6. In the **PCB layer** column, specify the PCB layer to which you want to connect the signal.
7. On the File menu, click **Save Project** to save the changes.

Note: When you create PCB breakout layer assignments in the Pin Planner, you can assign the pin to any layer, even if you did not yet define the PCB layer.

1.7.5. Creating I/O Assignments

I/O assignments are required in FPGA design and are also used during SSN analysis to estimate voltage noise.

Each input, output, or bidirectional signal in your design is assigned a physical pin location on the device using pin location assignments. Each signal has a physical I/O buffer that has a specific I/O standard, pin location, drive strength, and slew rate. The SSN Analyzer supports most I/O standards in a device family, such as the **LVTTL** and **LVCMOS** I/O standards.

Note: The SSN Analyzer does not support differential I/O standards, such as the **LVDS** I/O standard and its variations, because differential I/O standards contribute a small amount of SSN.

Related Information

- Literature and Technical Documentation
  For more information on the Altera website about supported I/O standards.
- I/O Management
  For more information about creating and managing I/O assignments, refer to the *Intel Quartus Prime Handbook*.

1.7.6. Decreasing Pessimism in SSN Analysis

In the absence of specific timing information, the SSN Analyzer analyzes your design under worst-case conditions.

Worst-case conditions include all pins acting as aggressor signals on all possible victim pins and all aggressor pins switching with the worst possible timing relationship. The results of SSN analysis under worst-case conditions are very pessimistic. You can improve the results of SSN Analysis by creating group assignments for specific types of pins. Use the following group assignments to decrease the pessimism in SSN analysis results:

- Assign pins to an output enable group—All pins in an output enable group must be either all input pins or all output pins. If all the pins in a group are always either all inputs or all outputs, it is impossible for an output pin in the group to cause SSN noise on an input pin in the group. You can assign pins to an output enable group with the **Output Enable Group** logic option.
- Assign pins to a synchronous group—I/O pins that are part of a synchronous group (signals that switch at the same time) may cause SSN, but do not result in any failures because the noise glitch occurs during the switching period of the signal. The noise, therefore, does not occur in the sampling window of that signal. You can assign pins to an output enable group with the **Synchronous Group** logic option. For example, in your design you have a bus with 32 pins that all belong to the same group. In a real operation, the bus switches at the same time, so any voltage noise induced by a pin on its groupmates does not matter, because it does not fall in the sampling window. If you do not assign the bus to a synchronous group, the other 31 pins can act as aggressors for the first pin in that group, leading to higher QL and QH noise levels during SSN analysis.

In some cases, the SSN Analyzer can detect the grouping for bidirectional pins by looking at the output enable signal of the bidirectional pins. However, Altera recommends that you explicitly specify the bidirectional groups and output groups in your design.

1.7.7. Excluding Pins as Aggressor Signals

The SSN Analyzer uses the following conditions to exclude pins as aggressor signals for a specific victim pin:

- A pin that is a complement of the victim pin. For example, any pin that is assigned a differential I/O standard cannot be an aggressor pin.
- A programming pin or JTAG pin because these pins are not active in user mode.
- Pins that have the same output enable signal as a bidirectional victim pin that the SSN Analyzer analyzes as an input pin. Pins with the same output enable signal also act as input pins and therefore cannot be aggressor pins at the same time.
- Pins in the same synchronous group as a victim output pin.
- A pin assigned the **I/O Maximum Toggle Rate** logic option with a frequency setting of zero. The SSN Analyzer does not consider pins with this setting as aggressor pins.

1.8. Performing SSN Analysis and Viewing Results

You can perform SSN analysis either on your entire design, or you can limit the analysis to specific I/O banks.

If you know the problem area for SSN is within one I/O bank and you are changing pin assignments only in that bank, you can run SSN analysis for just that one I/O bank to reduce analysis time.

Related Information

Literature and Technical Documentation
For more information about I/O bank numbering refer to the appropriate device handbook available on the Altera website.

1.8.1. Understanding the SSN Reports

When SSN analysis is complete, you can view detailed analysis reports. The detailed messages in the reports help you understand and resolve SSN problems.

The SSN Analyzer section of the Compilation report contains information generated during analysis, including the following reports:

Summary Report on page 19

Output Pins Report and Input Pins Report on page 19

Unanalyzed Pins Report on page 19

Confidence Metric Details on page 19

1.8.1.1. Summary Report

The Summary report summarizes the SSN Analyzer status and rates the SSN Analyzer confidence level as low, medium, or high.

The confidence level depends on the completeness of your board trace model assignments. The more assignments you complete, the higher the confidence level. However, the confidence level does not always contribute to the accuracy of the QL and QH noise levels on a victim pin. The accuracy of QH and QL noise levels depends the accuracy of your board trace model assignments.

1.8.1.2. Output Pins Report and Input Pins Report

The Output Pins report lists all the output pins and bidirectional pins that the SSN analyzer considers as outputs. The Input Pins report lists all the input pins and bidirectional pins that the SSN analyzer considers as inputs. Both reports list:

- Location assignments for the pins.
- QL and QH noise in volts.
- For the I/O standard used for that signal, what percentage the QL and QH margins are.

The QH and QL noise margins that fall in the critical range (> 90%) are shown in red. The QH and QL noise margins that fall in the range of 70% to 90% are shown in gray.

1.8.1.3. Unanalyzed Pins Report

The SSN analyzer doesn't analyze all pins. The ignored pins are reported in the Unanalyzed Pins report:

- Pins assigned the **LVDS** I/O standard or any LVDS variations, such as the **mini-LVDS** I/O standard.
- Pins created in the migration flow that cover power and supply pins in other packages.
- The negative terminals of pseudo-differential I/O standards; the noise on differential standards is reported as the differential noise and is reported on the positive terminal.

1.8.1.4. Confidence Metric Details

The Confidence Metric Details Report lists the values the SSN analyzer uses for unspecified I/O, board, and PCB assignments.

1.8.2. Viewing SSN Analysis Results in the Pin Planner

After SSN analysis completes, you can analyze the results in the Pin Planner. In the Pin Planner you can identify the SSN hotspots in your device, as well as the QL and QH noise levels.

The QL and QH results for each pin are displayed with a different color that represents whether the pin is below the warning threshold, below the critical threshold, or above the critical threshold. This color representation is also referred to as the SSN map of your FPGA device.

When you view the SSN map, you can customize which details to display, including input pins, output pins, QH signals, QL signals, and noise levels. You can also adjust the threshold levels for QH and QL noise voltages. Adjusting the threshold levels in the Pin Planner does not change the threshold levels reported during SSN analysis and does not change the data in any of the SSN reports.

You can also you change I/O assignments and board trace information and rerun the SSN Analyzer to view the SSN analysis results based on those modified settings.

Related Information

Show SSN Analyzer Results

1.9. Decreasing Processing Time for SSN Analysis

FPGA designs are getting larger in density, logic, and I/O count. The time it takes to complete SSN analysis and other Intel Quartus Prime software processes affects your development time.

Faster processing times can reduce your design cycle time. Use the following guidelines to reduce processing time:

- Direct the Intel Quartus Prime software to use more than one processor for parallel executables, including the SSN Analyzer
- Perform SSN analysis after I/O assignment analysis if your design files and constraints are complete, and you are interested in generating the SSN results early in the design process and want to adjust I/O placements to see if you can obtain better results
- Perform SSN analysis after fitting if you want to view preliminary SSN results that do not take into account complete I/O assignment and I/O timing results
- Perform engineering change orders (ECOs) on your design, rather than recompiling the entire design, if you want to rerun SSN analysis after changing I/O assignments

Related Information

- Compilation Process Settings Page
  For more information about using parallel processors, refer to Intel Quartus Prime Help.
- Engineering Change Management with the Chip Planner
  For more information about performing ECOs on your design, refer to the *Intel Quartus Prime Handbook*.

1.10. Scripting Support

A Tcl script allows you to run procedures and determine settings. You can also run some of these procedures at a command prompt.

The Intel Quartus Prime software provides several packages to compile your design and create I/O assignments for analysis and fitting. You can create a custom Tcl script that maps the design and runs SSN analysis on your design.

For detailed information about specific scripting command options and Tcl API packages, type the following command at a system command prompt to run the Intel Quartus Prime Command-Line and Tcl API Help browser:

```
quartus_sh --qhelp
```

Related Information

- Tcl Scripting
- Command-Line Scripting
  For more information about Intel Quartus Prime scripting support, including examples, refer to the *Intel Quartus Prime Handbook*.
- API Functions for Tcl
  For more information about Intel Quartus Prime scripting support, including examples, refer to Intel Quartus Prime Help.

1.10.1. Optimizing Pin Placements for Signal Integrity

You can create an assignment that directs the Fitter to optimize pin placements for signal integrity with a Tcl command.

The following Tcl command directs the Fitter to optimize pin placement for signal integrity without affecting design f_{MAX}:

```
set_global_assignment -name OPTIMIZE_SIGNAL_INTEGRITY "Normal Compilation"
```

Related Information

Optimizing Pin Placements for Signal Integrity on page 12

1.10.2. Defining PCB Layers and PCB Layer Thickness

You can create PCB layer and thickness assignments with a Tcl command.

Tcl Commands for Specifying PCB Layer Assignments

```
set_global_assignment -name PCB_LAYER_THICKNESS 0.00099822M -section_id 1
set_global_assignment -name PCB_LAYER_THICKNESS 0.00034036M -section_id 2
set_global_assignment -name PCB_LAYER_THICKNESS 0.00034036M -section_id 3
set_global_assignment -name PCB_LAYER_THICKNESS 0.00055372M -section_id 4
set_global_assignment -name PCB_LAYER_THICKNESS 0.00034036M -section_id 5
set_global_assignment -name PCB_LAYER_THICKNESS 0.00034036M -section_id 6
set_global_assignment -name PCB_LAYER_THICKNESS 0.00082042M -section_id 7
```

These Tcl commands specify that there are seven PCB layers in the design, each with a different thickness. In each assignment, the letter `M` indicates the unit of measurement is millimeters. When you specify PCB layer assignments with Tcl commands, you must list the layers in consecutive order. For example, you would receive an error during SSN Analysis if your Tcl commands created the following assignments:

```
set_global_assignment -name PCB_LAYER_THICKNESS 0.00099822M -section_id 1
set_global_assignment -name PCB_LAYER_THICKNESS 0.00082042M -section_id 7
```

To create assignments with the unit of measurement in mils, refer to the syntax in the following Tcl commands.

```
set_global_assignment -name PCB_LAYER_THICKNESS 14.9MIL -section_id 1
set_global_assignment -name PCB_LAYER_THICKNESS 6.6MIL -section_id 2
set_global_assignment -name PCB_LAYER_THICKNESS 14MIL -section_id 3
set_global_assignment -name PCB_LAYER_THICKNESS 6.6MIL -section_id 4
```

Related Information

Defining PCB Layers and PCB Layer Thickness on page 14

1.10.3. Specifying Signal Breakout Layers

You can create signal breakout layer assignments with a Tcl command.:

```
set_instance_assignment -name PCB_LAYER 10 -to e[2] set_instance_assignment -
name PCB_LAYER 3 -to e[3]
```

When you create PCB breakout layer assignments with Tcl commands, if you do not specify a PCB layer, or if you specify a PCB layer that does not exist, the SSN Analyzer breaks out the signal at the bottommost PCB layer.

Note: If you create a PCB layer breakout assignment to a layer that does not exist, the SSN Analyzer will generate a warning message.

1.10.4. Decreasing Pessimism in SSN Analysis

You can create output enable group and synchronous group assignments to help decrease pessimism during SSN Analysis with a Tcl command.

The following Tcl command assigns the bidirectional bus `DATAINOUT` to an output enable group:

```
set_instance_assignment -name OUTPUT_ENABLE_GROUP 1 -to DATAINOUT
```

The following Tcl command assigns the bus `PCI_ADD_io` to a synchronous group:

```
set_instance_assignment -name SYNCHRONOUS_GROUP 1 -to PCI_AD_io
```

Related Information

Decreasing Pessimism in SSN Analysis on page 17

1.10.5. Performing SSN Analysis

You can perform SSN analysis with a command-line command. Use the `quartus_si` package that is provided with the Intel Quartus Prime software.

Type the following command at a system command prompt to start the SSN Analyzer:

```
quartus_si <project name>
```

To analyze just one I/O bank, type the following command at a system command prompt:

```
quartus_si <project revision> <--bank = bank id>
```

For example, to run analyze the I/O bank 2A type the following command:

```
quartus_si counter --bank=2A
```

For more information about the `quartus_si` package, type `quartus_si -h` at a system command prompt.

Related Information

Performing SSN Analysis and Viewing Results on page 18

1.11. Document Revision History

Date	Version	Changes
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
December 2014	14.1.0	• Minimal text edits for clarity in the topic about understanding SSN.
June 2014	14.0.0	Updated format.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update
December 2010	10.0.1	Template update
July 2010	10.0.0	• Reorganized and edited the chapter • Added links to Intel Quartus Prime Help for procedural information previously included in the chapter
November 2009	9.1.0	• Added "Figure 6–9 shows the layout cross-section of a PCB in the Cadence Allegro PCB tool. The cross-section shows the stackup information of a PCB, which tells you the number of layers used in your PCB. The PCB shown in this example consists of various signal and circuit layers on which FPGA pins are routed, as well as the power and ground layers." on page 6–12 • Updated for the Intel Quartus Prime software 9.1 release
March 2009	9.0.0	Initial release

2. Signal Integrity Analysis with Third-Party Tools

2.1. 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 a single PCB is built.

If the board trace is designed poorly 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 printed circuit board (PCB) are important aspects to consider to ensure proper system operation. To avoid time-consuming redesigns and expensive board respins, the topology and routing of critical signals must be simulated. The high-speed interfaces available on current FPGA devices must be modeled accurately and integrated into timing models and board-level signal integrity simulations. The tools used in the design of an FPGA and its integration into a PCB must be "board-aware"—able to take into account properties of the board routing and the connected devices on the board.

The Intel 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 Intel Quartus Prime software:

- I/O timing with a default or user-specified capacitive load and no signal integrity analysis (default)
- The Intel Quartus Prime **Enable Advanced I/O Timing** option utilizing a user-defined board trace model to produce enhanced timing reports from accurate "board-aware" simulation models
- 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. I/O timing reports from the Intel Quartus Prime Timing Analyzer or the Intel Quartus Prime Classic Timing Analyzer are generated based only on point-to-point delays within the I/O buffer and assume the presence of the capacitive test load with no other details about the board specified. The default size of the load is based on the I/O standard selected for the pin. Timing is measured to the FPGA pin with no signal integrity analysis details.

The **Enable Advanced I/O Timing** option expands the details in I/O timing reports by taking board topology and termination components into account. A complete point-to-point board trace model is defined and accounted for in the timing analysis. This ability to define a board trace model is an example of how the Intel Quartus Prime software is "board-aware."

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

In this case, timing and signal integrity metrics between the I/O buffer and the defined far end load are analyzed and reported in enhanced reports generated by the Intel Quartus Prime Timing Analyzer.

The information about signal integrity in this chapter refers to board-level signal integrity based on I/O buffer configuration and board parameters, not simultaneous switching noise (SSN), also known as ground bounce or V_{CC} sag. 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 can cause temporary glitches in the specified level of ground or V_{CC} for the device.

This chapter is intended for FPGA and board designers and includes details about the concepts and steps involved in getting designs simulated and how to adjust designs to improve board-level timing and signal integrity. Also included is information about how to create accurate models from the Intel Quartus Prime software and how to use those models in simulation software.

The information in this chapter is meant for those who are familiar with the Intel Quartus Prime software and basic concepts of signal integrity and the design techniques and components in good PCB design. Finally, you should know how to set up simulations and use your selected third-party simulation tool.

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.

2.1.1. Signal Integrity Simulations with HSPICE and IBIS Models

The Intel 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. IBIS models of the FPGA I/O buffers are also created easily with the Intel Quartus Prime IBIS Writer.

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 them, such as the Mentor Graphics* HyperLynx* software. 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.

The I/O's IBIS and HSPICE model creation available in the Intel Quartus Prime software can help prevent problems before a costly board respin is required. In general, creating and running accurate simulations is difficult and time consuming. The tools in the Intel Quartus Prime software automate the I/O model setup and creation process by configuring the models specifically for your design. With these tools, you can set up and run accurate simulations quickly and acquire data that helps guide your 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

2.2. I/O Model Selection: IBIS or HSPICE

The Intel 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 Intel 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 2. 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.

For more information about IBIS files created by the Intel Quartus Prime IBIS Writer and IBIS files in general, as well as links to websites with detailed information, refer to *AN 283: Simulating Intel Devices with IBIS Models*.

Related Information

AN 283: Simulating Intel Devices with IBIS Models

2.3. FPGA to Board Signal Integrity Analysis Flow

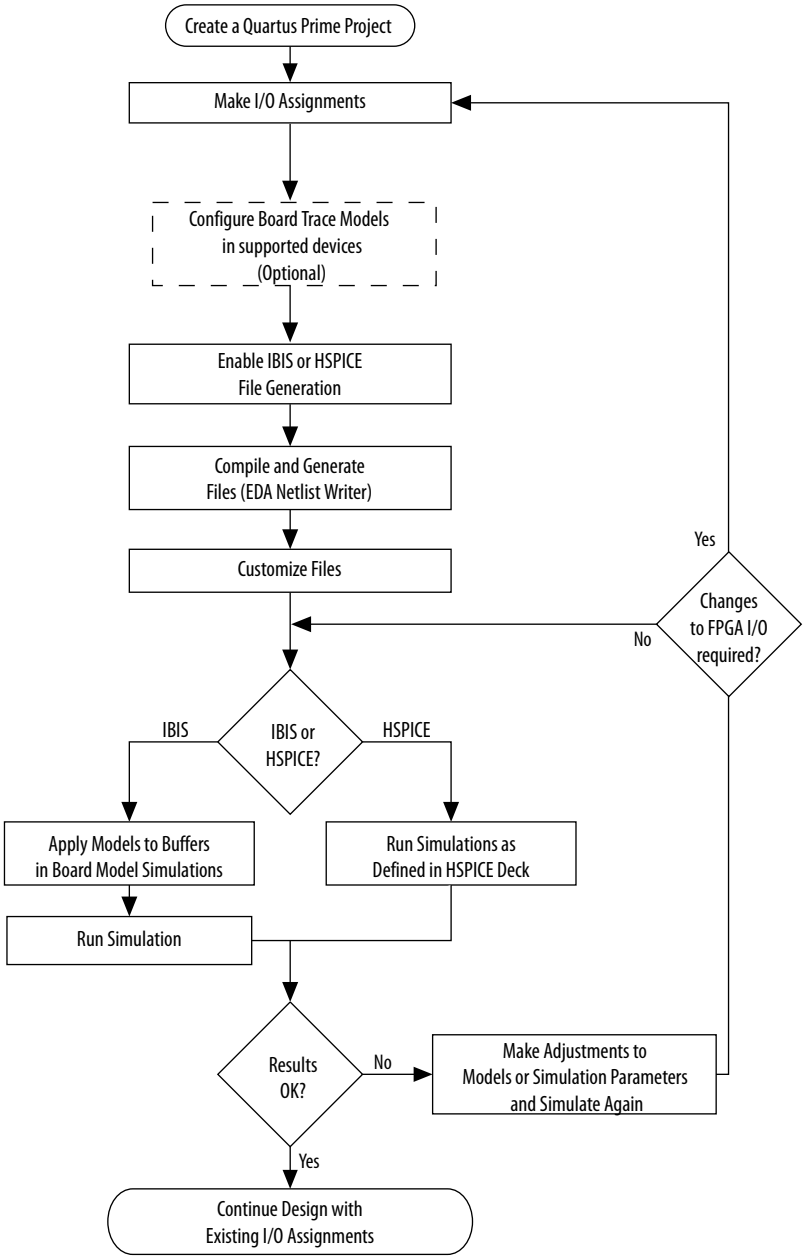
Board signal integrity analysis can take place at any point in the FPGA design process and is often performed before and after board layout. If it is performed early in the process as part of a pre-PCB layout analysis, the models used for simulations can be more generic.

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 Intel 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 11. 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.

2.3.1. 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, in the **Settings** dialog box, in the **Timing Analyzer** page, turn on the **Enable Advanced I/O Timing** option and configure 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 either in the Board Trace Model view of the Pin Planner, or click **SettingsDeviceDevice and Pin Options**.

The Intel Quartus Prime software can generate IBIS models and HSPICE decks without having to configure a board trace model with the **Enable Advanced I/O Timing** option. In fact, 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 Intel 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.

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.

2.3.2. Output File Generation

IBIS and HSPICE model files are not generated by the Intel Quartus Prime software by default. To generate or update the files automatically during each project compilation, select the type of file to generate and a location where to save the file in the project settings.

The IBIS and HSPICE Writers in the Intel Quartus Prime software are run as part of the EDA Netlist Writer during normal project compilation. If either writer is turned on in the project settings, IBIS or HSPICE files are created and stored in the specified location. For IBIS, a single file is generated containing information about all assigned pins. HSPICE file generation creates separate files for each assigned pin. You can run the EDA Netlist Writer separately from a full compilation in the Intel Quartus Prime software or at the command line.

Note: You must fully compile the project or perform I/O Assignment Analysis at least once for the IBIS and HSPICE Writers to have information about the I/O assignments and settings in the design.

2.3.3. Customize the Output Files

The files generated by either the IBIS or HSPICE Writer are text files that you can edit and customize easily for design or experimentation purposes.

IBIS files downloaded from the Altera website must be customized with the correct RLC values for the specific device package you have selected for your design. IBIS files generated by the IBIS Writer do not require this customization because they are configured automatically with the RLC values for your selected device. HSPICE decks require modification to include a detailed description of your board. With **Enable Advanced I/O Timing** turned on and a board trace model defined in the Intel Quartus Prime software, generated HSPICE decks automatically include that model's parameters. However, Intel recommends that you replace that model with a more detailed model that describes your board design more accurately. A default simulation included in the generated HSPICE decks measures delay between the FPGA and the far-end device. 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.

2.3.4. 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 Intel Quartus Prime software must take the change in the size of the load between the IBIS delay and the Intel Quartus Prime t_{CO} measurement into account. Warning messages during compilation alert you to this change.

2.3.5. 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 Intel 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 Intel Quartus Prime software based on the changes you have made, rerun the simulations to check whether your changes corrected the problem.

2.4. 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.

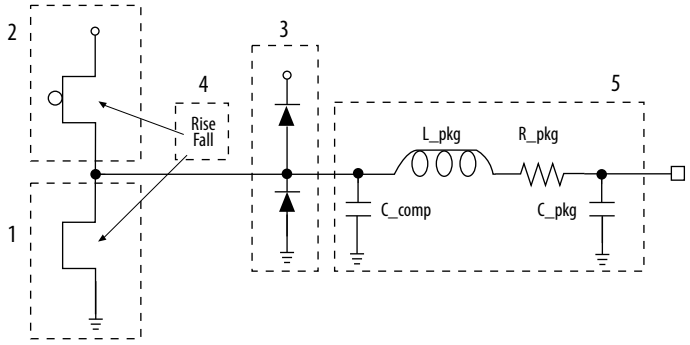
Because of their behavioral nature, IBIS models do not have to include any information about the internal circuit design of the I/O buffer. Most component manufacturers, including Intel, provide IBIS models for free download and use in signal integrity analysis simulation tools. You can download generic device family IBIS models from the Altera website for early design simulation or use the IBIS Writer to create custom IBIS models for your existing design.

2.4.1. 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.

Figure 12. Five Basic Elements of an I/O Buffer in IBIS Models



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.

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.

2.4.2. Creating Accurate IBIS Models

There are two methods to obtain Intel device IBIS files for your board-level signal integrity simulations. You can download generic IBIS models from the Altera website. You can also use the IBIS writer in the Intel Quartus Prime software to create design-specific models.

The IBIS file generated by the Intel Quartus Prime software contains models of both input and output termination, and is supported for IBIS model versions of 4.2 and later. Arria® V , Cyclone® V , and Stratix® V device families allow the use of bidirectional I/O with dynamic on-chip termination (OCT).

Dynamic OCT is used where a signal uses a series on-chip termination during output operation and a parallel on-chip termination during input operation. Typically this is used in Altera External Memory Interface IP.

The Intel 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.

2.4.2.1. Download IBIS Models

Intel provides IBIS models for almost all FPGA and FPGA configuration devices. You can use the IBIS models from the website to perform early simulations of the I/O buffers you expect to use in your design as part of a pre-layout analysis.

Downloaded IBIS models have the RLC package values set to one particular device in each device family.

The **`.ibs`** file can be customized for your device package and can be used for any simulation. IBIS models downloaded and used for simulations in this manner are generic. They describe only a certain set of models listed for each device on the Intel IBIS Models page of the Altera website. To create customized models for your design, use the IBIS Writer as described in the next section.

To simulate your design with the model accurately, you must adjust the RLC values in 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

Intel IBIS Models
For information about whether models for your selected device are available.

2.4.2.2. Generate Custom IBIS Models with the IBIS Writer

If you have started your FPGA design and have created custom I/O assignments, you can use the Intel Quartus Prime IBIS Writer to create custom IBIS models to accurately reflect your assignments.

Examples of custom assignments include drive strength settings or the enabling of clamping diodes for ESD protection. IBIS models created with the IBIS Writer take I/O assignment settings into account.

If the **Enable Advanced I/O Timing** option is turned off, the generated **.ibs** files are based on the load value setting for each I/O standard on the **Capacitive Loading** page of the **Device and Pin Options** dialog box in the **Device** dialog box. With the **Enable Advanced I/O Timing** option turned on, IBIS models use an effective capacitive load based on settings found in the board trace model on the **Board Trace Model** page in the **Device and Pin Options** dialog box or the **Board Trace Model** view in the Pin Planner. The effective capacitive load is based on the sum of the **Near capacitance**, **Transmission line distributed capacitance**, and the **Far capacitance** settings in the board trace model. Resistance values and transmission line inductance values are ignored.

Note: If you made any changes from the default load settings, the delay in the generated IBIS model cannot safely be added to the Intel Quartus Prime t_{CO} measurement to account for the double counting problem. This is because the load values between the two delay measurements do not match. When this happens, the Intel Quartus Prime software displays warning messages when the EDA Netlist Writer runs to remind you about the load value mismatch.

Related Information

- Intel IBIS models
- Generating IBIS Output Files with the Intel Quartus Prime Software
  In *Intel Quartus Prime Help*
- AN 283: Simulating Intel Devices with IBIS Models

2.4.3. Design Simulation Using the Mentor Graphics HyperLynx Software

You must integrate IBIS models downloaded from the Altera website or created with the Intel Quartus Prime IBIS Writer into board design simulations to accurately model timing and signal integrity.

The HyperLynx software from Mentor Graphics is one of the most popular tools for design simulation. The HyperLynx software makes it easy to integrate IBIS models into simulations.

The HyperLynx software is a PCB analysis and simulation tool for high-speed designs, consisting of two products, LineSim and BoardSim.

LineSim is an early simulation tool. Before any board routing takes place, you use LineSim to simulate "what if" scenarios that assist in creating routing rules and defining board parameters.

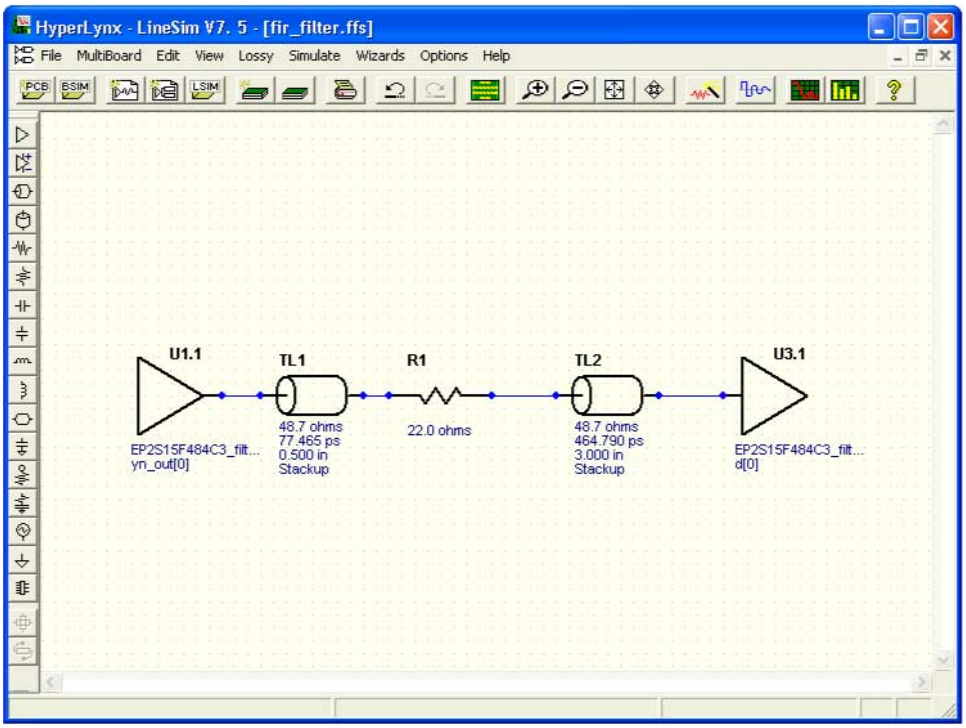
BoardSim is a post-layout tool that you use to analyze existing board routing. You select one or more nets from a board layout file and BoardSim simulates those nets in a manner similar to LineSim. With board and routing parameters, and surrounding signal routing known, highly accurate simulations of the final fabricated PCB are possible.

This 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 applies 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 13. 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 Intel 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 Intel 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 Mentor Graphics webpage.

Related Information

www.mentor.com

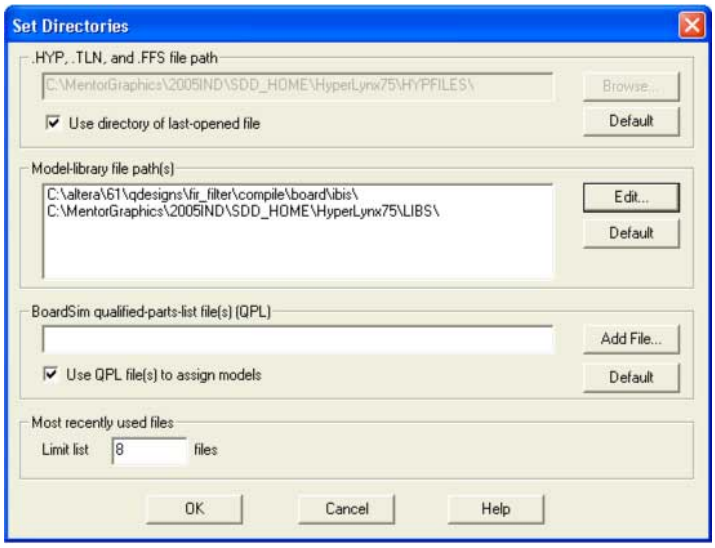
2.4.4. Configuring LineSim to Use Intel IBIS Models

You must configure LineSim to find and use the downloaded or generated IBIS models for your design. To do this, add the location of your `.ibs` file or files to the LineSim Model Library search path. Then you apply a selected model to a buffer in your schematic.

To add the Intel 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:

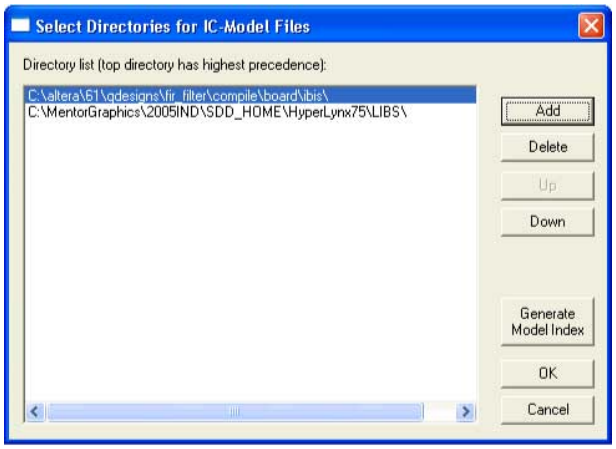
1. From the Options menu, click **Directories**. The **Set Directories** dialog box appears. The **Model-library file path(s)** list displays the order in which LineSim searches file directories for model files.

Figure 14. 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 15. LineSim Select Directories Dialog Box



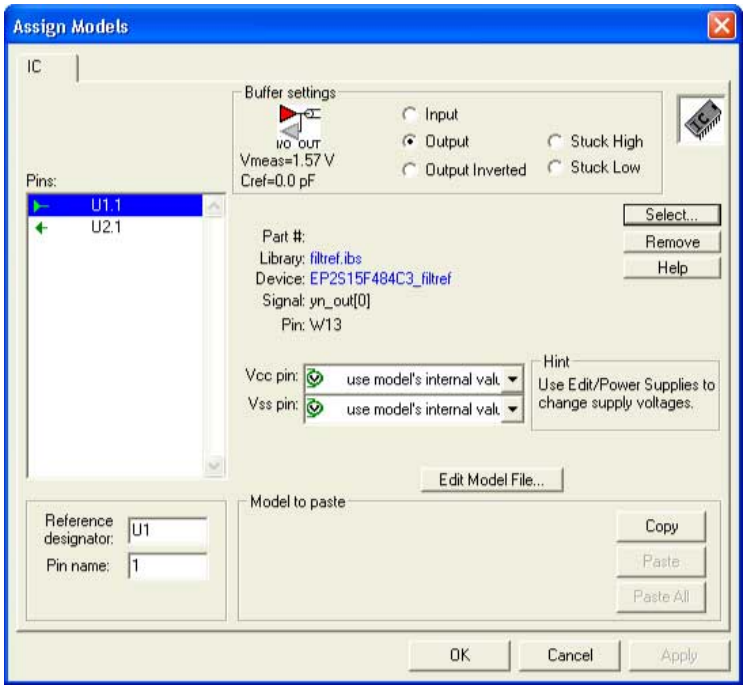
3. Click **Add**
4. Browse to the default IBIS model location, *<project directory>*`/board/ibis`. Click **OK**.
5. Click **Up** to move the IBIS model directory to the top of the list. Click **Generate Model Index** to update LineSim's model database with the models found in the added directory.
6. 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**.

2.4.5. 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:

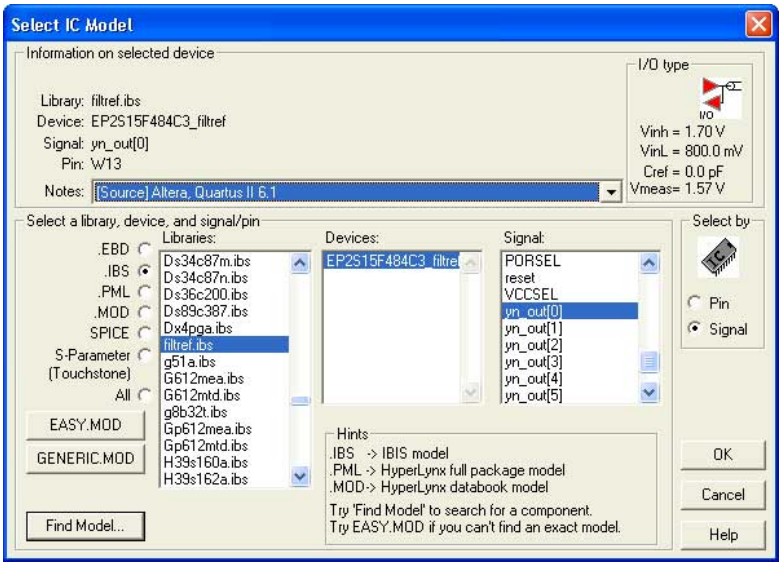
1. Double-click a buffer symbol in your schematic to open the **Assign Models** dialog box. You can also click **Assign Models** from the buffer symbol's right-click menu.

Figure 16. 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 17. LineSim Select IC Model Dialog Box



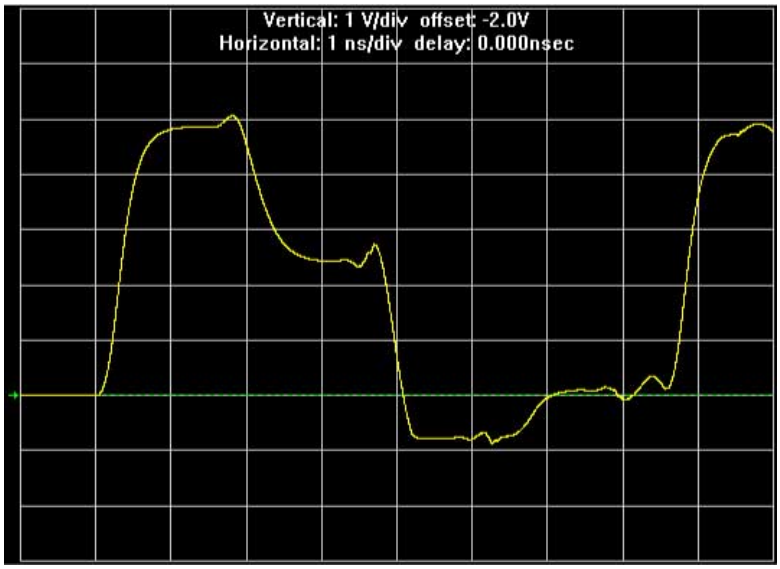
4. To filter the list of available libraries to display only IBIS models, select **.IBS**. Scroll through the **Libraries** list, and click the name of the library for your design. By default, this is <*project name*>**.ibs**.
5. The device for your design should be selected as the only item in the **Devices** 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.

2.4.6. Running and Interpreting LineSim Simulations

You can run any simulation and make adjustments to the I/O assignments or simulation parameters as required.

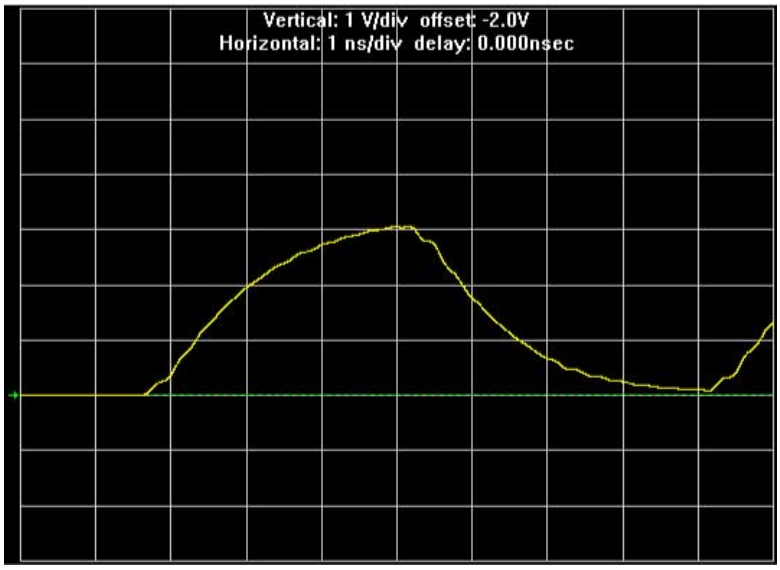
For example, if you see too much overshoot in the simulated signal at the destination buffer after running a simulation, you can adjust the drive strength I/O assignment setting to a lower value. Regenerate the `.ibs` file, and run the simulation again to verify whether the change fixes the problem.

Figure 18. 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 19. 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

2.5. 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. For devices that support advanced I/O timing, when **Enable Advanced I/O Timing** is turned on, the HSPICE decks generated by the Intel 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 Intel 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 41

2.5.1. Supported Devices and Signaling

The HSPICE Writer in the Intel Quartus Prime software supports Arria , Cyclone, and Stratix devices for the creation of a board trace model in the Intel 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 Intel Arria 10 devices, you may need to download the Encrypted HSPICE model from the Altera 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 Devices
  For more information about the Encrypted HSPICE model.

2.5.2. Accessing HSPICE Simulation Kits

You can access the available HSPICE models with the Intel Quartus Prime software's HSPICE Writer tool and also at the Spice Models for Intel Devices web page.

The Intel 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 Devices
For more information about downloadable HSPICE models.

2.5.3. The Double Counting Problem in HSPICE Simulations

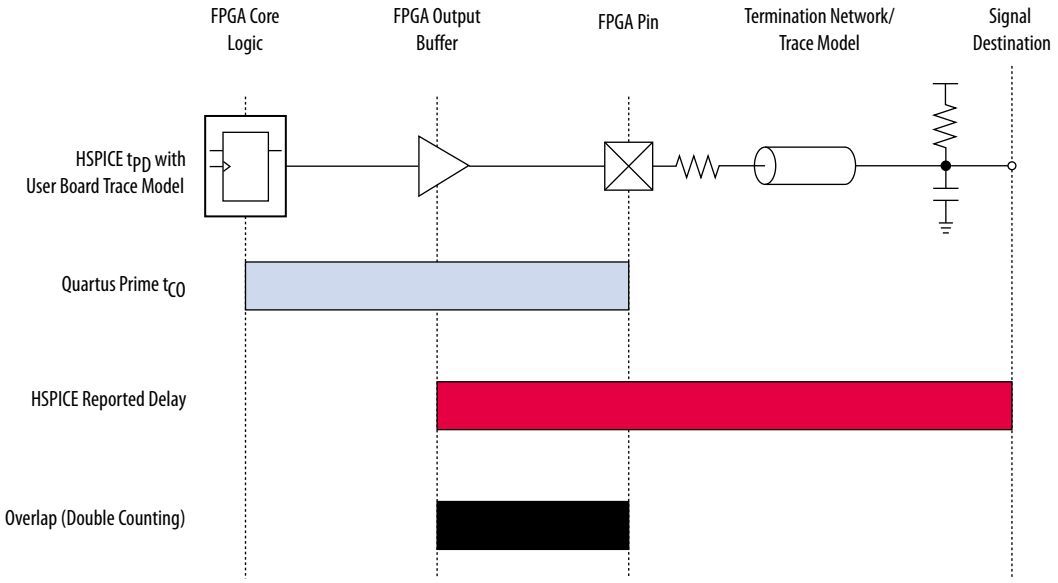
Simulating I/Os using accurate models is extremely helpful for finding and fixing FPGA I/O timing and board signal integrity issues before any boards are built. However, the usefulness of such simulations is directly related to the accuracy of the models used and whether the simulations are set up and 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 Intel 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.

2.5.3.1. Defining the Double Counting Problem

The double counting problem is inherent to the difference between the method to analyze output timing in the Intel Quartus Prime software versus the method HSPICE models use. The timing analyzer tools in the Intel 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 20. 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.

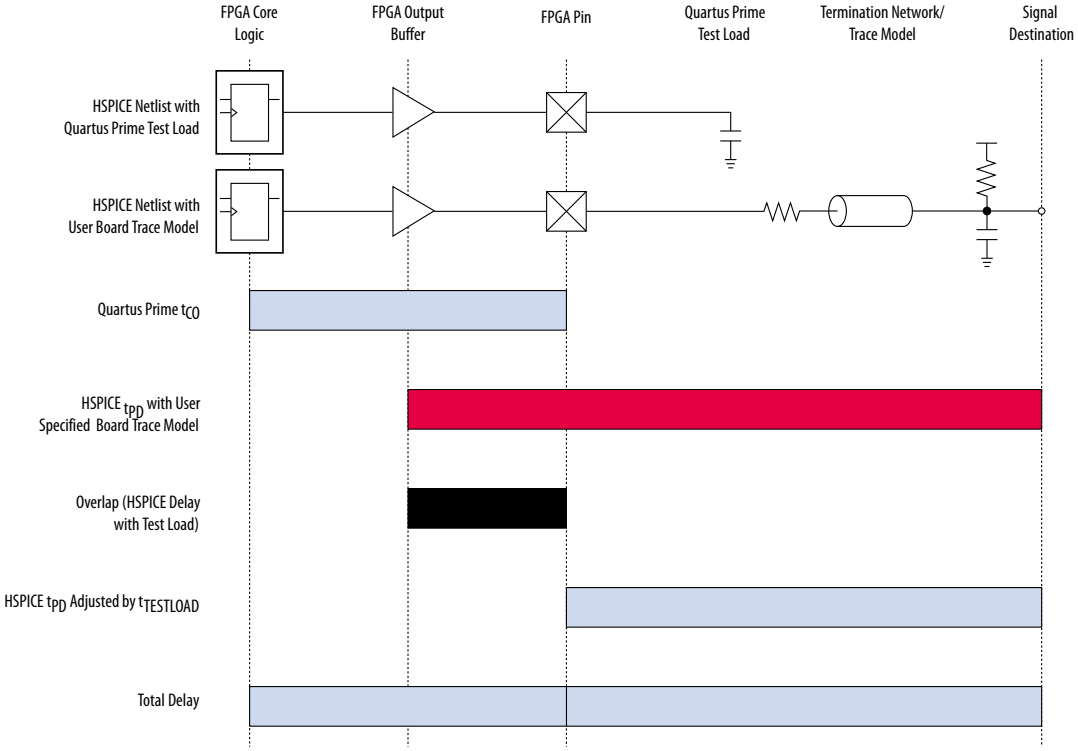
One approach to fix the problem is subtracting the overlap between t_{CO} and t_{PD} to account for the double count. However, this adjustment is not accurate, because each measurement considers a different load.

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.

2.5.3.2. 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 Intel Quartus Prime software for the t_{CO} measurement.

Figure 21. 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 Intel Quartus Prime software are designed to account for the double-counting problem based on this calculation automatically.

2.5.4. HSPICE Writer Tool Flow

This section includes information to help you get started using the Intel Quartus Prime software HSPICE Writer tool. The information in this section assumes you have a basic knowledge of the standard Intel Quartus Prime software design flow, such as project and assignment creation, compilation, and timing analysis.

2.5.4.1. 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 Intel 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.

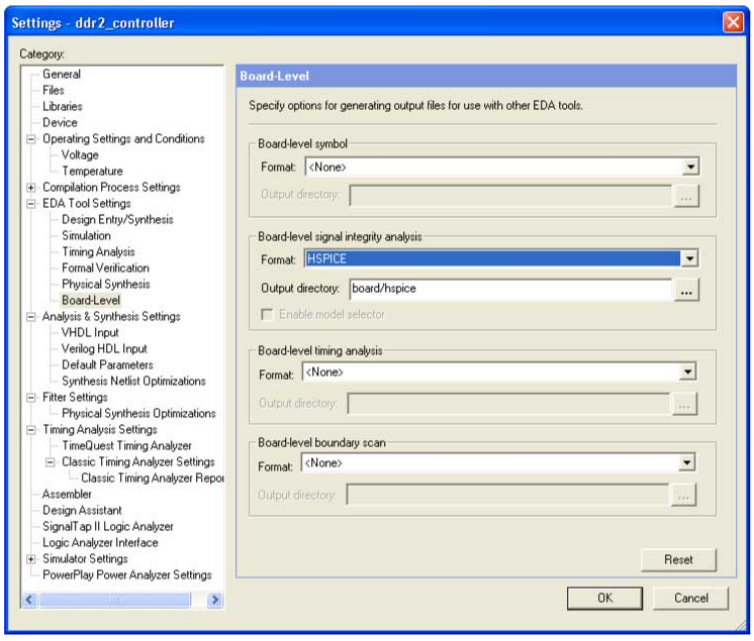
The Intel Quartus Prime software provides multiple methods for creating these assignments:

- Using the Pin Planner
- Using the assignment editor
- Manually editing the **`.qsf`** file
- By making assignments in a scripted Intel Quartus Prime flow using Tcl

2.5.4.2. Enabling HSPICE Writer

You must enable the HSPICE Writer in the **Settings** dialog box of the Intel Quartus Prime software to generate the HSPICE decks from the Intel Quartus Prime software.

Figure 22. EDA Tool Settings: Board Level Options Dialog Box



2.5.4.3. 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**.

2.5.4.4. 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 Intel 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 54
- Sample Input for I/O HSPICE Simulation Deck on page 50

2.5.4.5. Invoking HSPICE Writer

After HSPICE Writer is enabled, the HSPICE simulation files are generated automatically each time the project is completely compiled. The Intel 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.

2.5.4.6. 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.

2.5.4.7. 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

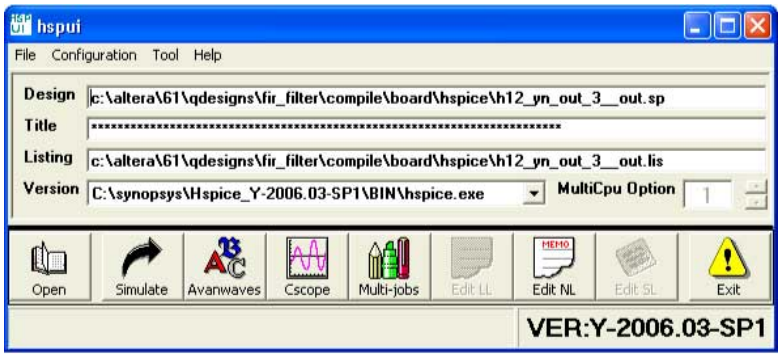
```
* Sample source stimulus placeholder
*   - Replace this with your I/O driver model
```

Replace the sample stimulus model with a model for the device that drives the FPGA.

2.5.5. 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 23. HSPICE User Interface Window



Click **Open** and browse to the location of the HSPICE model files generated by the Intel 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.

2.5.6. 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 Intel 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 54

2.5.7. 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 Intel Quartus Prime software reported T_H and T_{SU} input timing numbers.

2.5.8. 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.

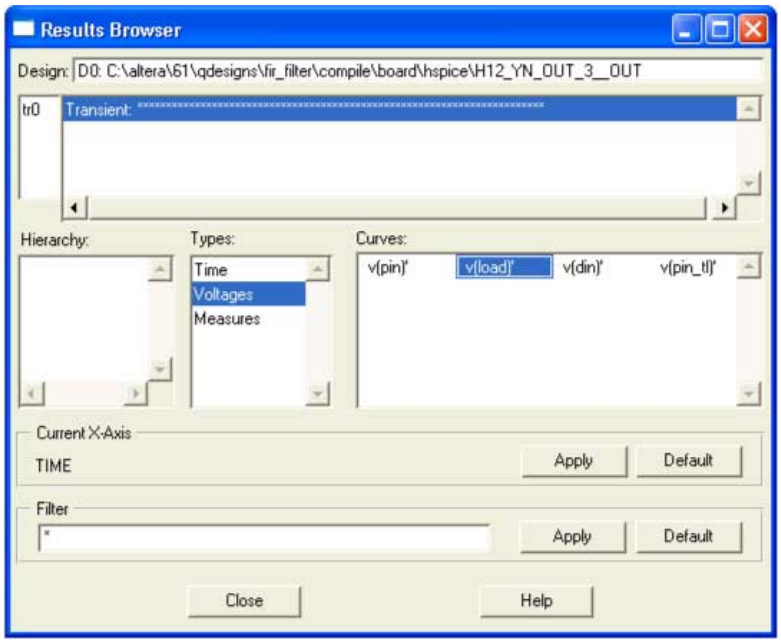
These measurements are found in the `.lis` file and named `tpd_rise` and `tpd_fall`. For output simulations, these values are already adjusted for the double count. To determine the complete delay from the FPGA logic to the load pin, add either of these measurements to the Intel Quartus Prime t_{CO} delay. For input simulations, add either of these measurements to the Intel 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.

2.5.9. 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 24. 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

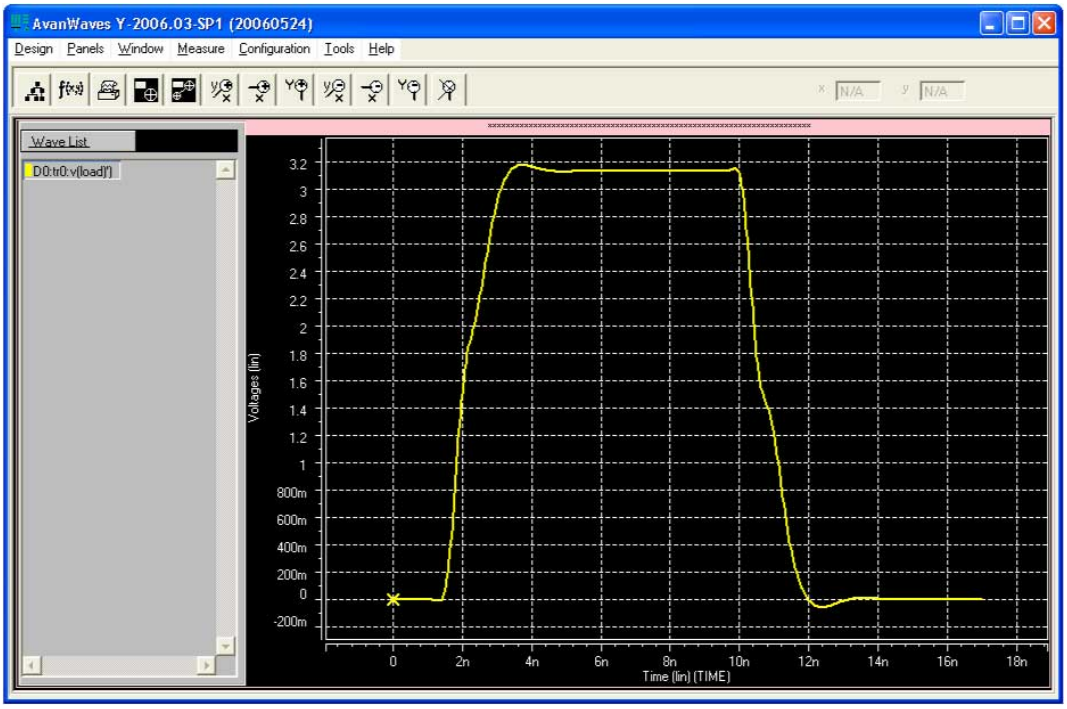
description to select which simulation to view. By default, the descriptions are derived from the first line of the HSPICE file, so the description might appear as a line of asterisks.

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.

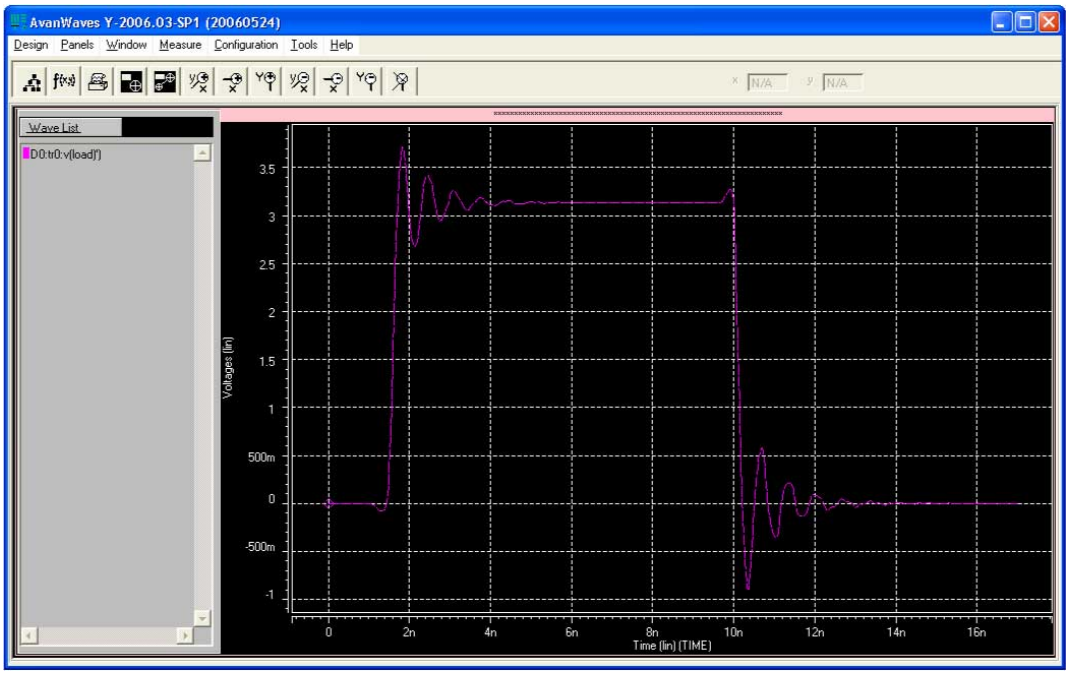
Figure 25. AvanWaves Waveform Viewer



2.5.10. Making Design Adjustments Based on HSPICE Simulations

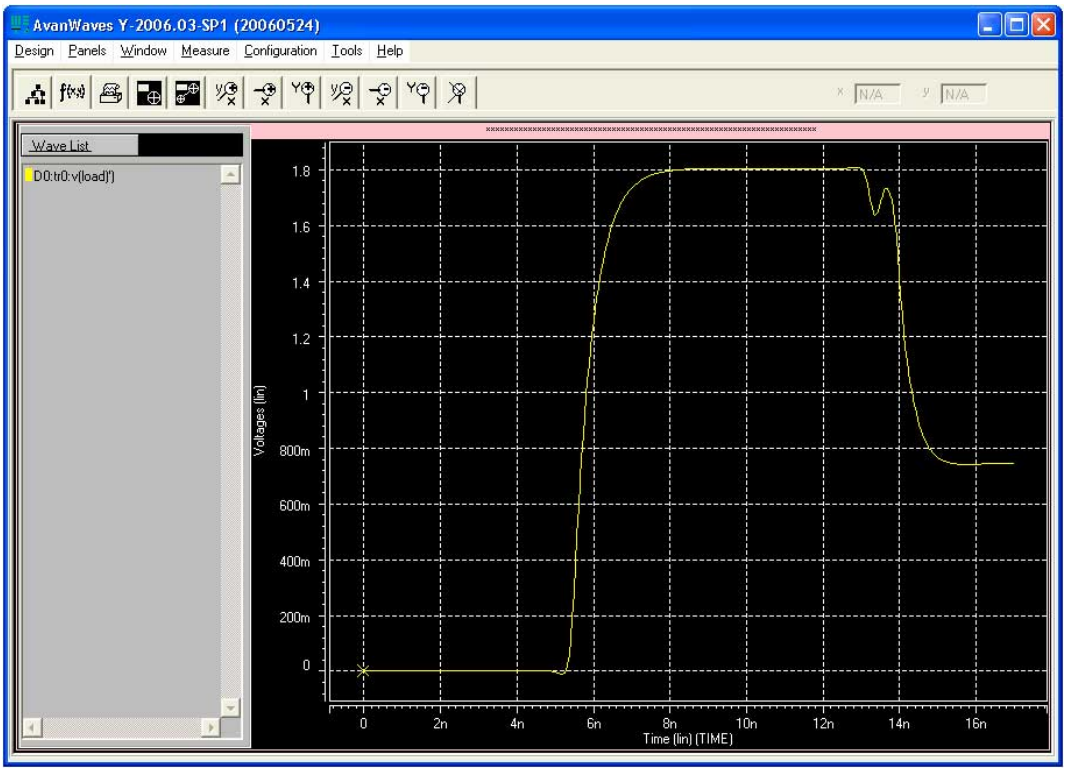
Based on the results of your simulations, you can make adjustments to the I/O assignments or simulation parameters if required. For example, after you run a simulation and see overshoot or ringing in the simulated signal at the destination buffer, you can adjust the drive strength I/O assignment setting to a lower value. Regenerate the HSPICE deck, and run the simulation again to verify that the change fixed the problem.

Figure 26. 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 Intel 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 27. 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

2.5.11. 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.

2.5.11.1. 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 Intel 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 Intel 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:
```

2.5.11.2. 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
```

2.5.11.3. 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.

2.5.11.4. 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.

2.5.11.5. 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’
```

2.5.11.6. 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
```

2.5.11.7. 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
```

2.5.11.8. 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
```

2.5.11.9. 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)
```

2.5.11.10. 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
```

2.5.12. 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.

2.5.12.1. 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 Intel 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 Intel 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:
```

2.5.12.2. 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.

2.5.12.3. 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.

2.5.12.4. 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 Intel Quartus Prime software.

2.5.12.5. 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'
```

2.5.12.6. 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
```

2.5.12.7. 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
```

2.5.12.8. 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
```

2.5.12.9. 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
```

2.5.12.10. 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 Intel Quartus Prime software test load. The simulated delay of this second buffer can be interpreted as the amount of double-counting between the Intel 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 41

2.5.12.11. 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'
```

2.5.13. 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.

2.5.13.1. 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 dependant upon how the Intel Quartus Prime software calculates delays and is not based on which PVT corner is being simulated. By default, the Intel 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 40

2.5.13.2. 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 Intel 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 Intel 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.

2.5.13.3. 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.

2.5.13.4. 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 40

2.6. Document Revision History

Table 3. Document Revision History

Date	Version	Changes
2017.11.06	17.1.0	• Reorganized chapter introduction.
2016.10.31	16.1.0	• Corrected statement about timing simulation and double counting.
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	Updated format.
December 2010	10.0.1	Template update.
July 2010	10.0.0	Updated device support.
November 2009	9.1.0	No change to content.
March 2009	9.0.0	• Was volume 3, chapter 12 in the 8.1.0 release. • No change to content.
November 2008	8.1.0	• 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	8.0.0	• 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. • 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.

3. Mentor Graphics PCB Design Tools Support

You can integrate the Mentor Graphics ® I/O Designer or DxDesigner PCB design tools into the Intel Quartus Prime design flow. This combination provides a complete FPGA-to-board design workflow.

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.

The Mentor Graphics I/O Designer software allows you to take advantage of the full FPGA symbol design, creation, editing, and back-annotation flow supported by the Mentor Graphics tools.

This chapter covers the following topics:

- Mentor Graphics and Intel software integration flow
- Generating supporting files
- Adding Intel Quartus Prime I/O assignments to I/O Designer
- Updating assignment changes between the I/O Designer the Intel Quartus Prime software
- Generating I/O Designer symbols
- Creating DxDesigner symbols from the Intel 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 Mentor Graphics tools and pass the information back to the Intel Quartus Prime software for placement and routing. Part librarians can also benefit from this chapter by learning how to use output from the Intel Quartus Prime software to create new library parts and symbols.

The procedures in this chapter require the following software:

- The Intel Quartus Prime software version 5.1 or later
- DxDesigner software version 2004 or later
- Mentor Graphics I/O Designer software (optional)

Note: To obtain and license the Mentor Graphics tools and for product information, support, and training, refer to the Mentor Graphics website.

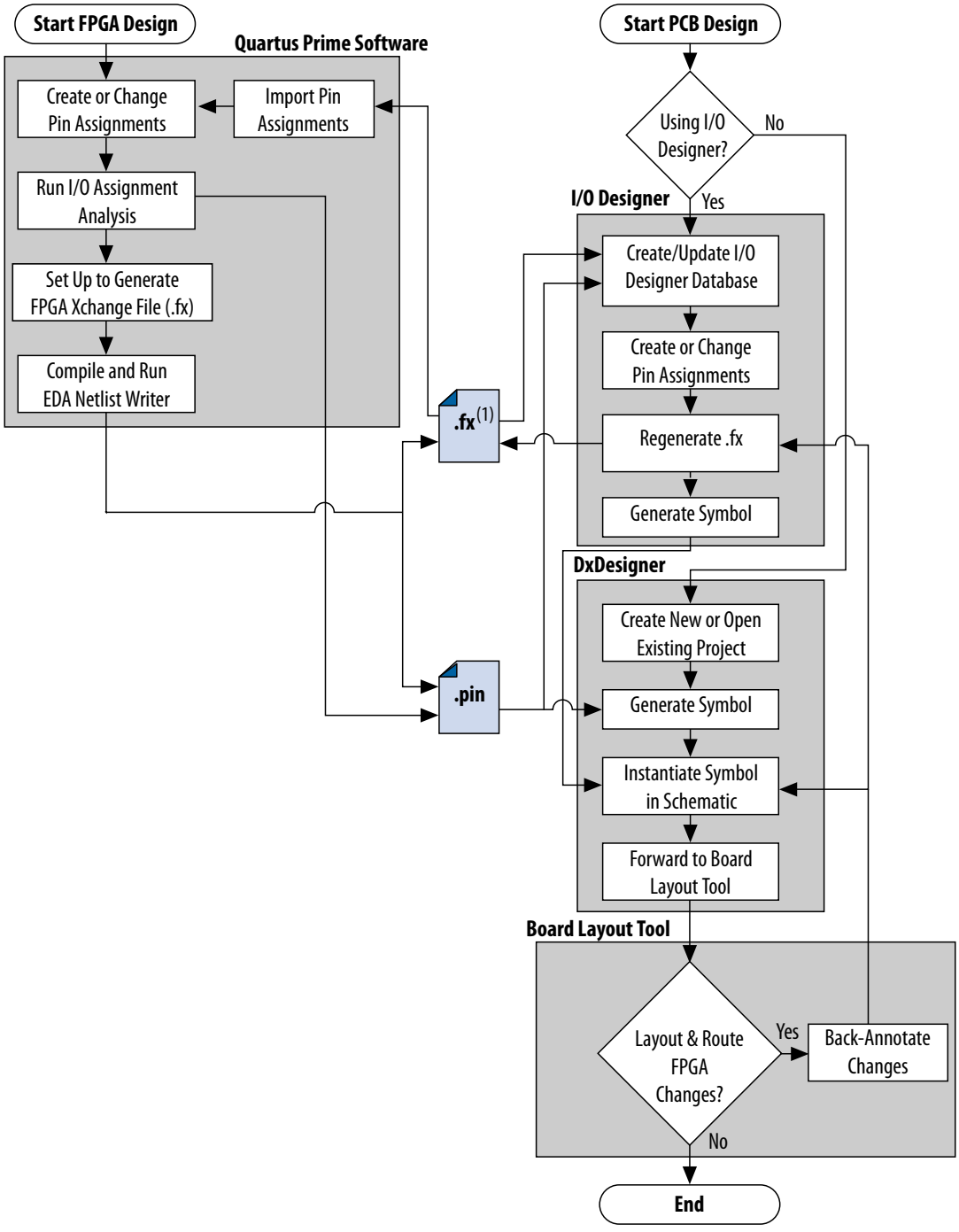
Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

3.1. FPGA-to-PCB Design Flow

You can create a design flow integrating an Intel FPGA design from the Intel Quartus Prime software, and a circuit schematic in the DxDesigner software.

Figure 28. Design Flow with and Without the I/O Designer Software



Note: The Intel Quartus Prime software generates the **.fx** in the output directory you specify in the **Board-Level** page of the **Settings** dialog box. However, the Intel Quartus Prime software and the I/O Designer software can import pin assignments from an **.fx** located in any directory. Use a backup **.fx** to prevent overwriting existing assignments or importing invalid assignments.

To integrate the I/O Designer into your design flow, follow these steps:

1. In the Intel Quartus Prime software, click **Assignments ➤ Settings ➤ EDA Tool Settings ➤ Board-Level** to specify settings for **.fx** symbol file generation.
2. Compile your design to generate the **.fx** and Pin-Out File (**.pin**) in the Intel Quartus Prime project directory.
3. Create a board design with the DxDesigner software and the I/O Designer software by performing the following steps:
   a. Create a new I/O Designer database based on the **.fx** and the **.pin** files.
   b. In the I/O Designer software, make adjustments to signal and pin assignments.
   c. Regenerate the **.fx** in the I/O Designer software to export the I/O Designer software changes to the Intel Quartus Prime software.
   d. Generate a single or fractured symbol for use in the DxDesigner software.
   e. Add the symbol to the **sym** directory of a DxDesigner project, or specify a new DxDesigner project with the new symbol.
   f. Instantiate the symbol in your DxDesigner schematic and export the design to the board layout tool.
   g. Back-annotate pin changes created in the board layout tool to the DxDesigner software and back to the I/O Designer software and the Intel Quartus Prime software.
4. Create a board design with the DxDesigner software without the I/O Designer software by performing the following steps:
   a. Create a new DxBoardLink symbol with the **Symbol** wizard and reference the **.pin** from the Intel Quartus Prime software in an existing DxDesigner project.
   b. Instantiate the symbol in your DxDesigner schematic and export the design to a board layout tool.

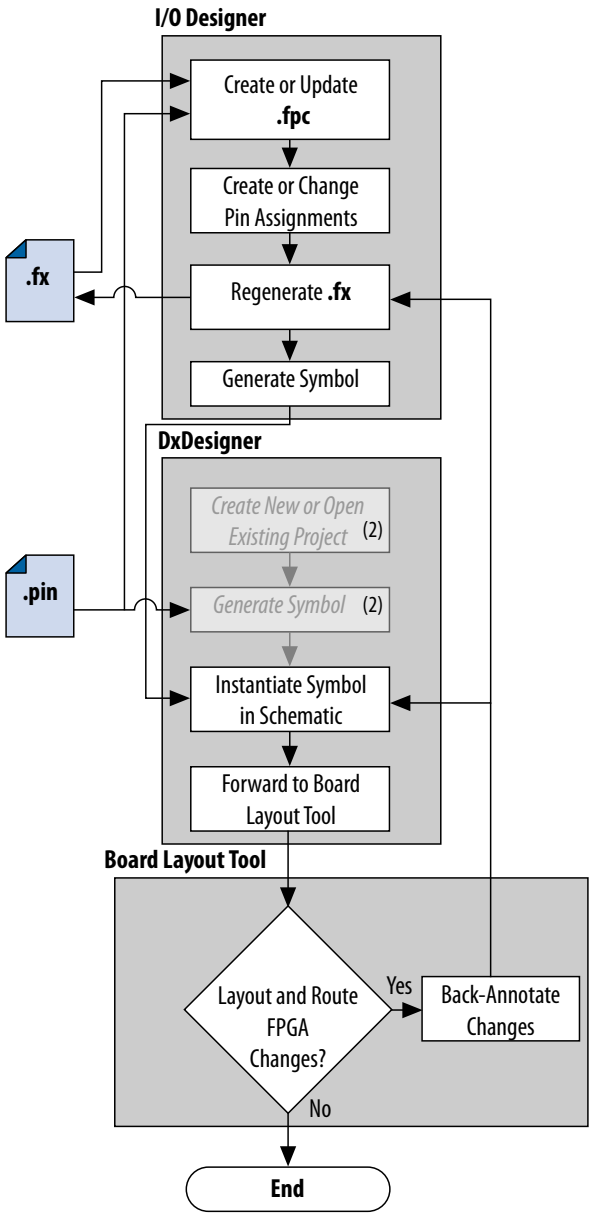
Note: You can update these symbols with design changes with or without the I/O Designer software. If you use the Mentor Graphics I/O Designer software and you change symbols with the DxDesigner software, you must reimport the symbols into I/O Designer to avoid overwriting your symbol changes.

3.2. Integrating with I/O Designer

You can integrate the Mentor Graphics I/O Designer software into the Intel Quartus Prime design flow. Pin and signal assignment changes can be made anywhere in the design flow with either the Intel Quartus Prime Pin Planner or the I/O Designer software. The I/O Designer software facilitates moving these changes, as well as synthesis, placement, and routing changes, between the Intel Quartus Prime software, an external synthesis tool (if used), and a schematic capture tool such as the DxDesigner software.

This section describes how to use the I/O Designer software to transfer pin and signal assignment information to and from the Intel Quartus Prime software with an **.fx**, and how to create symbols for the DxDesigner software.

Figure 29. I/O Designer Design Flow



Note: (2) DxDesigner software-specific steps in the design flow are not part of the I/O Designer flow.

3.2.1. Generating Pin Assignment Files

You transfer I/O pin assignments from the Intel Quartus Prime software to the Mentor Graphics PCB tools by generating optional **.pin** and **.fx** files during Intel Quartus Prime compilation. These files contain pin assignment information for use in other tools.

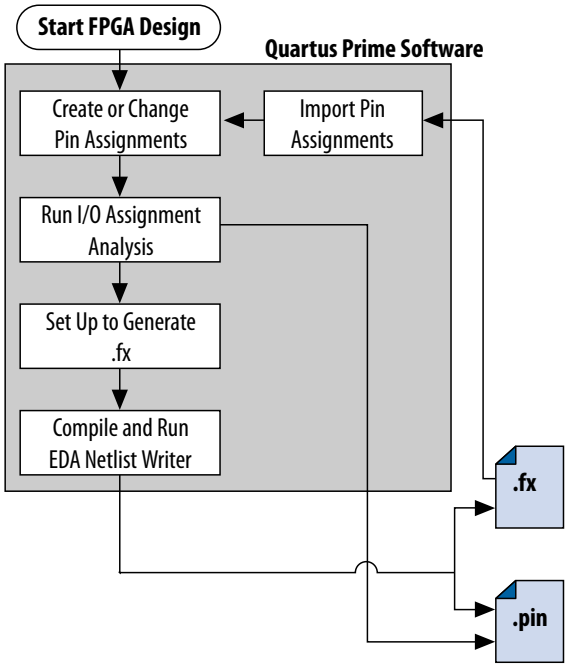
Click **Assignments ➤ Settings ➤ Board-Level** to specify settings for optional PCB tool file generation. Click **Processing ➤ Start Compilation** to compile the design to generate the file(s) in the project directory.

The Intel Quartus Prime-generated **.pin** contains the I/O pin name, number, location, direction, and I/O standard for all used and unused pins in the design. Click **Assignments ➤ Pin Planner** to modify I/O pin assignments. You cannot import pin assignment changes from a Mentor Graphics **.pin** into the Intel Quartus Prime software.

The **.fx** is an input or output of either the Intel Quartus Prime or I/O Designer software. You can generate an **.fx** in the Intel Quartus Prime software for symbol generation in the Mentor Graphics I/O Designer software. an Intel Quartus Prime **.fx** contains the pin name, number, location, direction, I/O standard, drive strength, termination, slew rate, IOB delay, and differential pins. An I/O Designer **.fx** additionally includes information about unused pins and pin set groups.

The I/O Designer software can also read from or update an Intel Quartus Prime Settings File (**.qsf**). You can use the **.qsf** in the same way as use of the **.fx**, but pin swap group information does not transfer between I/O Designer and the Intel Quartus Prime software. Use the **.fx** rather than the **.qsf** for transferring I/O assignment information.

Figure 30. Generating .pin and .fx files



3.2.2. I/O Designer Settings

You can directly export I/O Designer symbols to the DxDesigner software. To set options for integrating I/O Designer with Dx Designer, follow these steps:

1. Start the I/O Designer software.
2. Click **Tools ➤ Prefences**.
3. Click **Paths**, and then double-click the **DxDesigner executable file** path field to select the location of the DxDesigner application.
4. Click **Apply**.
5. Click **Symbol Editor**, and then click **Export**. In the Export type menu, under **General**, select **DxDesigner/PADS-Designer**.
6. Click **Apply**, and then click **OK**.
7. Click **File ➤ Properties**.
8. Click the **PCB Flow** tab, and then click **Path to a DxDesigner project directory**.
9. Click **OK**.
   If you do not have a new DxDesigner project in the Database wizard and a DxDesigner project, you must create a new database with the DxDesigner software, and then specify the project location in I/O Designer.

3.2.3. Transferring I/O Assignments

You can transfer Intel Quartus Prime signal and pin assignments contained in **.pin** and **.fx** files into an I/O Designer database. Use the I/O Designer Database Wizard to create a new database incorporating the **.fx** and **.pin** files. You can also create a new, empty database and manually add the assignment information. If there is no available signal or pin assignment information, you can create an empty database containing only a selection of the target device. This technique is useful if you know the signals in your design and the pins you want to assign. You can subsequently transfer this information to the Intel Quartus Prime software for placement and routing.

You may create a very simple I/O Designer database that includes only the **.pin** or **.fx** file information. However, when using only a **.pin**, you cannot import I/O assignment changes from I/O Designer back into the Intel Quartus Prime software without also generating an **.fx**. If your I/O Designer database includes only **.fx** file information, the database may not contain all the available I/O assignment information. The Intel Quartus Prime **.fx** file only lists assigned pins. The **.pin** lists all assigned and unassigned device pins. Use both the **.pin** and the **.fx** to produce the most complete set of I/O Designer database information.

To create a new I/O Designer database using the Database wizard, follow these steps;

1. Start the I/O Designer software. The **Welcome to I/O Designer** dialog box appears. Select **Wizard to create new database** and click **OK.**
   If the **Welcome to I/O Designer** dialog box does not appear, you can access the wizard through the menu. To access the wizard, click **File ➤ Database Wizard**.
2. Click **Next**. The **Define HDL source file** page appears
   If no HDL files are available, or if the **.fx** contains your signal and pin assignments, you can skip Step 3 and proceed to Step 4.
3. If your design includes a Verilog HDL or VHDL file, you can add a top-level Verilog HDL or VHDL file in the I/O Designer software. Adding a file allows you to create functional blocks or get signal names from your design. You must create all physical pin assignments in I/O Designer if you are not using an **.fx** or a **.pin**.
   Click **Next**. The **Database Name** page appears.

4. In the **Database Name** page, type your database file name. Click **Next**. The Database Location window appears.
5. Add a path to the new or an existing database in the **Location** field, or browse to a database location. Click **Next**. The **FPGA flow** page appears.
6. In the Vendor menu, click **Altera**.
7. In the Tool/Library menu, click **Intel Quartus Prime <version>** to select your version of the Intel Quartus Prime software.

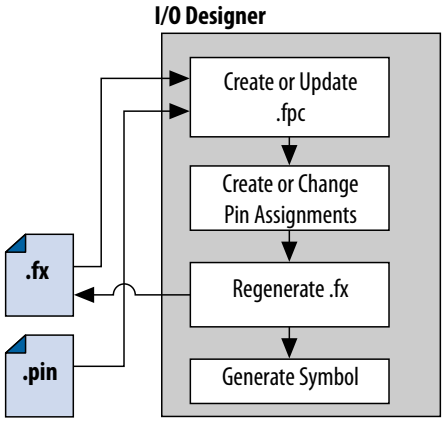
   Note: The Intel Quartus Prime software version listed may not match your actual software version. If your version is not listed, select the latest version. If your target device is not available, the device may not yet be supported by the I/O Designer software.
8. Select the appropriate device family, device, package, and speed (if applicable), from the corresponding menus. Click **Next**. The **Place and route** page appears.
9. In the **FPGAX file name** field, type or browse to the backup copy of the **.fx** generated by the Intel Quartus Prime software.
10. In the **Pin report file name** field, type or browse to the **.pin** generated by the Intel Quartus Prime software. Click **Next**.
    You can also select a **.qsf** for update. The I/O Designer software can update the pin assignment information in the **.qsf** without affecting any other information in the file.

    Note: You can import a **.pin** without importing an **.fx**. The I/O Designer software does not generate a **.pin**. To transfer assignment information to the Intel Quartus Prime software, select an additional file and file type. Intel recommends selecting an **.fx** in addition to a **.pin** for transferring all the assignment information in the **.fx** and **.pin** files. In some versions of the I/O Designer software, the standard file picker may incorrectly look for a **.pin** instead of an **.fx**. In this case, select **All Files (*.*)** from the **Save as type** list and select the file from the list.
11. On the **Synthesis** page, specify an external synthesis tool and a synthesis constraints file for use with the tool. If you do not use an external synthesis tool, click **Next**.
12. On the **PCB Flow** page, you can select an existing schematic project or create a new project as a symbol information destination.
    - To select an existing project, select Choose existing project and click Browse after the Project Path field. The Select project dialog box appears. Select the project.
    - To create a new project, in the Select project dialog box, select Create new empty project. Type the project file name in the Name field and browse to the location where you want to save the file. Click OK.
13. If you have not specified a design tool to which you can send symbol information in the I/O Designer software, click **Advanced** in the **PCB Flow** page and select your design tool. If you select the DxDesigner software, you have the option to specify a Hierarchical Occurrence Attributes (**.oat**) file to import into the I/O Designer software. Click **Next** and then click **Finish** to create the database.Updating

3.2.4. Updating I/O Designer with Intel Quartus Prime Pin Assignments

As you fine tune your design in the Intel Quartus Prime software, changes to design logic and pin assignments are likely. You must transfer any pin assignment changes made during design iterations for correct analysis in your circuit schematic and board layout tools. You transfer Intel Quartus Prime pin assignment changes to I/O Designer by updating the **.fx** and the **.pin** files in the Intel Quartus Prime software. When you update the **.fx** or the **.pin**, the I/O Designer database imports the changes automatically when configured according to the following instructions.

Figure 31. Updating Intel Quartus Prime Pin Assignments in I/O Designer



To update the **.fx** in your selected output directory and the **.pin** in your project directory after making changes to the design, perform the following tasks:

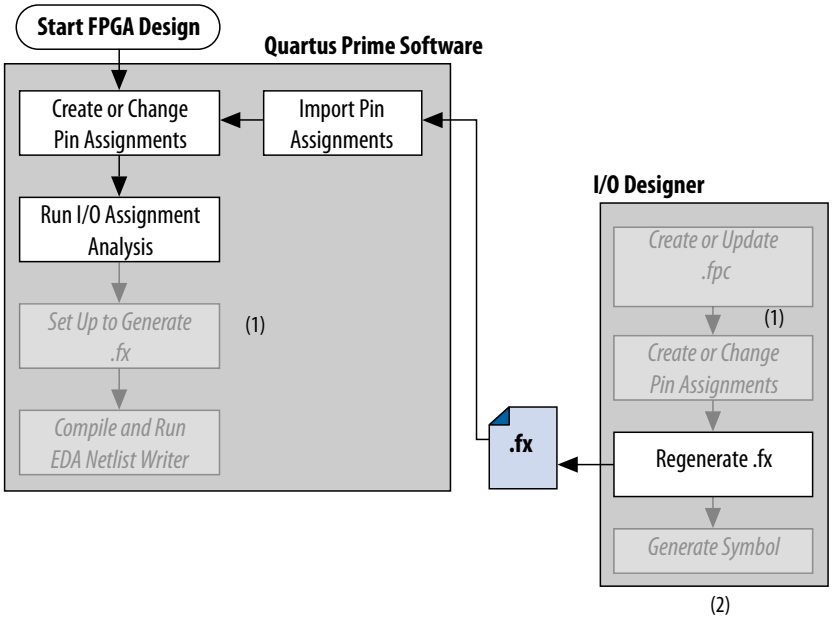
1. In the I/O Designer software, click **File ➤ Properties**.
2. Under **FPGA Xchange**, specify the **.fx** file name and location.
3. Under **Place and Route**, specify the **.pin** file name and location.
   After you have set up these file locations, the I/O Designer software monitors these files for changes. If the specified **.fx** or **.pin** is modified during design processing, three indicators flash red in the lower right corner of the I/O Designer GUI. You can click the indicators to open the **I/O Designer Update Wizard** dialog box. The **I/O Designer Update Wizard** dialog box lists the updated files in the database.
4. Make logic or pin assignment changes in your design.
5. Click **Processing ➤ Start ➤ Start I/O Assignment Analysis** to validate your latest assignment changes.
6. To preserve your changes an update the corresponding the **.fx** and **.pin** files, click **Processing ➤ Start ➤ Start EDA Netlist Writer** or **Processing ➤ Start Compilation**.

   Note: Your I/O Designer database should us a backup copy of the **.fx** generated by the Intel Quartus Prime software. Otherwise, updating the file in the Intel Quartus Prime software overwrites any changes made to the file by the I/O Designer software. If there are I/O Designer assignments in the **.fx** that you want to preserve, create a backup copy of the file before updating it in the Intel Quartus Prime software, and verify that your I/O Designer database points to the backup copy.

3.2.5. Updating Intel Quartus Prime with I/O Designer Pin Assignments

As you fine tune your board design in I/O Designer, changes to signal routing and layout are likely. You must import any routing and layout changes into the Intel Quartus Prime software for accurate place and route to match the new pin-out. The I/O Designer tool supports this flow.

Figure 32. Importing I/O Designer Pin Assignments



To import I/O Designer pin assignments, follow these steps:

1. Make pin assignment changes directly in the I/O Designer software, or the software can automatically update changes made in a board layout tool that are back-annotated to a schematic entry program such as the DxDesigner software.
2. To update the **.fx** with the changes, click **Generate ➤ FPGA Xchange File**.
3. Open your Intel Quartus Prime project.
4. Click **Assignments ➤ Import Assignments**.
5. (Optional) To preserve original assignments before import, turn on **Copy existing assignments into <project name>.qsf.bak** before importing before importing the **.fx**.
6. Select the **.fx** and click **Open**.
7. Click **OK**.

3.2.6. Generating Schematic Symbols in I/O Designer

Circuit board schematic creation is one of the first tasks required in the design of a new PCB. You can use the I/O Designer software to generate schematic symbols for your Intel Quartus Prime FPGA design for use in the DXDesigner schematic entry tools. The I/O Designer software can generate symbols for use in various Mentor

Graphics schematic entry tools, and can import changes back-annotated by board layout tools to update the database and update the Intel Quartus Prime software with the **.fx**

Most FPGA devices contain hundreds of pins, requiring large schematic symbols that may not fit on a single schematic page. Symbol designs in the I/O Designer software can be split or fractured into various functional blocks, allowing multiple part fractures on the same schematic page or across multiple pages. In the DxDesigner software, these part fractures join together with the use of the HETER0 attribute.

You can use the I/O Designer **Symbol** wizard to quickly create symbols that you can subsequently refine. Alternatively, you can import symbols from another DXDesigner project, and then assign an FPGA to the symbol. To import symbols in the I/O Designer software, **File ➤ Import Symbol**.

I/O Designer symbols are either functional, physical (PCB), or both. Signals imported into the database, usually from Verilog HDL or VHDL files, are the basis of a functional symbol. No physical device pins must be associated with the signals to generate a functional symbol. This section focuses on board-level PCB symbols with signals directly mapped to physical device pins through assignments in either the Intel Quartus Prime Pin Planner or in the I/O Designer database.

3.2.6.1. Generating Schematic Symbols

To create a symbol based on a selected Intel FPGA device, follow these steps:

1. Start the I/O Designer software.
2. Click **Symbol ➤ Symbol Wizard**.
3. In the **Symbol name** field, type the symbol name. The **DEVICE** and **PKG_TYPE** fields display the device and package information.

   Note: If **DEVICE** and **PKG_TYPE** are blank or incorrect, close the Symbol wizard and specify the correct device information (**File ➤ Properties ➤ FPGA Flow)**.
4. Under **Symbol type**, click **PCB**. Under **Use signals**, click **All**, then click **Next**.
5. Select fracturing options for your symbol. If you are using the Symbol wizard to edit a previously created fractured symbol, you must turn on **Reuse existing fractures** to preserve your current fractures. Select other options on this page as appropriate for your symbol. Click **Next**.
6. Select additional fracturing options for your symbol. Click **Next**.
7. Select the options for the appearance of the symbols. Click **Next**.
8. Define the information you want to label for the entire symbol and for individual pins. Click **Next**.
9. Add any additional signals and pins to the symbol. Click **Finish**.
   You can view your symbol and any fractures you created with the Symbol Editor. You can edit parts of the symbol, delete fractures, or rerun the Symbol wizard. When you modify pin assignments in I/O Designer database, I/O Designer symbols automatically reflect these changes. Modify assignments in the I/O Designer software by supplying and updated **.fx** from the Intel Quartus Prime software, or by back-annotating changes in your board layout tool.

3.2.7. Exporting Schematic Symbols to DxDesigner

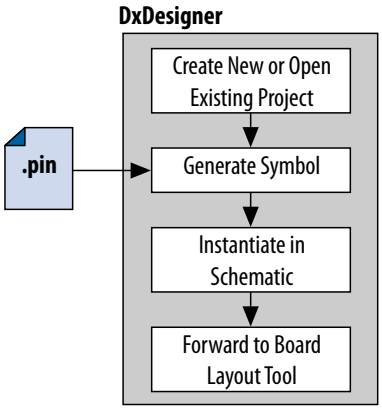
You can export your I/O Designer schematic symbols for to DxDesigner for further design entry work. To generate all fractures of a symbol, click **Generate ➤ All Symbols**. To generate only the currently displayed symbol, click **Generate ➤ Current Symbol Only**. The DxDesigner project **/sym** directory preserves each symbol in the database as a separate file. You can instantiate the symbols in your DxDesigner schematics.

3.3. Integrating with DxDesigner

You can integrate the Mentor Graphics DxDesigner schematic capture tool into the Intel 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. Use DxDesigner in conjunction with I/O Designer software for a complete FPGA I/O and PCB design flow.

If you use DxDesigner without the I/O Designer software, the design flow is one-way, using only the **.pin** generated by the Intel Quartus Prime software. You can only make signal and pin assignment changes in the Intel Quartus Prime software. You cannot back-annotate changes made in a board layout tool or in a DxDesigner symbol to the Intel Quartus Prime software.

Figure 33. DxDesigner-only Flow (without I/O Designer)



3.3.1. DxDesigner Project Settings

DxDesigner new projects automatically create FPGA symbols by default. However, if you are using the I/O Designer with DxDesigner, you must enable DxBoardLink Flow options for integration with the I/O Designer software. 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**.

3.3.2. 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 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 and how itthe grid you have set 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 Intel 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.

3.4. Analyzing FPGA Simultaneous Switching Noise (SSN)

With the Intel Quartus Prime software, you can extract pin assignment data and perform SSN analysis of your design. Perform SSN analysis early in the board layout stage as part of your overall pin planning process. Use the Intel Quartus Prime SSN Analyzer to optimize the pin assignments for better SSN performance.

3.5. Scripting API

The I/O Designer software includes a command line Tcl interpreter. All commands input through the I/O Designer GUI translate into Tcl commands run by the tool. You can run individual Tcl commands or scripts in the I/O Designer Console window, rather than using the GUI.

You can use the following Tcl commands to control I/O Designer.

- `set_fpga_xchange_file <file name>`—specifies the **.fx** from which the I/O Designer software updates assignments.
- `update_from_fpga_xchange_file`—updates the I/O Designer database with assignment updates from the currently specified **.fx**.
- `generate_fpga_xchange_file`—updates the **.fx** with I/O Designer software changes for transfer back into the Intel Quartus Prime software.
- `set_pin_report_file -quartus_pin <file name>`—imports assignment data from an Intel Quartus Prime software .pin file.
- `symbolwizard`—runs the I/O Designer Symbol wizard.
- `set_dx_designer_project -path <path>`

3.6. Document Revision History

Table 4. Document Revision History

Date	Version	Changes
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
2014.06.30	14.0.0	• 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	12.0.0	• Removed survey link.
December 2010	10.1.0	• Changed to new document template.

4. Cadence PCB Design Tools Support

4.1. Cadence PCB Design Tools Support

The Intel 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 Intel 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 Intel 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

4.2. Product Comparison

Table 5. 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 Intel Quartus Prime software through a circuit schematic in the Cadence Allegro Design Entry HDL software or the Cadence Allegro Design Entry CIS software.

Figure 34. Design Flow with the Cadence Allegro Design Entry HDL Software

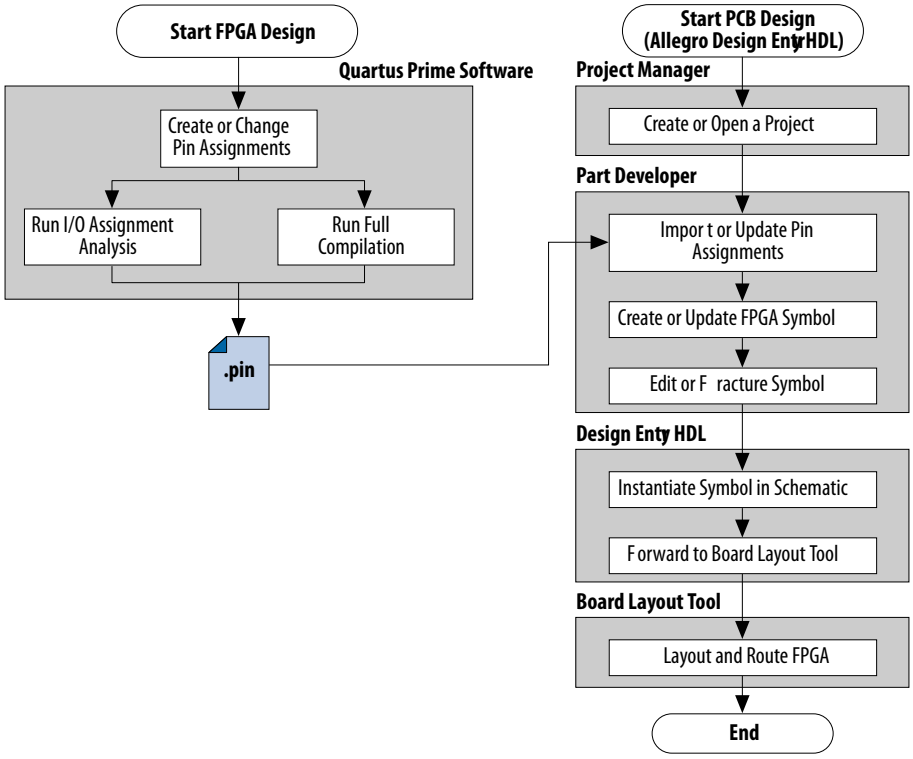
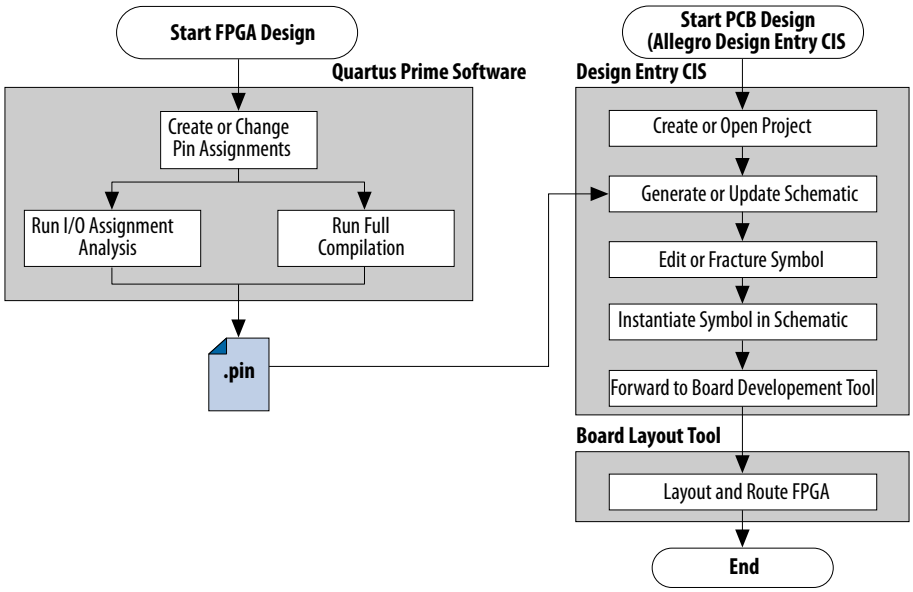


Figure 35. Design Flow with the Cadence Allegro Design Entry CIS Software



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 Design

To integrate an Intel FPGA design starting in the Intel 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 Intel 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.

      or

      If you are using the Cadence Allegro Design Entry CIS software for your schematic design, follow these steps:
   e. Generate a new part in a new or existing Cadence Allegro Design Entry CIS project, referencing the **.pin** output file from the Intel Quartus Prime software. You can also update an existing symbol with a new **.pin**.
   f. Split the symbol into smaller parts as necessary.
   g. Instantiate the symbol in your Cadence Allegro Design Entry CIS schematic and transfer the design to your board layout tool.

4.3.2. Performing Simultaneous Switching Noise (SSN) Analysis of Your FPGA

With the Intel Quartus Prime software, you can extract pin assignment data and perform SSN analysis of your FPGA design for designs targeting the Stratix III device family.

You can analyze SSN in your device early in the board layout stage as part of your overall pin planning process; however, you do not have to perform SSN analysis to generate pin assignment data from the Intel Quartus Prime software. You can use the SSN Analyzer tool to optimize the pin assignments for better SSN performance of your device.

4.4. Setting Up the Intel Quartus Prime Software

You can transfer pin and signal assignments from the Intel Quartus Prime software to the Cadence design tools by generating the Intel Quartus Prime project **.pin**.
The **.pin** is an output file generated by the Intel Quartus Prime Fitter containing pin assignment information. You can use the Intel 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 Intel 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

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.

4.4.1. Generating a .pin File

To generate a **.pin**, follow these steps:

1. Compile your design.
2. Locate the **.pin** in your Intel Quartus Prime project directory with the name <project name>**.pin**.

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.

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 Intel Quartus Prime software. You can only make signal and pin assignment changes in the Intel 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 Design Flow with the Cadence Allegro Design Entry HDL Software.

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 Intel 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 Intel 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 6. 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

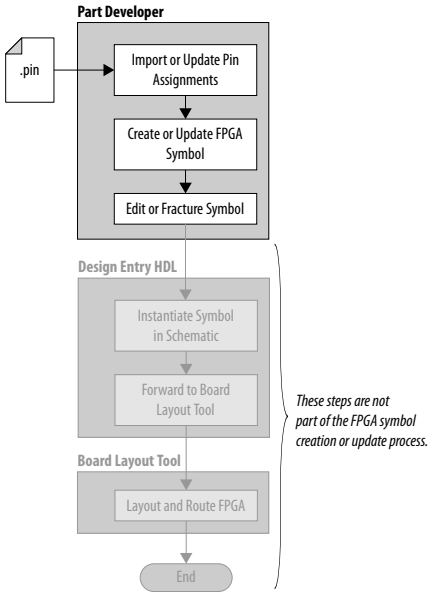
Related Information

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 77

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 Intel 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 Intel 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 Intel 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.

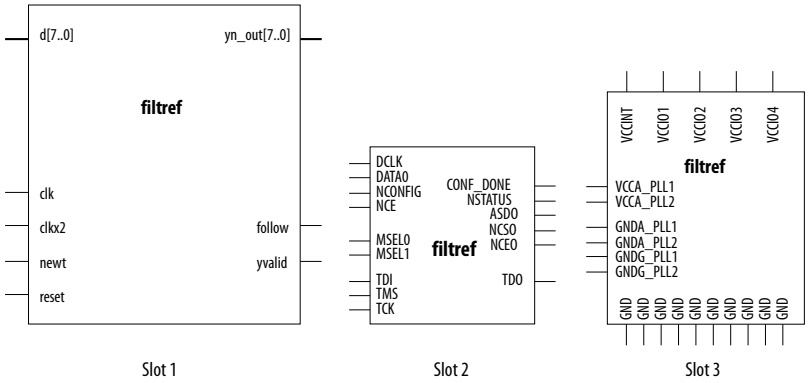
4.5.1.1.3. Editing and Fracturing Symbol

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 36. 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.

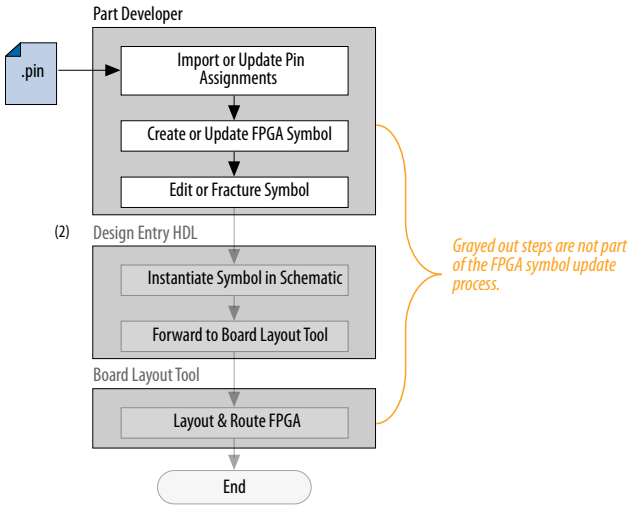
Caution: The Cadence Allegro PCB Librarian Part Developer tool allows you to remap pin assignments in the **Package Pin** tab of the main Cadence Allegro PCB Librarian Part Developer window. If signals remap to different pins in the Cadence Allegro PCB Librarian Part Developer tool, the changes reflect only in regenerated symbols for use in your schematics. You cannot transfer pin assignment changes to the Intel 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 Intel 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 Intel Quartus Prime software, placing signals on different pins after recompiling the design, or use the Intel 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 Intel Quartus Prime software facilitates this flow.

Figure 37. 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 Intel 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 77

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.

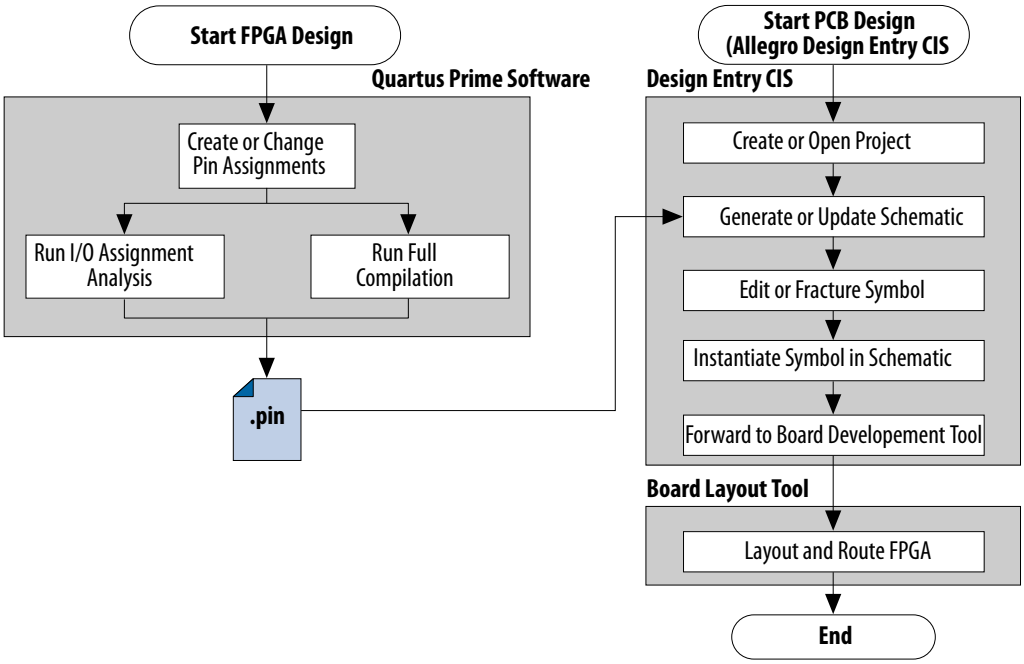
Related Information

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 Intel Quartus Prime software. You can only make signal and pin assignment changes in the Intel Quartus Prime software. These changes reflect as updated symbols in a Cadence Allegro Design Entry CIS schematic project.

Figure 38. 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 Intel 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 Intel 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 Intel 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 Intel 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 Intel 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.

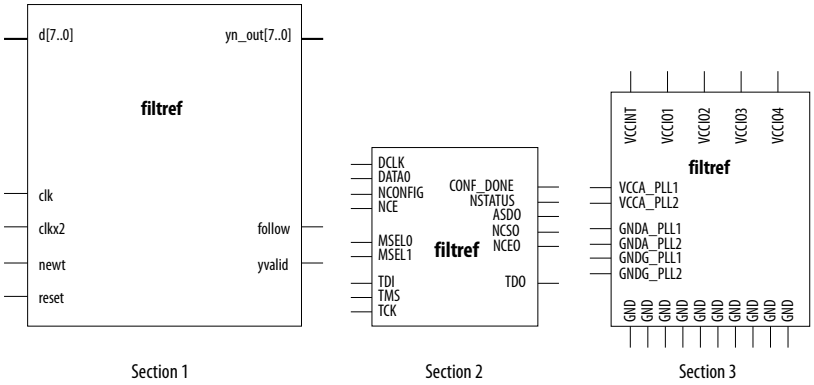
   You can locate the generated symbol in the selected library or in a new library found in the **Outputs** folder of the design in the Project Manager window. Double-click the name of the new symbol to see its graphical representation and edit it manually using the tools available in the Cadence Allegro 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.

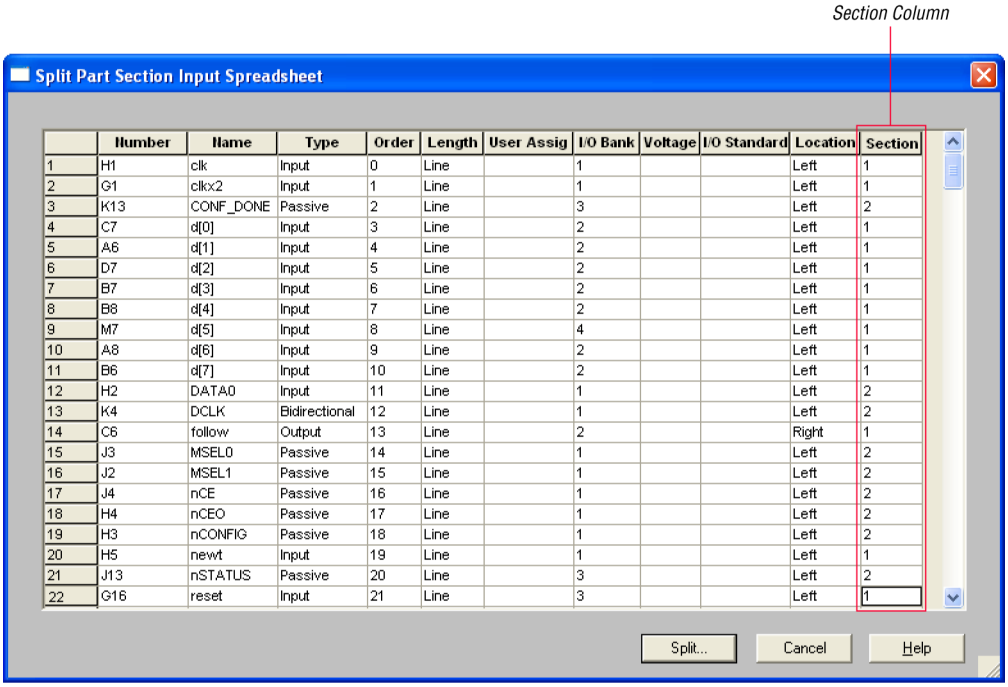
Figure 39. Splitting a Symbol into Multiple Sections



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 40. 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.**

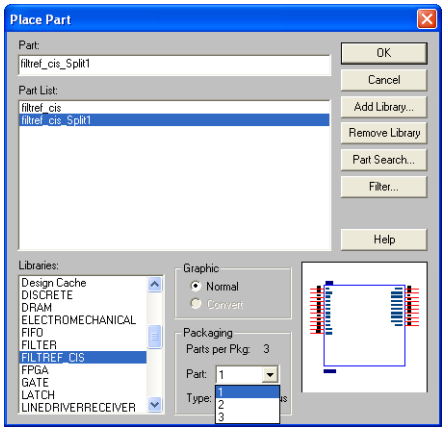
View and edit each section individually. To view the new sections of the part, double-click the part. The Part Symbol Editor window appears and the first section of the part displays for editing. On the View menu, click **Package** to view thumbnails of all the part sections. To edit the section of the 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 41. 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 Intel 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 Altera 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 Intel 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. Document Revision History

Table 7. Document Revision History

Date	Intel Quartus Prime Version	Changes
2018.09.24	18.1.0	• Document title renamed • Other minor edits
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	Converted to DITA format.
June 2012	12.0.0	Removed survey link.
November 2011	10.0.2	Template update.
December 2010	10.0.1	Template update.
July 2010	10.0.0	• 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	9.1.0	• 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	9.0.0	• Chapter 9 was previously Chapter 7 in the 8.1 software release. • No change to content.
November 2008	8.1.0	Changed to 8-1/2 x 11 page size.
May 2008	8.0.0	Updated references.

5. Reviewing Printed Circuit Board Schematics with the Intel 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 Intel Quartus Prime project provides important information specific to your programmable logic design, which you can use in conjunction with the device literature available on Altera's 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 Intel Quartus Prime software also provides the Pin Planner to assist you during your PCB schematic review process.

Related Information

Schematic Review Worksheets

5.1. Reviewing Intel Quartus Prime Software Settings

Review these settings in the Intel Quartus Prime software to help you review your PCB schematic.

The **Device** dialog box in the Intel 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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Related Information

Migration Devices Dialog Box
In Intel Quartus Prime Help

5.1.1. 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.

5.1.1.1. 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 95

5.1.1.2. 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.

5.1.1.3. 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.

5.1.1.4. 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 96

5.1.1.5. 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 Intel Quartus Prime Help

5.2. Reviewing Device Pin-Out Information in the Fitter Report

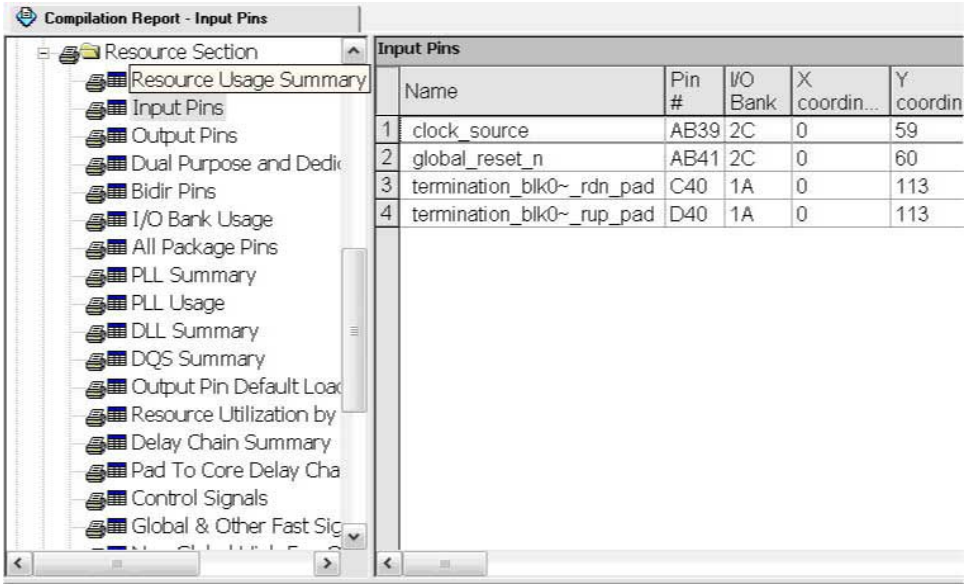
After you compile your design, you can use the reports in the Resource section of the Fitter report to check your device pin-out in detail.

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 42. Resource Section Report

Open the **Compilation Report** tab with `Ctrl+R`, then click **Fitter ➤ Plan StageInput Pins** (or **Output Pins** or **Bidir Pins**). 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.

5.3. 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.

5.4. Using Additional Intel 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 Intel 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 24
- Managing Device I/O Pins

5.5. Using Additional Intel Quartus Prime Software Tools

Use the Pin Planner to assist you with reviewing your PCB schematics.

You can also use the SSN Analyzer to assist you with reviewing your PCB schematics.

5.5.1. Pin Planner

The Intel 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

5.5.2. SSN Analyzer

The SSN Analyzer supports pin planning by estimating the voltage noise caused by the simultaneous switching of output pins on the device. Because of the importance of the potential SSN performance for a specific I/O placement, you can use the SSN Analyzer to analyze the effects of aggressor I/O signals on a victim I/O pin.

5.6. Document Revision History

Table 8. Document Revision History

Date	Version	Changes
2018.09.24	18.1.0	• First release as part of the stand-alone *Intel Quartus Prime Standard EditionUser Guide*
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel Quartus Prime*.
June 2014	14.0.0	Template update.
November 2012	12.1.0	Minor update of Pin Planner description for task and report windows.
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. No change to content.
July 2010	10.0.0	Initial release.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard 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 Standard Edition User Guide

Scripting

Updated for Intel® Quartus® Prime Design Suite: **18.1**

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

Contents

1. Command Line Scripting..... 4
1.1. Benefits of Command-Line Executables.....4
1.2. Introductory Example.....4
1.3. Command-Line Scripting Help..... 5
1.4. Project Settings with Command-Line Options.....6
1.4.1. Option Precedence.....6
1.5. Compilation with quartus_sh --flow.....8
1.6. Text-Based Report Files..... 9
1.7. Using Command-Line Executables in Scripts..... 10
1.8. Common Scripting Examples..... 11
1.8.1. Create a Project and Apply Constraints..... 11
1.8.2. Check Design File Syntax..... 12
1.8.3. Create a Project and Synthesize a Netlist Using Netlist Optimizations..... 12
1.8.4. Archive and Restore Projects..... 13
1.8.5. Perform I/O Assignment Analysis..... 13
1.8.6. Update Memory Contents Without Recompiling..... 13
1.8.7. Create a Compressed Configuration File..... 14
1.8.8. Fit a Design as Quickly as Possible..... 14
1.8.9. Fit a Design Using Multiple Seeds..... 14
1.9. The QFlow Script..... 15
1.10. Document Revision History.....16

2. Tcl Scripting..... 18
2.1. Tool Command Language..... 18
2.2. Intel Quartus Prime Tcl Packages..... 19
2.2.1. Loading Packages.....20
2.3. Intel Quartus Prime Tcl API Help..... 20
2.3.1. Command-Line Options.....22
2.3.2. The Intel Quartus Prime Tcl Console Window.....23
2.4. End-to-End Design Flows..... 24
2.5. Creating Projects and Making Assignments.....24
2.6. Compiling Designs..... 25
2.6.1. The flow Package..... 25
2.6.2. Compile All Revisions.....25
2.7. Reporting.....26
2.7.1. Saving Report Data in csv Format..... 26
2.8. Timing Analysis..... 27
2.9. Automating Script Execution.....27
2.9.1. Execution Example.....28
2.9.2. Controlling Processing..... 29
2.9.3. Displaying Messages.....29
2.10. Other Scripting Features..... 29
2.10.1. Natural Bus Naming.....30
2.10.2. Short Option Names..... 30
2.10.3. Collection Commands..... 30
2.10.4. The post_message Command..... 31
2.10.5. Accessing Command-Line Arguments.....32

2.10.6. The quartus() Array....33
2.11. The Intel Quartus Prime Tcl Shell in Interactive Mode Example....33
2.12. The tclsh Shell....35
2.13. Tcl Scripting Basics....35
2.13.1. Hello World Example....35
2.13.2. Variables....35
2.13.3. Substitutions....35
2.13.4. Arithmetic....36
2.13.5. Lists....37
2.13.6. Arrays....37
2.13.7. Control Structures....38
2.13.8. Procedures....38
2.13.9. File I/O....39
2.13.10. Syntax and Comments....40
2.13.11. External References....40
2.14. Tcl Scripting Revision History....41
A. Intel Quartus Prime Standard Edition User Guides....42

1. Command Line Scripting

FPGA design software that easily integrates into your design flow saves time and improves productivity. The Intel® 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 Intel Quartus Prime GUI, allowing you to use the exact combination of tools that best suits your needs.

1.1. Benefits of Command-Line Executables

Intel 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 Intel Quartus Prime executable files into a script, batch file, or a makefile to automate design flows. These scripting capabilities facilitate the integration of Intel 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 Intel Quartus Prime GUI. You can use the Intel Quartus Prime GUI and command-line executables at different stages in the design flow. For example, you might use the Intel Quartus Prime GUI to edit the floorplan for the design, use the command-line executables to perform place-and-route, and return to the Intel 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 Intel Quartus Prime Help

1.2. Introductory Example

Create a new Intel Quartus Prime project, fit the design, and generate programming files with this example included with the Intel Quartus Prime software.

If installed, the tutorial design is located in the *<Intel Quartus Prime directory>*/`qdesigns/fir_filter` directory.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

1. Ensure that *<Intel Quartus Prime directory>*`/quartus/bin` directory is in your PATH environment variable.
2. Copy the tutorial directory in a local folder.
3. In a console, type the four commands in the new project directory:

```
quartus_map filtref --source=filtref.bdf --family="Cyclone V"
quartus_fit filtref --part=EP3C10F256C8 --pack_register=minimize_area
quartus_asm filtref
quartus_sta filtref
```

   a. With the first instruction you create a new Intel Quartus Prime project named **filtref**, set the top-level file as `filtref.bdf`, set Cyclone® V as the target device family, and perform logic synthesis and technology mapping on the design files.
   b. The second command performs place and route by fitting the **filtref** project into the specified device, and directs the Fitter to pack sequential and combinational functions into single logic cells to reduce device resource usage.
   c. The third command creates a device programming image for the **filtref** project.
   d. The last line performs basic timing analysis on the **filtref** project using the Intel Quartus Prime Timing Analyzer, reporting worst-case setup slack, worst-case hold slack, and other measurements.
4. Create a batch file or script file with the commands, like the UNIX shell script below:

```
#!/bin/sh
PROJECT=filtref
TOP_LEVEL_FILE=filtref.bdf
FAMILY="Cyclone V"
PART=EP3C10F256C8
PACKING_OPTION=minimize_area
quartus_map $PROJECT --source=$TOP_LEVEL_FILE --family=$FAMILY
quartus_fit $PROJECT --part=$PART --pack_register=$PACKING_OPTION
quartus_asm $PROJECT
quartus_sta $PROJECT
```

5. Execute the script and compile your project.

Related Information

Intel Quartus Prime Scripting Reference Manual

1.3. 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 Intel Quartus Prime Command-Line and Tcl API Help browser for an easy graphical view of the help information.

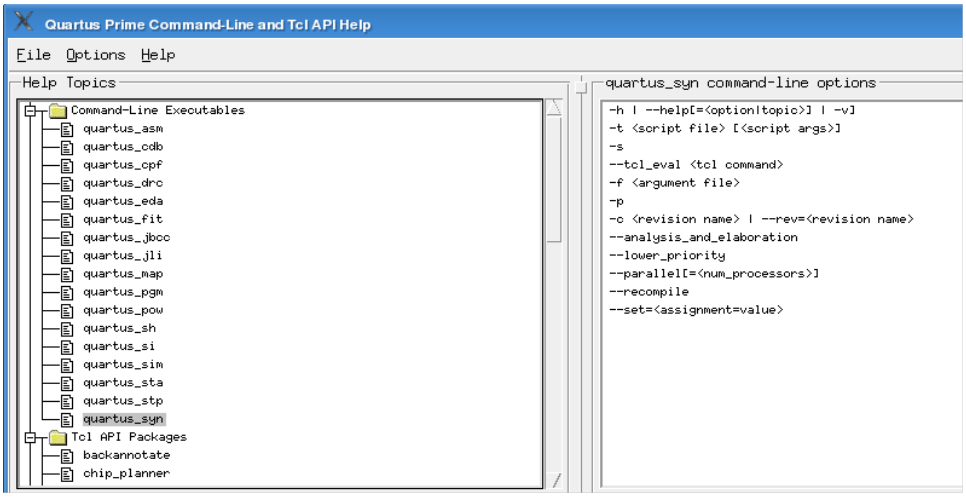
To use the Intel Quartus Prime Command-Line and Tcl API Help browser, type the following command:

```
quartus_sh --qhelp
```

This command starts the Intel Quartus Prime Command-Line and Tcl API Help browser, a viewer for information about the Intel Quartus Prime Command-Line executables and Tcl API.

Use the `-h` option with any of the Intel 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. Intel Quartus Prime Command-Line and Tcl API Help Browser



1.4. Project Settings with Command-Line Options

The Intel 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 Intel Quartus Prime Tcl scripting API. On existing projects, you can also open the project in the Intel 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

- Tcl Scripting on page 18
- Intel Quartus Prime Settings File (.qsf) Definition
  in Intel Quartus Prime Help
- Intel Quartus Prime Standard Edition Settings File Reference Manual

1.4.1. Option Precedence

Project assignments follow a set of precedence rules. Assignments for a project can exist in three places:

- Intel 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. Intel Quartus Prime software defaults
`--read_settings_files = off`	1. Command-line options 2. Project database (db directory, if it exists) 3. Intel 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.

The example assumes that a project named fir_filter exists, and that the analysis and synthesis step has been performed.

```
quartus_fit fir_filter --pack_register=off
quartus_sta fir_filter
mv fir_filter_sta.rpt fir_filter_1_sta.rpt
quartus_fit fir_filter --pack_register=minimize_area     --
write_settings_files=off
quartus_sta fir_filter
mv fir_filter_sta.rpt fir_filter_2_sta.rpt
```

The first command, `quartus_fit fir_filter --pack_register=off`, runs the `quartus_fit` executable with no aggressive attempts to reduce device resource usage.

The second command, `quartus_sta fir_filter`, performs basic timing analysis for the results of the previous fit.

The third command uses the UNIX `mv` command to copy the report file output from `quartus_sta` to a file with a new name, so that the results are not overwritten by subsequent timing analysis.

The fourth command runs `quartus_fit` a second time, and directs it to attempt to pack logic into registers to reduce device resource usage. With the `--write_settings_files=off` option, the command-line executable does not update the `.qsf` to reflect the changed register packing setting. Instead, only the compiler database files reflect the changed setting. If the `--write_settings_files=off` option is not specified, the command-line executable updates the `.qsf` to reflect the register packing setting.

The fifth command reruns timing analysis, and the sixth command renames the report file, so that it is not overwritten by subsequent timing analysis.

Use the options `--read_settings_files=off` and `--write_settings_files=off` (where appropriate) to optimize the way that the Intel Quartus Prime software reads and updates settings files.

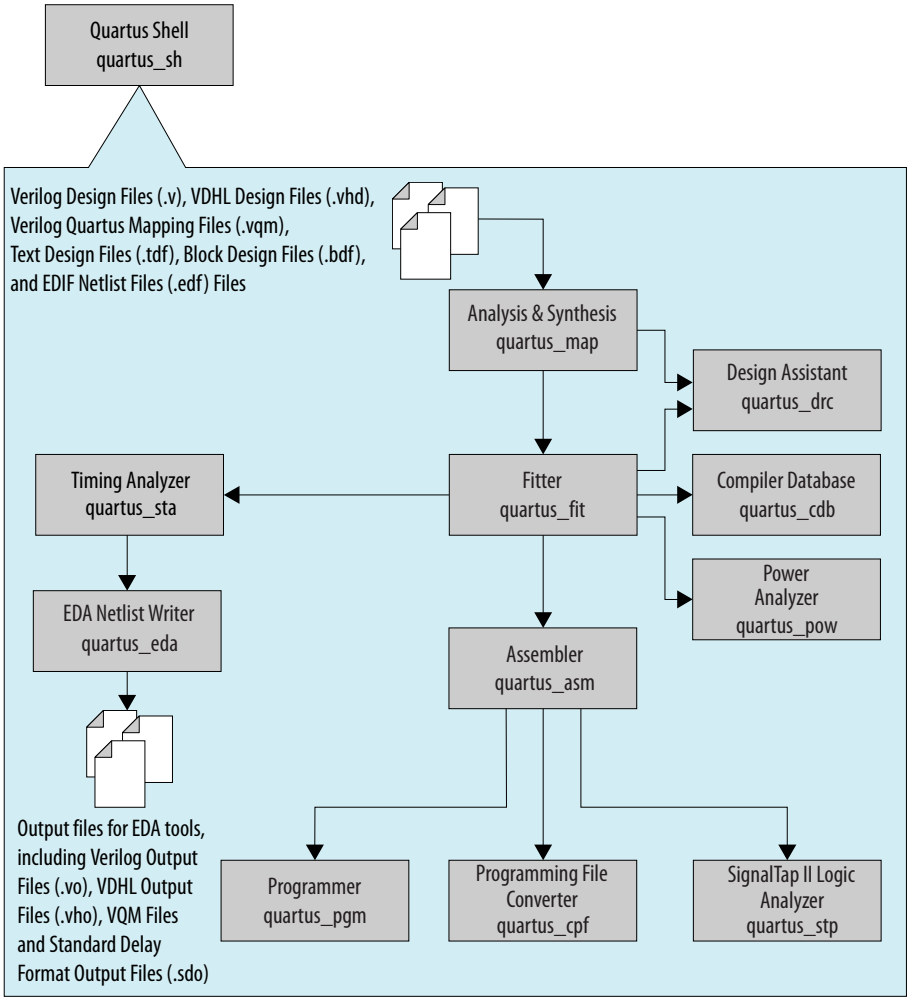
In this example, the `quartus_asm` executable does not read or write settings files:

```
quartus_map filtref --source=filtref --part=EP3C10F256C8
quartus_fit filtref --pack_register=off --read_settings_files=off
quartus_asm filtref --read_settings_files=off --write_settings_files=off
```

1.5. Compilation with quartus_sh --flow

The figure shows a typical Intel 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.

The following example runs compilation, 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.

1.6. 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 Intel Quartus Prime Help
- ::quartus::report
  in Intel Quartus Prime Help

1.7. Using Command-Line Executables in Scripts

You can use command-line executables in scripts that control other software, in addition to Intel 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 Intel 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 Intel 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 Intel Quartus Prime software, and generate output netlists for other simulation software.

This script shows a script that synthesizes a design with the Synopsys* Synplify software, simulates the design using the Mentor Graphics* ModelSim® software, and then compiles the design targeting a Cyclone V device.

```
#!/bin/sh
# Run synthesis first.
# This example assumes you use Synplify software
synplify -batch synthesize.tcl
# If your Quartus Prime project exists already, you can just
# recompile the design.
# You can also use the script described in a later example to
# create a new project from scratch
quartus_sh --flow compile myproject
# Use the quartus_sta executable to do fast and slow-model
# timing analysis
quartus_sta myproject --model=slow
quartus_sta myproject --model=fast
# Use the quartus_eda executable to write out a gate-level
# Verilog simulation netlist for ModelSim
quartus_eda my_project --simulation --tool=modelsim --format=verilog
```

```
# Perform the simulation with the ModelSim software
vlib cycloneV_ver
vlog -work cycloneV_ver /opt/quartusii/eda/sim_lib/cycloneV_atoms.v
vlib work
vlog -work work my_project.vo
vsim -L cycloneV_ver -t 1ps work.my_project
```

1.8. Common Scripting Examples

You can create scripts including command line executable to control common Intel Quartus Prime processes.

1.8.1. 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 <Intel Quartus Prime *installation directory*>/`qdesigns/fir_filter/` directory.

```
project_new filtref -overwrite
# Assign family, device, and top-level file
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C12F256C6
set_global_assignment -name BDF_FILE filtref.bdf
# 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_map 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.

1.8.2. Check Design File Syntax

The UNIX shell script example below assumes the Intel Quartus Prime software **fir_filter** tutorial project exists in the current directory. You can find the **fir_filter** project in the <*Intel Quartus Prime directory*>/qdesigns/fir_filter directory unless the Intel Quartus Prime software tutorial files are not installed.

The script checks the exit code of the quartus_map executable to determine whether there is an error during the syntax check. Files with syntax errors are added to the FILES_WITH_ERRORS variable, and when all files are checked, the script prints a message indicating syntax errors.

When options are not specified, the executable uses the project database values. If not specified in the project database, the executable uses the Intel Quartus Prime software default values. For example, the **fir_filter** project is set to target the Cyclone device family, so it is not necessary to specify the --family option.

```
#!/bin/sh
FILES_WITH_ERRORS=""
# Iterate over each file with a .bdf or .v extension
for filename in `ls *.bdf *.v`
do
# Perform a syntax check on the specified file
    quartus_map fir_filter --analyze_file=$filename
  # If the exit code is non-zero, the file has a syntax error
    if [ $? -ne 0 ]
    then
       FILES_WITH_ERRORS="$FILES_WITH_ERRORS $filename"
    fi
done
if [ -z "$FILES_WITH_ERRORS" ]
then
    echo "All files passed the syntax check"
    exit 0
else
    echo "There were syntax errors in the following file(s)"
    echo $FILES_WITH_ERRORS
    exit 1
fi
```

1.8.3. Create a Project and Synthesize a Netlist Using Netlist Optimizations

This example creates a new Intel 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_map 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_map top --source=top.edf --enable_register_retiming=on
    --enable_wysiwyg_resynthesis=on --part=EP3C10F256C8
```

1.8.4. Archive and Restore Projects

You can archive or restore an Intel 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 Intel Quartus Prime Projects

1.8.5. Perform I/O Assignment Analysis

You can perform I/O assignment analysis with a single command. I/O assignment analysis checks pin assignments to ensure they do not violate board layout guidelines. I/O assignment analysis does not require a complete place and route, so it can quickly verify that your pin assignments are correct.

```
quartus_fit --check_ios <project name> --rev=<revision name>
```

1.8.6. 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>
```

1.8.7. Create a Compressed Configuration File

You can create a compressed configuration file in two ways. The first way is to run `quartus_cpf` with an option file that turns on compression.

To create an option file that turns on compression, type the following command at a command prompt:

```
quartus_cpf -w <filename>.opt
```

This interactive command guides you through some questions, then creates an option file based on your answers. Use `--option` to cause `quartus_cpf` to use the option file. For example, the following command creates a compressed `.pof` that targets an EPCS64 device:

```
quartus_cpf --convert --option=<filename>.opt --device=EPCS64 <file>.sof
<file>.pof
```

Alternatively, you can use the Convert Programming Files utility in the Intel Quartus Prime software GUI to create a Conversion Setup File (`.cof`). Configure any options you want, including compression, then save the conversion setup. Use the following command to run the conversion setup you specified.

```
quartus_cpf --convert <file>.cof
```

1.8.8. Fit a Design as Quickly as Possible

This example assumes that a project called **top** exists in the current directory, and that the name of the top-level entity is **top**. The `--effort=fast` option causes the `quartus_fit` to use the fast fit algorithm to increase compilation speed, possibly at the expense of reduced f_{MAX} performance. The `--one_fit_attempt=on` option restricts the Fitter to only one fitting attempt for the design.

To attempt to fit the project called **top** as quickly as possible, type the command shown at a command prompt.

```
quartus_fit top --effort=fast --one_fit_attempt=on
```

1.8.9. Fit a Design Using Multiple Seeds

This shell script example assumes that the Intel 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 *<Intel 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 Intel 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_map 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 Intel Quartus Prime software script (by typing `quartus_dse` at a command prompt) to improve design performance by performing automated seed sweeping.

1.9. 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_map` (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 Intel 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 *<Intel Quartus Prime directory>*`/common/tcl/apps/qflow/` directory.

1.10. Document Revision History

Table 3. Document Revision History

Date	Version	Changes
2017.05.08	17.0.0	• Reorganized content on topics: Benefits of Command-Line Executables and Project Settings with Command-Line Options.
2015.11.02	15.1.0	Changed instances of *Quartus II* to *Intel 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.
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:

continued...

Date	Version	Changes
		• "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 to control the Intel Quartus Prime software and to perform a wide range of functions, such as compiling a design or scripting common tasks.

For example, use Tcl scripts to perform the following tasks:

- Manage an Intel Quartus Prime project
- Make assignments
- Define design constraints
- Make device assignments
- Compile your design
- Perform timing analysis
- Access reports

Tcl scripts also facilitate project or assignment migration. For example, when designing in different projects with the same prototype or development board, you can write a script to automate reassignment of pin locations in each new project. The Intel Quartus Prime software can also generate a Tcl script based on all the current assignments in the project, which aids in switching assignments to another project.

The Intel Quartus Prime software Tcl commands follow the EDA industry Tcl application programming interface (API) standards for command-line options. This simplifies learning and using Tcl commands. 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 Intel Quartus Prime software. You can modify these example scripts for use with your own designs. You can find more Tcl scripts in the Design Examples section of the Support area on the Altera website.

Related Information

Tcl Design Examples

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.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

By combining Tcl commands and Intel Quartus Prime API functions, you can create your own procedures and automate your design flow. Run Intel Quartus Prime software in batch mode, or execute individual Tcl commands interactively in the Intel Quartus Prime Tcl shell.

Intel Quartus Prime software supports Tcl/Tk version 8.5, supplied by the Tcl DeveloperXchange.

2.2. Intel Quartus Prime Tcl Packages

The Intel Quartus Prime software groups Tcl commands into packages by function.

Table 4. Intel Quartus Prime Tcl Packages

Package Name	Package Description
backannotate	Back annotate assignments
chip_planner	Identify and modify resource usage and routing with the Chip Editor
database_manager	Manage version-compatible database files
device	Get device and family information from the device database
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
flow	Compile a project, run command-line executables, and other common flows
incremental compilation	Manipulate design partitions and Logic Lock (Standard) regions, and settings related to incremental compilation
insystem_memory_edit	Read and edit memory contents in Intel devices
insystem_source_probe	Interact with the In-System Sources and Probes tool in an Intel device
iptclgen	Generate memory IP
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
partial_reconfiguration	Contain the set of Tcl functions for performing partial reconfiguration
project	Create and manage projects and revisions, make any project assignments including timing assignments
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-specific SDC commands
simulator	Configure and perform simulations
sta	Contain the set of Tcl functions for obtaining advanced information from the Timing Analyzer
stp	Run the Signal Tap Logic Analyzer
synthesis_report	Contain the set of Tcl functions for the Dynamic Synthesis Report tool
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 Intel 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
```

2.2.1. Loading Packages

To load an Intel 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 Intel Quartus Prime Tcl package.

Related Information

Command Line Scripting on page 4

2.3. Intel Quartus Prime Tcl API Help

Intel Quartus Prime Tcl help allows easy access to information about the Intel Quartus Prime Tcl commands.

- This command opens the Intel Quartus Prime Command-Line and Tcl API help browser, which documents all commands and options in the Intel Quartus Prime Tcl API. At a system command prompt, access the Intel 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:

  ```
  tcl> help
  ---------------------------------------------------------------------------
  ---------------------------------
  Available Intel Quartus Prime Tcl Packages:
  ---------------------------------
  Loaded              Not Loaded
  ------------------  ---------------------------------
  ```

```
::quartus::device   ::quartus::external_memif_toolkit
::quartus::misc     ::quartus::iptclgen
::quartus::project  ::quartus::design
                    ::quartus::rtm
                    ::quartus::partial_reconfiguration
                    ::quartus::report
                    ::quartus::names
                    ::quartus::incremental_compilation
                    ::quartus::flow

* Type "help -tcl"
to get an overview on Intel Quartus Prime Tcl usages.

* Type "help <package name>"
  to view a list of Tcl commands available for
  the specified Intel Quartus Prime Tcl package.
```

The Tcl console provides help options that display specific information:

Table 5. Help Options Available in the Intel Quartus Prime Tcl Environment

Help Command	Description
`help`	Displays complete list of available Intel 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 Intel Quartus Prime package that you specify, including the list of available Tcl commands.
	• If you do not specify `-version`, the Intel Quartus Prime software loads the latest version of the package.
	• If the package is not loaded, the Intel 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`

`<command_name> -h`
or
`<command_name> -help`
	Displays the short help of a Intel 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 Intel Quartus Prime Tcl package. If you do not specify `-version`, the Intel 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`

continued...

Help Command	Description

`help -cmd <command_name> -[-version <version>]`
or
`<command_name> -long_help`
	Displays the complete help text for an Intel Quartus Prime Tcl command. If you do not specify `-version`, the Intel 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 Intel Quartus Prime Tcl usage.
`help -quartus`	To view help on the predefined global Tcl array that contains project information and information about the Intel Quartus Prime executable that is currently running.
`quartus_sh --qhelp`	Launches the Tk viewer for Intel 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 Intel Quartus Prime online help. Search for the command or package name to find details about that command or package.

2.3.1. Command-Line Options

You can use any of the following command line options with executables that support Tcl:

Table 6. 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`

2.3.1.1. 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 *<Intel 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
```

2.3.1.2. Interactive Shell Mode

Running an executable with the `-s` option starts an interactive Tcl shell. For example, to open the Intel 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.

2.3.1.3. 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
```

2.3.2. The Intel Quartus Prime Tcl Console Window

To run Tcl commands directly in the Intel Quartus Prime **Tcl Console** window, click **View ➤ Utility Windows**. By default, the **Tcl Console** window is docked in the bottom-right corner of the Intel Quartus Prime GUI. All Tcl commands typed in the **Tcl Console** are interpreted by the Intel Quartus Prime Tcl shell.

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 Intel Quartus Prime executables from a system command prompt outside of the Intel 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 Intel Quartus Prime **Tcl Console** window.

2.4. End-to-End Design Flows

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 Intel Quartus Prime Tcl interpreter supports all core Tcl commands, and adds numerous commands specific to the Intel 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 Intel 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 Intel Quartus Prime GUI, and are also documented as options for the `execute_flow` Tcl command. If you can make settings in the Intel 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.

2.5. 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, Intel Quartus Prime software automatically commits the assignment changes to the `.qsf` file. To prevent this behavior, specify the `-dont_export_assignments` logic option.

Example 1. Create and Compile a Project

The following example creates a project, makes assignments, and compiles the design. The example uses the `fir_filter` tutorial design files in the `qdesigns` installation directory. Run this script in the `fir_filter` directory, with the `quartus_sh` executable.

```
load_package flow
# Create the project and overwrite any settings
# files that exist
project_new fir_filter -revision filtref -overwrite
# Set the device, the name of the top-level BDF,
# and the name of the top-level entity
set_global_assignment -name FAMILY Cyclone
set_global_assignment -name DEVICE EP1C6F256C6
set_global_assignment -name BDF_FILE filtref.bdf
set_global_assignment -name TOP_LEVEL_ENTITY filtref
# Add other pin assignments here
set_location_assignment -to clk Pin_G1
# compile the project
execute_flow -compile
project_close
```

Related Information

- Intel Quartus Prime Standard Edition Settings File Reference Manual
- Interactive Shell Mode on page 23

2.6. Compiling Designs

You can run the Intel Quartus Prime command-line executables from Tcl scripts. Use the included `flow` package to run various Intel Quartus Prime compilation flows, or run each executable directly.

2.6.1. The flow Package

The `flow` package includes two commands for running Intel Quartus Prime command-line executables, either individually or together in standard compilation sequence.

- The `execute_module` command allows you to run an individual Intel 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 Intel Quartus Prime executables from scripts or from the Intel Quartus Prime Tcl Console.

2.6.2. 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
```

2.7. Reporting

You can extract information from the Compilation Report to evaluate results. The Intel 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"
}
```

2.7.1. Saving Report Data in csv Format

You can create a Comma Separated Value (`.csv`) file from any Intel 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.

2.8. Timing Analysis

The Intel Quartus Prime Timing Analyzer includes support for industry-standard SDC commands in the `sdc` package.

The Intel Quartus Prime software includes comprehensive Tcl APIs and SDC extensions for the Timing Analyzer in the `sta`, and `sdc_ext` packages. The Intel Quartus Prime software also includes a `tdc` package that obtains information from the Timing Analyzer.

Related Information

Intel Quartus Prime Standard Edition Settings File Reference Manual

2.9. 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 Intel Quartus Prime executable that performs one step in a flow. For example, two modules are Analysis and Synthesis (`quartus_map`), and timing analysis (`quartus_sta`).

A flow is a series of modules that the Intel 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_map`)
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 Intel 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 Intel 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 Intel 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.

2.9.1. 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 Intel Quartus Prime software starts compilation with analysis and synthesis, performed by the `quartus_map` 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_map top rev_1
```

Then, the Intel 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
```

2.9.2. 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"
}
```

2.9.3. Displaying Messages

Because of the way the Intel 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 31
- Automating Script Execution on page 27

2.10. Other Scripting Features

The Intel Quartus Prime Tcl API includes other general-purpose commands and features described in this section.

2.10.1. Natural Bus Naming

The Intel 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 Intel 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 Intel Quartus Prime Tcl prompt:

```
enable_natural_bus_naming -h
```

2.10.2. 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.

2.10.3. Collection Commands

Some Intel 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 Intel Quartus Prime Tcl API uses a collection ID to access the collection.

There are two Intel 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.

2.10.3.1. 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 Intel Quartus Prime Help

2.10.3.2. 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"
```

2.10.4. The post_message Command

To print messages that are formatted like Intel 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 Intel 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, Intel Quartus Prime software defaults to `info`.

With the Intel 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"
```

2.10.5. 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 2. 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 3. 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
```

2.10.5.1. The cmdline Package

You can use the `cmdline` package included with the Intel 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 Intel 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
```

2.10.6. The quartus() Array

The global `quartus()` Tcl array includes other information about your project and the current Intel 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
```

2.11. The Intel 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 Intel Quartus Prime software gives the revision the same name as the project.
- If a `.qpf` file for this project already exists, the Intel Quartus Prime software will display 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 Intel 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_map`, `quartus_fit`, `quartus_asm`, and `quartus_sta`. This sequence of events is the same as selecting **Processing ➤ Start Compilation** in the Intel 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.

2.12. The tclsh Shell

On the UNIX and Linux operating systems, the tclsh shell included with the Intel 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.

2.13. Tcl Scripting Basics

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 Intel Quartus Prime executables or with the tclsh shell.

2.13.1. 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.

2.13.2. 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"
```

2.13.3. Substitutions

Tcl performs three types of substitution:

- Variable value substitution
- Nested command substitution
- Backslash substitution

2.13.3.1. Variable Value Substitution

Variable value substitution, refers to accessing the value stored in a variable with a dollar sign ("$") before the variable name.

2.13.3.2. 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]
```

2.13.3.3. 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"
```

2.13.4. 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 Intel 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).

2.13.5. 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
```

2.13.6. 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.

2.13.7. 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.

2.13.8. 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
```

2.13.9. 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
```

2.13.10. 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
}
```

2.13.11. 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

2.14. Tcl Scripting Revision History

Table 7. Document Revision History

Date	Version	Changes
2015.11.02	15.1.0	• Changed instances of *Quartus II* to *Intel Quartus Prime*. • Updated the list of Tcl packages in the *Intel Quartus Prime Tcl Packages* section. • Updated the *Intel 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 LogicLock 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.

A. Intel Quartus Prime Standard Edition User Guides

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

Related Information

- Intel Quartus Prime Standard Edition User Guide: Getting Started
  Introduces the basic features, files, and design flow of the Intel Quartus Prime Standard Edition software, including managing Intel Quartus Prime Standard Edition projects and IP, initial design planning considerations, and project migration from previous software versions.
- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  Describes creating and optimizing systems using Platform Designer (Standard), a system integration tool that simplifies integrating customized IP cores in your project. Platform Designer (Standard) automatically generates interconnect logic to connect intellectual property (IP) functions and subsystems.
- Intel Quartus Prime Standard Edition User Guide: Design Recommendations
  Describes best design practices for designing FPGAs with the Intel Quartus Prime Standard Edition software. HDL coding styles and synchronous design practices can significantly impact design performance. Following recommended HDL coding styles ensures that Intel Quartus Prime Standard Edition synthesis optimally implements your design in hardware.
- Intel Quartus Prime Standard Edition User Guide: Design Compilation
  Describes set up, running, and optimization for all stages of the Intel Quartus Prime Standard Edition Compiler. The Compiler synthesizes, places, and routes your design before generating a device programming file.
- Intel Quartus Prime Standard Edition User Guide: Design Optimization
  Describes Intel Quartus Prime Standard 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, and optimization of device resource usage.
- Intel Quartus Prime Standard Edition User Guide: Programmer
  Describes operation of the Intel Quartus Prime Standard 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 Standard 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 Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Quartus Prime Standard 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 Standard 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 Standard Edition User Guide: Debug Tools
  Describes a portfolio of Intel Quartus Prime Standard 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 Standard Edition User Guide: Timing Analyzer
  Explains basic static timing analysis principals and use of the Intel Quartus Prime Standard 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 Standard Edition User Guide: Power Analysis and Optimization
  Describes the Intel Quartus Prime Standard 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 Standard 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 Pin Planner to visualize, modify, and validate all I/O assignments in a graphical representation of the target device.
- Intel Quartus Prime Standard 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 Standard Edition User Guide: Scripting
  Describes use of Tcl and command line scripts to control the Intel Quartus Prime Standard Edition software and to perform a wide range of functions, such as managing projects, specifying constraints, running compilation or timing analysis, or generating reports.