4.4. Setting Up the Simulation Environment ................................................................. 81
  4.4.1. Simulation Levels .......................................................... 81
  4.4.2. HDL Support for EDA Simulators ......................................................... 82
  4.4.3. Value Change Dump (VCD) Support ....................................................... 82
  4.4.4. Simulating Intel FPGA IP Cores .............................................................. 82

5. Conclusion .............................................................................................................. 83

6. Document Revision History for Intel FPGA Design Flow for Xilinx Users ............. 84
1. Introduction to Intel® FPGA Design Flow for Xilinx* Users

Designing for Intel® Field Programmable Gate Array (FPGA) devices is similar, in concept and practice, to designing for Xilinx® FPGAs. In most cases, you can simply import your register transfer level (RTL) into the Intel Quartus® Prime Pro Edition software and begin compiling your design to the target device.

This document is intended for Xilinx designers who are familiar with the Xilinx Vivado* software and want to convert existing Vivado designs to the Intel Quartus Prime Pro Edition software environment.

This application note starts with a description of the current Xilinx and Intel FPGA technologies and compares devices available for three different process technologies. It further highlights unique features of Intel Stratix® 10, Intel Arria® 10, and Intel Cyclone® 10 GX devices supported in the latest edition of the Intel Quartus Prime Pro Edition software.

The next chapter draws a parallel between the design flows in the Intel Quartus Prime Pro Edition software and Xilinx Vivado software, comparing features whenever possible.

The following chapter provides guidelines to convert Vivado designs to the Intel Quartus Prime Pro Edition software, including Xilinx IP Catalog modules and instantiated primitives. The last part of the chapter demonstrates how to translate device and design constraints.

This application note uses the latest information available for the Intel Quartus Prime Pro Edition software version 17.1 and Xilinx Vivado Design Suite version 2017.2, supporting the latest programmable chips.
2. Technology Comparison

2.1. Intel FPGAs

Intel FPGAs are ideal for a wide variety of applications, from high-volume applications to state-of-the-art products. Each FPGA series include different features, such as embedded memory, digital signal processing (DSP) blocks, high-speed transceivers, or high-speed I/O pins, to cover a broad range of end products. Intel has four classes of FPGAs to meet market needs from the industry’s highest density and performance to the most cost effective:

- Stratix FPGA and SoC family enables you to deliver high-performance, state-of-the-art products to market faster with lower risk and higher productivity
- Arria family delivers optimal performance and power efficiency in the midrange.
- Cyclone FPGA series is built to meet your low-power, cost-sensitive design needs, enabling you to get to market faster.

The latest Intel Stratix 10 FPGAs and SoCs are built on the Intel’s 14 nm Tri-Gate process, and introduces the heterogenous 3D system-in-package (SiP) technology. The SiP technology enables in-package integration of a range of components such as analog, memory, ASIC, CPU, etc. while keeping the FPGA fabric monolithic. In addition, Intel Stratix 10 FPGAs integrate transceiver die or tiles from different process nodes in the same package. This unique solution addresses these challenges: higher bandwidth, lower power, smaller form factor, and increased functionality and flexibility. For more information about SiP, refer to the Enabling Next-Generation Platforms Using Intel’s 3D System-in-Package Technology white paper.

Intel SoCs bring high integration and advanced system, power, and security management capabilities to Intel's product portfolio. Industry-standard Arm* tools, along with a broad ecosystem of operating systems and development tools, support Intel SoCs.

Related Information

- Enabling Next-Generation Platforms Using Intel’s 3D System-in-Package Technology White Paper
- Achieving the Highest Levels of Integration in Programmable Logic White Paper
2.2. Xilinx FPGAs

Xilinx UltraScale+ family of FPGAs, 3D ICs and MPSoCs are built on TSMC’s 16nm process and use a homogeneous integration technology which breaks the FPGA fabric into multiple dice. This approach uses Stacked(1) Silicon Interconnect (SSI) to connect the die tiles. For more information refer to the Xilinx website.

**Related Information**
- Xilinx Virtex* UltraScale+ Product Brief
- Xilinx All Programmable 3D ICs

2.3. Comparison Table

The following table lists different range devices available in last 3 process technologies for Intel and Xilinx:

<table>
<thead>
<tr>
<th>Process Technology Comparison</th>
<th>20 nm</th>
<th>16 nm</th>
<th>14 nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Performance Or Fastest, Most Powerful</td>
<td>Virtex* UltraScale*</td>
<td>-</td>
<td>Virtex UltraScale+ Zynq* UltraScale+(2)</td>
</tr>
<tr>
<td>Best Price/ performance/watt Or Balance of cost, power, performance</td>
<td>Intel Arria 10(3)</td>
<td>Kintex UltraScale</td>
<td>Kintex UltraScale+ Zynq UltraScale+ MPSoC</td>
</tr>
<tr>
<td>Cost-Optimized Or Low system cost plus performance</td>
<td>Intel Cyclone 10 GX</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

---

(1) Horizontal stacking of the die on an interposer

(2) MPSoC and RFSoC variants are available.

(3) SoC variants are available.
2.4. Intel FPGA Device Features

Table 2. Intel Arria 10 Device Features

<table>
<thead>
<tr>
<th>Performance</th>
<th>A speed grade faster core performance and up to a 20% fMAX advantage compared to the competition, using publicly-available Intel FPGA IP Evaluation Mode designs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Intel Arria 10 FPGAs and SoCs are up to 40 percent lower power than previous generation FPGAs and SoCs</td>
</tr>
</tbody>
</table>
| Industry’s only | • Hard floating-point digital signal processing (DSP) blocks with speeds up to 1.5 tera floating-point operations per second (TFLOPS)(4)  
  • 20 nm ARM-based SoC |
| Applications | FPGAs and SoCs deliver the integration needed to address a wide range of applications for many industries, including communications, defense, broadcast, high-performance computing, test, and medical |

Table 3. Intel Stratix 10 Device Features

<table>
<thead>
<tr>
<th>Performance</th>
<th>Built on the Intel 14 nm Tri-Gate process, Intel Stratix 10 devices deliver 2X core performance gains over previous-generation, high-performance FPGAs with up to 70% lower power.(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Take advantage of heterogeneous 3D system-in-package (SiP) technology to integrate a monolithic FPGA core fabric with 3D SiP transceiver tiles and other advanced components in a single package.</td>
</tr>
</tbody>
</table>
| Floating-point operations | • The hardened floating-point operators within each DSP block, initially introduced in the Intel Arria 10 device family, is extended to deliver an order of magnitude greater throughput in Intel Stratix 10 FPGAs and SoCs.  
  • Digital signal processing (DSP) designs can achieve up to 10 tera floating point operations per second (TFLOPS) of IEEE 754 single-precision floating-point operations. |
| Processor   | Highly efficient quad-core Arm Cortex*-A53 processor cluster optimized for ultra-high performance per watt, which reduces power consumption up to 70% over previous-generation SoC FPGAs.(5) |
| Applications | Uniquely positioned to address next-generation, high-performance systems in the most demanding applications including communications, datacenter acceleration, high performance computing, radar processing, ASIC prototyping, and many more. |

(4) Tests measure performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. Performance comparison methodology and detailed results documented in the Intel Arria 10 Performance Benchmarking Methodology and Results white paper on www.altera.com/arriaperformance.

Table 4. **Intel Cyclone 10 GX**

<table>
<thead>
<tr>
<th>Industry’s first</th>
<th>Low-cost FPGA with IEEE 754-compliant hard floating-point DSP blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td>Optimized for high-bandwidth, performance applications such as Industrial Vision, Robotics, and Automotive Infotainment.</td>
</tr>
</tbody>
</table>

**Related Information**

- Intel FPGA Products Overview
- Intel SoC Products Overview
- Intel Arria 10 Performance Benchmarking Methodology and Results
3. FPGA Tools Comparison

The Intel Quartus Prime design software delivers the highest performance, optimum logic utilization, and fastest compile times for high-end FPGA designs. New algorithms and technologies enable the Intel Quartus Prime Pro Edition software to scale the range of densities and features that next-generation FPGAs offer.

The Intel Quartus Prime Pro Edition software allows you to implement a FPGA design either by using command-line executables and scripting, or by using the Intel Quartus Prime GUI.

3.1. Hardware and Software Tools for FPGA Design

The Intel Quartus Prime Pro Edition software provides tools similar to those found in the Xilinx Vivado software. The following table shows Xilinx tool suites and the Intel FPGA tool targeting similar design needs:

<table>
<thead>
<tr>
<th>Table 5. Hardware and Software Tools Available</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Xilinx</strong></td>
</tr>
<tr>
<td>Vivado HL Design Edition</td>
</tr>
<tr>
<td>• UltraScale</td>
</tr>
<tr>
<td>• UltraScale+</td>
</tr>
<tr>
<td>• 7 Series</td>
</tr>
<tr>
<td>• Spartan-6</td>
</tr>
<tr>
<td>• Virtex-6</td>
</tr>
<tr>
<td>• CoolRunner*</td>
</tr>
<tr>
<td>• Previous generations</td>
</tr>
<tr>
<td>Vivado HL WebPACK Edition</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SDAccel Environment</td>
</tr>
<tr>
<td>• Selected UltraScale and 7 Series devices.</td>
</tr>
<tr>
<td>SDSoC Environment</td>
</tr>
</tbody>
</table>

continued...
### 3. FPGA Tools Comparison

<table>
<thead>
<tr>
<th>Xilinx</th>
<th>Intel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Development Kit</td>
<td>Nios® II Embedded Design Suite (EDS)</td>
<td>Comprehensive development package to develop and debug code for SoCs</td>
</tr>
<tr>
<td>System Generator(6) for DSP</td>
<td>DSP Builder for Intel FPGAs(7)(8)</td>
<td>DSP Design Tool</td>
</tr>
<tr>
<td>Vivado High-Level Synthesis</td>
<td>Intel HLS Compiler</td>
<td>High-Level synthesis tool that takes in untimed software code as input and generates register-transfer level code for FPGAs.</td>
</tr>
</tbody>
</table>

#### Related Information
- Intel FPGA Development Tools
- Xilinx Vivado Design Suite

#### 3.2. FPGA Design Flow Using Command Line Scripting

Automating the FPGA design process saves time and increases productivity. The Vivado software and the Intel Quartus Prime Pro Edition software provide the tools necessary to automate your FPGA design flow.

The compilation flow is the sequence and methods by which the software translates design files, maps the translated design to device-specific elements, places and routes the design in the device, and generates programming files. The Intel Quartus Prime Pro Edition software performs these functions through stages such as Analysis and Synthesis, Fitter, Assembler, and Timing Analyzer. If you are familiar with the command-line flow in the Vivado software, you can detect parallels with the Intel Quartus Prime Pro Edition software.

The following figure shows the typical stages of the compilation flow and the corresponding Xilinx Vivado and Intel FPGA Intel Quartus Prime Pro Edition command line instructions for each stage.

---

(6) To use System Generator for DSP in Vivado, you must buy the Vivado HL System Edition, which supports all features of the Vivado HL Design Edition plus System Generator for DSP.

(7) The Intel Quartus Prime Pro Edition software includes the DSP Builder for Intel FPGAs.

### Figure 1. Basic Design Flow Stages and Command Line Instructions

<table>
<thead>
<tr>
<th>Xilinx Vivado Software</th>
<th>Project Creation</th>
<th>Intel Quartus Prime Pro Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_ip</td>
<td>Read IP</td>
<td>quartus_ipgenerate</td>
</tr>
<tr>
<td>synth_design -rtl</td>
<td>RTL Analysis</td>
<td>quartus_syn --analysis_and_elaboration</td>
</tr>
<tr>
<td>synth_design</td>
<td>Synthesis</td>
<td>quartus_syn</td>
</tr>
<tr>
<td>opt_design</td>
<td>Implementation</td>
<td>quartus_fit --plan</td>
</tr>
<tr>
<td>place_design</td>
<td></td>
<td>quartus_fit --early_place</td>
</tr>
<tr>
<td>route_design</td>
<td></td>
<td>quartus_fit --place</td>
</tr>
<tr>
<td></td>
<td></td>
<td>quartus_fit --route</td>
</tr>
<tr>
<td></td>
<td></td>
<td>quartus_fit --retime*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>quartus_fit --finalize</td>
</tr>
<tr>
<td>N/A</td>
<td>Timing Closure Recommendations</td>
<td>quartus_fit --fastforward*</td>
</tr>
<tr>
<td>report_timing_summary</td>
<td>Report Timing Summary</td>
<td>quartus_staa</td>
</tr>
<tr>
<td>report_power</td>
<td>Report Power (Optional)</td>
<td>quartus_pow</td>
</tr>
<tr>
<td>write_bitstream</td>
<td>Program and Debug</td>
<td>quartus_asm</td>
</tr>
<tr>
<td>program_hw_device</td>
<td></td>
<td>quartus_pgm</td>
</tr>
</tbody>
</table>

* Retime and Fast Forward compilation available only for Intel® Stratix® 10 devices. Fast Forward generates detailed reports that estimate performance gains achievable.
The Hyper-Aware Design Flow

The Hyper-Aware Design Flow allows you to take full advantage of the Intel Stratix 10 Intel Hyperflex™ architecture. This flow combines automated register retiming (Hyper-Retiming), with implementation of target timing closure recommendations (Fast Forward compilation), to maximize use of Hyper-Registers and drive the highest performance for Intel Stratix 10 designs:

- **Hyper-Retiming:**
  A key innovation of the Intel Stratix 10 architecture is the addition of multiple Hyper-Registers in every routing segment and block input. Maximizing the use of Hyper-Registers improves design performance. The prevalence of Hyper-Registers improves balance of time delays between registers and mitigates critical path delays.

- **Fast Forward Compilation:**
  If you require optimization beyond Hyper-Retiming, run Fast Forward compilation to generate timing closure recommendations that break key performance bottlenecks. Fast Forward compilation shows precisely where to make the most impact with RTL changes, and reports the performance benefits you can expect from each change.

**Related Information**

- Hyper-Aware Design Flow on page 41
- Compilation Flows
  - In *Intel Quartus Prime Pro Edition Handbook Volume 1*
- Command Line Scripting
- System Debugging Tools Overview
  - In *Intel Quartus Prime Pro Edition Handbook Volume 3*

### 3.2.1. Command-Line Executable Equivalents

The table and following sections describe and compare the two software flows using command line executables. The examples belong to the fir_filter design, included in the Intel Quartus Prime Pro Edition installation.

**Table 6. Command-Line Executable Equivalents**

<table>
<thead>
<tr>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-Project Mode</strong></td>
<td><strong>Project Mode</strong></td>
<td></td>
</tr>
<tr>
<td>read_ip</td>
<td>add_files</td>
<td>quartus_ipgenerate</td>
</tr>
<tr>
<td></td>
<td>import_files</td>
<td></td>
</tr>
<tr>
<td>synth_design</td>
<td>launch_runs</td>
<td>quartus_syn</td>
</tr>
<tr>
<td>opt_design</td>
<td>synth_1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>place_design</td>
<td>launch_runs</td>
<td>quartus_fit (full compile) OR quartus_fit --plan</td>
</tr>
<tr>
<td>phys_opt_design(9)</td>
<td>implant_1</td>
<td></td>
</tr>
<tr>
<td>route_design</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*continued...*
### 3. FPGA Tools Comparison

<table>
<thead>
<tr>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non-Project Mode</strong></td>
<td><strong>Project Mode</strong></td>
<td></td>
</tr>
<tr>
<td>phys_opt_design(^9)</td>
<td><code>quartus_fit --early_place\(^9\)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>quartus_fit --place</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>quartus_fit --route</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>quartus_fit --finalize</code></td>
<td></td>
</tr>
<tr>
<td>Not Available</td>
<td><code>quartus_fit --retime\(^10\)</code></td>
<td>Enabled by default, and runs before finalizing. Moves existing registers into Hyper-Registers for fine-grained performance improvement.</td>
</tr>
<tr>
<td>Not Available</td>
<td><code>quartus_fit --fastforward\(^10\)</code></td>
<td>Disabled by default. Runs after retime but before finalize. Generates detailed reports that estimate performance gains achievable by making specific RTL modifications.</td>
</tr>
<tr>
<td>report_timing</td>
<td><code>quartus_sta</code></td>
<td>Performs a static timing analysis on the design.</td>
</tr>
<tr>
<td>write_bitstream</td>
<td><code>quartus_asm</code></td>
<td>Generates programming file from post-place-and-route design.</td>
</tr>
<tr>
<td>report_power</td>
<td><code>quartus_pow</code></td>
<td>Performs power estimation on the design.</td>
</tr>
<tr>
<td>write_sdf</td>
<td><code>quartus_edad</code></td>
<td>Generates output netlist files for use with other EDA tools.</td>
</tr>
<tr>
<td>write_verilog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write_vhdl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>write_checkpoint</td>
<td><code>quartus_cdb</code></td>
<td>Saves the snapshot of the design database.</td>
</tr>
<tr>
<td>Not Available</td>
<td><code>quartus_sh --flow compile</code></td>
<td>Automates the compilation flow.</td>
</tr>
</tbody>
</table>

For command line help on any of the Intel Quartus Prime executables, type `<command-line executable> --help` at the command prompt. A GUI-enabled help browser is also available that covers all Intel Quartus Prime command-line executables.

\(^9\) Optional

\(^10\) Retime and Fast-Forward Compilation available only for Intel Stratix 10 devices.
3.2.1.1. synth_design

In the Intel Quartus Prime Pro Edition command-line flow, the `quartus_syn` executable performs both synthesis (`synth_design`) and mapping of design elements to device resources (`opt_design`).

The following command runs logic synthesis and technology mapping of a design named `filtref`:

```
quartus_syn filtref --rev=<revision-name>
```

**Note:**
For command line help, type `quartus_syn --help` at the command prompt.

In addition, you can recompile only the partitions that you change, instead of compiling the whole design.

```
quartus_sh --flow recompile filtref
```

For command line help, type `quartus_syn --help` at the command prompt.

3.2.1.2. place_design/route_design

Depending on the use mode, the Vivado software provides different commands to place and route device resources into the FPGA device. In Project Mode, the `launch_runs impl 1` executable performs place and route, and the equivalent Intel Quartus Prime Pro Edition executable is `quartus_fit`. In Non-Project Mode, the

---

(11) Rapid Recompile is not available for Intel Stratix 10 devices.
Vivado software provides the `place_design` and `route_design` executables. The Intel Quartus Prime Pro Edition software allows you to perform place and route stages separately in the `quartus_fit` executable through arguments.

The Intel Quartus Prime Pro Edition Fitter includes the following stages:

- **Plan**—places all periphery elements (such as I/Os and PLLs) and determines a legal clock plan, without core placement or routing.
- **Early Place**—places all core elements in an approximate location to facilitate design planning. Finalizes clock planning for Intel Stratix 10 designs.
- **Place**—places all core elements in a legal location.
- **Route**—creates all routing between the elements in the design.
- **Retime**—performs register retiming and moves existing registers into Hyper-Registers to increase performance by removing retiming restrictions and eliminating critical paths.
- **Finalize**—for Intel Arria 10 and Intel Cyclone 10 GX devices, converts unnecessary tiles to High-Speed or Low-Power. For Intel Stratix 10 devices, performs post-route.
- **Fast Forward**—generates detailed reports that estimate performance gains achievable by making specific RTL modifications.

You can run each Fitter stage standalone by providing the appropriate argument to the `quartus_fit` executable. For more information, run `quartus_fit --help`.

The following example performs place-and-route by fitting the logic of the Intel Quartus Prime Pro Edition `filtref` project:

```
quartus_fit filtref
```

For command line help, type `quartus_fit --help` at the command prompt.

### 3.2.1.3. report_timing

In place of the `report_timing` executable that the Vivado software provides for performing a static timing analysis on your design, the Intel Quartus Prime Pro Edition software provides the `quartus_sta` executable.

To specify timing constrains, the Intel Quartus Prime Pro Edition software uses the industry standard Synopsys* Design Constraint (SDC) file format. The Xilinx’s Design Constraint File (.xdf) constraint format is based on the SDC format. For details on converting XDC to SDC files, refer to the *Timing Constraints* section.

This example performs timing analysis on the `filtref` project using the SDC timing constraints file, `filtref.sdc`, to determine whether the design meets the timing requirements:

```
quartus_sta filtref --sdc=filtref.sdc
```

For command line help, type `quartus_sta --help` at the command prompt.

---

(12) Retime and Fast-Forward Compilation available only for Intel Stratix 10 devices.
3.2.1.4. write_bitstream

The Vivado software provides the `write_bitstream` executable to generate FPGA programming files. The Intel Quartus Prime software provides the `quartus_asm` executable to generate programming files for FPGA configuration.

The following example creates the `filtref.sof` programming file for the `filtref` project:
```
quartus_asm filtref
```

For command line help, type `quartus_asm --help` at the command prompt.

3.2.1.5. write_sdf/write_verilog/write_vhdl

In Vivado, the `write_sdf` executable reads data from design files, and writes timing delays in `.sdf` files. The `write_verilog` executable uses this output and generates the netlists for third-party tools. Similarly, the Intel Quartus Prime Pro Edition software provides the `quartus_eda` executable to generate netlists and other output files for use with third-party EDA tools.

The following example creates the `filtref.vo` simulation Verilog HDL netlist file, that you can use to simulate the `filtref` project with ModelSim*:
```
quartus_eda filtref --simulation=on --format=verilog --tool=modelsim
```

For command line help, type `quartus_eda --help` at the command prompt.

3.2.1.6. report_power

The `report_power` executable provides power and thermal estimates after place and route to estimate a design’s power consumption. Similarly, `quartus_pow` estimates the thermal dynamic and thermal static power that a design consumes.

The following example uses a `.vcd` file as input to perform power analysis and filter glitch on the `filtref` project:
```
quartus_pow filtref --input_vcd=<vcd filename> --vcd_filter_glitches=on
```

For command line help, type `quartus_pow --help` at the command prompt.

Related Information

Value Change Dump (VCD) Support on page 82

3.2.1.7. write_checkpoint

In Vivado, the `write_checkpoint` command allows you to save a project at any point in the design process. In the Intel Quartus Prime Pro Edition software, you can export the results of a compilation at various stages of compilation flow using the `quartus_cdb` executable.
In addition, the `quartus_cdb` executable allows you to import and export version-compatible databases. This ability simplifies design migration between versions of the Intel Quartus Prime Pro Edition software; you can import a database from a version of the Intel Quartus Prime Pro Edition into another version of the software, without the need of full compilation. After import, you only need to rerun timing analysis or simulation with the updated timing models.

### 3.2.1.8. Run Complete Design Flow

The Intel Quartus Prime Pro Edition shell (`quartus_sh`) provides the `--flow` option, that allows you to perform complete compilation of a design project, including synthesis, implementation, timing analysis and bitfile generation. The Xilinx Vivado software does not have a similar command.

The following example runs compilation, timing analysis, and programming file generation with a single command:

```
quartus_sh --flow compile filtref
```

For command line help, type `quartus_sh --help=flow` at the command prompt.

**Related Information**

Intel Quartus Prime Scripting Reference Manual

### 3.2.2. Programming and Configuration File Support in the Intel Quartus Prime Pro Edition Software

The Intel Quartus Prime Pro Edition software requires different programming and configuration files based on the type of device and configuration mode.

The table lists the programming and configuration file formats that the Intel FPGAs, CPLDs, and configuration devices support.

**Table 7. Programming and Configuration File Format**

<table>
<thead>
<tr>
<th>File Format</th>
<th>FPGA</th>
<th>CPLD</th>
<th>Configuration Device</th>
<th>Serial Configuration Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM Object File (.sof)</td>
<td>Yes</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Programmer Object File (.pof)</td>
<td>—</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>JEDEC JESD71 STAPL Format File (.jam)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>—</td>
</tr>
<tr>
<td>Jam Byte Code File (.jbc)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>—</td>
</tr>
</tbody>
</table>

**Related Information**

Programming Intel FPGA Devices

In *Intel Quartus Prime Pro Edition Handbook Volume 3*

### 3.3. FPGA Design Flow Using Tools with GUIs

The Intel Quartus Prime Pro Edition and the Vivado software GUIs address the major FPGA design steps in different ways.
Table 8. GUI Feature Equivalents

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Creation</td>
<td>New Project</td>
<td>New Project Wizard</td>
</tr>
<tr>
<td>Design Entry</td>
<td>HDL Editor</td>
<td>HDL Editor</td>
</tr>
<tr>
<td></td>
<td>EDA Netlist</td>
<td>EDA Netlist</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>Schematic/Block Editor</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>State Machine Editor</td>
</tr>
<tr>
<td>IP Catalog</td>
<td>IP Catalog and Parameter Editor</td>
<td></td>
</tr>
<tr>
<td>IP Integrator</td>
<td>Platform Designer System Integration Tool</td>
<td></td>
</tr>
<tr>
<td>IP Packager</td>
<td>Platform Designer Component Editor</td>
<td></td>
</tr>
<tr>
<td>IP Status</td>
<td>Report IP Status</td>
<td>Upgrade IP Components</td>
</tr>
<tr>
<td>Design Constraints</td>
<td>Device, Physical and Timing Constraints window</td>
<td>Assignment Editor, Timing Analyzer Text Editor</td>
</tr>
<tr>
<td>Synthesis</td>
<td>Synthesis</td>
<td>Analysis and Synthesis</td>
</tr>
<tr>
<td></td>
<td>Third-Party EDA Synthesis</td>
<td>Third-Party EDA Synthesis</td>
</tr>
<tr>
<td>Design Implementation</td>
<td>Implementation</td>
<td>Fitter (Plan, Early Place, Place, Route, Retime(^{13}) and Finalize)</td>
</tr>
<tr>
<td>Finalize Pinout</td>
<td>Byte Planner for memory banks</td>
<td>Interface Planner</td>
</tr>
<tr>
<td></td>
<td>Device Window and Package Window in I/O Planning View Layout</td>
<td>Pin Planner</td>
</tr>
<tr>
<td>Viewing and Editing Design Placement</td>
<td>Device Window (in I/O Planner View Layout)</td>
<td>Chip Planner</td>
</tr>
<tr>
<td></td>
<td>Package Window (in I/O Planner View Layout)</td>
<td></td>
</tr>
<tr>
<td>Static Timing Analysis</td>
<td>Report Timing</td>
<td>Timing Analyzer</td>
</tr>
<tr>
<td>Generation of Device Programming Files</td>
<td>Hardware Manager</td>
<td>Assembler</td>
</tr>
<tr>
<td>Power Analysis</td>
<td>Xilinx Power Estimator (XPE)</td>
<td>Early Power Estimation (EPE)</td>
</tr>
<tr>
<td></td>
<td>Report Power</td>
<td>Power Analyzer</td>
</tr>
<tr>
<td>Simulation</td>
<td>Vivado Simulator</td>
<td>ModelSim - Intel FPGA Starter Edition</td>
</tr>
<tr>
<td></td>
<td>Third-Party Simulation Tools</td>
<td>Third-Party Simulation Tools</td>
</tr>
<tr>
<td>Hardware verification</td>
<td>Hardware Manager</td>
<td>System Console</td>
</tr>
<tr>
<td></td>
<td>Integrated Logic Analyzer (ILA) and System ILA IP</td>
<td>Signal Tap Logic Analyzer</td>
</tr>
<tr>
<td></td>
<td>Xilinx Virtual Input Output (VIO)</td>
<td>In-System Sources and Probes</td>
</tr>
<tr>
<td></td>
<td>JTAG-to-AXI Master</td>
<td>System Console</td>
</tr>
<tr>
<td></td>
<td>IBERT IP and Serial I/O Analyzer Tool</td>
<td>Transceiver Toolkit</td>
</tr>
<tr>
<td></td>
<td>Memory Calibration Debug Tool</td>
<td>EMIF Debug Toolkit</td>
</tr>
<tr>
<td></td>
<td>EMIF Debug GUI</td>
<td></td>
</tr>
</tbody>
</table>

\(^{13}\) Available only for Intel Arria 10 devices.
3. FPGA Tools Comparison

### 3.3.1. Project Creation

<table>
<thead>
<tr>
<th>Table 9.</th>
<th><strong>IP Status Comparison</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GUI Feature</strong></td>
<td><strong>Xilinx Vivado Software</strong></td>
</tr>
<tr>
<td>New Project</td>
<td>New Project</td>
</tr>
</tbody>
</table>

Similar to the **New Project** command in the Vivado software, the Intel Quartus Prime Pro Edition software provides the **New Project Wizard** tool ([File ➤ New Project Wizard](#)), which guides you through specifying a project name and directory, top-level design entity, any EDA tools you are using, and a target device.

---

(14) Signal Probe available only for Intel Arria 10 devices.

(15) Hyper-Aware Design Flow available only for Intel Stratix 10 devices.

(16) Rapid Recompile available only for Intel Arria 10 devices.
Comparison

After creating a new project, the Intel Quartus Prime Pro Edition software automatically generates the following project files necessary for successful compilation:

Table 10. Project Files Comparison

<table>
<thead>
<tr>
<th></th>
<th>Intel Quartus Prime Pro Edition</th>
<th>Xilinx Vivado</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Type</strong></td>
<td><strong>Description</strong></td>
<td><strong>File Type</strong></td>
</tr>
<tr>
<td>Project File</td>
<td>Intel Quartus Prime Project File (.qpf)</td>
<td>Xilinx Project File (.xpr)</td>
</tr>
<tr>
<td>Project Settings</td>
<td>Intel Quartus Prime Settings File (.qsf)</td>
<td>Lists design files, entity settings, target device, synthesis directives, placement constraints</td>
</tr>
</tbody>
</table>

Features

You can modify the compiler settings by changing the assignments through the GUI or directly on the .qsf file.

*Note:* Avoid modifying assignments through the .qsf file and through the GUI simultaneously.

3.3.2. Design Entry

The Intel Quartus Prime software supports all the design entry methods that the Vivado software supports.

Table 11. Design Entry Methods Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Entry</td>
<td>HDL Editor</td>
<td>HDL Editor</td>
</tr>
<tr>
<td></td>
<td>EDA Netlist</td>
<td>EDA Netlist</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>Schematic/Block Editor</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>State Machine Editor</td>
</tr>
<tr>
<td></td>
<td>IP Catalog</td>
<td>IP Catalog and Parameter Editor</td>
</tr>
<tr>
<td></td>
<td>IP Integrator</td>
<td>Platform Designer System Integration Tool</td>
</tr>
<tr>
<td></td>
<td>IP Packager</td>
<td>Platform Designer Component Editor</td>
</tr>
</tbody>
</table>
Managing Project Files

In the Xilinx Vivado software, you use the Add Source dialog box to add or remove existing design files. To add or remove existing design files from a project in the Intel Quartus Prime software:

- Click Assignments ➤ Settings to open the Settings dialog box.
- In the Category list, select Files to open the Files page. This page allows you to add or remove files.

3.3.2.1. HDL Editor

In the Vivado software you create a new HDL design file using the New Source Wizard on the Project menu. To create a new HDL design file in the Intel Quartus Prime Pro Edition software, click File ➤ New, and then select the type of file you want to create.

- To assist you in creating HDL designs, the Intel Quartus Prime software provides design example templates for VHDL and Verilog HDL, including language constructs examples to help you get started on a design.
- The Intel Quartus Prime Text Editor offers syntax coloring for highlighting HDL reserved words and comments.

Related Information

Design Practices and HDL Coding Styles
In Intel Quartus Prime Pro Edition Handbook Volume 1

3.3.2.2. Schematic/Block Editor

In the Intel Quartus Prime Pro Edition software, you can use Intel FPGA-supplied design elements, such as Boolean gates and registers, or you can create your own symbols from HDL or EDA netlist design entities.

- To create a block design file from a VHDL or Verilog HDL design file, click File ➤ Create/Update, and click Create Symbol Files for Current File.
- To create a new schematic file (*.bdf) in the Intel Quartus Prime Pro Edition software, point to File ➤ New and select the Block Diagram/Schematic File.
- To insert block symbols into the schematic, double-click the schematic file and choose the appropriate block symbols.
3.3.2.3. State Machine Editor

The Intel 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.

3.3.2.4. IP Catalog and Parameter Editor

Use the IP Catalog to generate Intel FPGA equivalents for Xilinx primitives and IP Catalog cores. To display the IP Catalog in the Intel Quartus Prime Pro Edition software, click View ➤ IP Catalog.

Features of the IP Catalog

- Allows you to create custom IP cores that are optimized for the design's target device.
- Intel provides a library-of parameterized-modules (LPM). This library offers architecture-independent functions for all devices that the Intel Quartus Prime Pro Edition supports.
- The IP Catalog is also available from Platform Designer. The Platform Designer IP Catalog includes exclusive system interconnect, video and image processing, and other system-level IPs that are available only if you access the IP Catalog from Platform Designer. To open Platform Designer from the Intel Quartus Prime Pro Edition software, click Tools ➤ Platform Designer. To display the IP Catalog from Platform Designer, click View ➤ IP Catalog.

Related Information

- IP Catalog and Parameter Editor
  In Intel Quartus Prime Help
- Intel FPGA IP Cores/LPM
  In Intel Quartus Prime Help
3.3.2.5. Platform Designer System Integration Tool

Similar to Xilinx’s Vivado IP Integration tool, Intel FPGA provides the Platform Designer System Integration tool (Tools ➤ Platform Designer).

Features

Platform Designer enables the use of processors (such as the Intel FPGA Nios II embedded processor), interfaces to off-chip processors, standard peripherals, IP cores, on-chip memory, off-chip memory, and user-defined logic into a custom system module.

Platform Designer generates a single system module that instantiates these components and automatically generates the necessary interconnect logic to bind them together.

Migration

When migrating from Xilinx to Intel FPGA, one of the main differences to consider while creating systems is the standard bus interface:

- Xilinx uses AMBA* AXI as the standard bus interface for communication between IPs. Consequently, custom IPs that you build with Vivado software use the AMBA 4 AXI protocol.
- Even though Intel FPGA uses Avalon® as the standard bus interface, Platform Designer supports multiple AMBA AXI interfaces. For details about interface support, refer to the Intel Quartus Prime Pro Edition Handbook Volume 1.

Alternatively, Platform Designer makes the migration easier by allowing conversion from Avalon to AMBA AXI interface via AMBA AXI Agents and AMBA AXI translators.


Related Information

- Creating a System with Platform Designer
  In Intel Quartus Prime Pro Edition Handbook Volume 1
- Platform Designer Transformations
  In Intel Quartus Prime Pro Edition Handbook Volume 1

3.3.2.6. Platform Designer Component Editor

In the Vivado software, you can use the IP Packager to add custom IPs to the IP catalog. In the Intel Quartus Prime Pro Edition software, you create and package IP components with the Platform Designer Component Editor.
• To create a new component using the Component Editor, in the Platform Designer main window click **File ➤ New Component**.

• When you define a component with the Component Editor, Platform Designer writes the information to an _hw.tcl file. This file contains the component’s description, interfaces, and HDL files.

• If the component requires custom features that the Platform Designer Component Editor does not support, for example, an elaboration callback, you can use the Component Editor to create the _hw.tcl file, and then manually edit the file to complete the component definition.

**Related Information**

Creating Platform Designer Components  
In *Intel Quartus Prime Pro Edition Handbook Volume 1*

### 3.3.3. IP Status

When migrating a project in the Vivado software, the Report IP Status window displays the list of IPs and the recommendation for upgrading, among other information. The Intel Quartus Prime Pro Edition software uses the **Upgrade IP Components** window.

**Table 12. IP Status Tools Comparison**

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Status</td>
<td>Report IP Status</td>
<td>Upgrade IP Components</td>
</tr>
</tbody>
</table>

When you open a project containing outdated IP, 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.

Icons in the Upgrade IP Components dialog box indicate when IP upgrade is required, optional, or unsupported for IP cores in your design. You must upgrade IP cores that require upgrade before you can compile the IP variation in the current version of the Intel Quartus Prime Pro Edition software. The upgrade process preserves the original IP variation file in the project directory as `<my_variant>_BAK.qsys`

**Related Information**

Upgrade IP Components Dialog Box (Project Menu)  
In *Intel Quartus Prime Help*

### 3.3.4. Design Constraints

The Vivado software provides GUI editors (Device/Physical/Timing windows) to create and edit design constraints. Xilinx designs store all the constraints and attributes in Xilinx Design Constraint (.xdc) files, including timing and device constraints. You can also edit .xdc files with a text editor.

Intel FPGA designs use separate files for device (.qsf) and timing (.sdc) constraints, and stores timing constraints in Synopsys Design Constraints (SDC) format. To view and edit pin assignments, device options, and logic options, use the Assignment Editor. To view and edit timing constraints, use the Text Editor in the Timing Analyzer GUI.
### Table 13. Design Constraint Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Constraints</td>
<td>Device, Physical and Timing Constraints window</td>
<td>Assignment Editor, Timing Analyzer Text Editor</td>
</tr>
</tbody>
</table>

### Features

The table summarizes the file format and assignment types that the tools in the Intel Quartus Prime Pro Edition software set.

### Table 14. Intel Quartus Prime Pro Edition Assignment Tools

<table>
<thead>
<tr>
<th>Assignment Type</th>
<th>File Format</th>
<th>Tools to Make Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing</td>
<td>SDC</td>
<td>Timing Analyzer</td>
</tr>
<tr>
<td>I/O-related</td>
<td>TCL</td>
<td>Interface Planner</td>
</tr>
<tr>
<td>Others</td>
<td>QSF</td>
<td>Pin Planner, Interface Planner</td>
</tr>
</tbody>
</table>

With separate constraint files, you avoid searching for timing constraints among other device constraints. Additionally, you can modify the timing constraints and check for validity without recompiling. In place of the I/O Planning in the Vivado software, Intel Quartus Prime Pro Edition software offers the Interface Planner to plan interfaces and device periphery, and the Pin Planner to edit, validate, and export pin assignments.

For equivalence between design constraints, refer to the *Set Equivalent Xilinx Design Constraints* section.

### Related Information

**Constraining Designs with Intel Quartus Prime Tools**

In *Intel Quartus Prime Pro Edition Handbook Volume 2*

#### 3.3.4.1. Assignment Editor

The Intel Quartus Prime Assignment Editor (**Assignments ➤ Assignment Editor**) allows you to add device and placement constraints to a design. The Assignment Editor provides a spreadsheet-like interface for assigning all instance-specific settings and constraints.

The Intel Quartus Prime software dynamically validates changes that you make through the editor, and issues errors or warnings for invalid assignments.

The **System** tab of the Intel Quartus Prime message window acknowledges adding or changing assignments.

### Figure 4. Intel Quartus Prime Assignment Editor
3.3.4.2. Create Timing Constraints with the Timing Analyzer GUI

To create timing constraints with the Timing Analyzer GUI:

1. Create a timing netlist by clicking Netlist ➤ Create Timing Netlist.
2. Click File ➤ New SDC File to open a new SDC file.
3. From the Constraints menu, select the constraint you want to add.
   
   The selected constraint's dialog box opens, and allows you to set the constraint's parameters.

   \[\text{Figure 5. Example: Create Clock Dialog Box}\]

4. Enter the values in the dialog box, and click Insert to insert the SDC command into the open SDC file.
5. Save the updated SDC file.

   The constraints are available on the Constraint menu are:

   - Create Clock
   - Create Generated Clock
   - Set Clock Latency
   - Set Clock Uncertainty
   - Set Clock Groups
   - Remove Clock
   - Set Input Delay
   - Set Output Delay
   - Derive PLL Clocks
   - Derive Clock Uncertainty
   - Set False Path
   - Set Multicycle Path
   - Set Maximum Delay
   - Set Minimum Delay
   - Set Net Delay

3.3.4.3. Create Timing Constraints with the Timing Analyzer Text Editor

The Intel Quartus Prime Timing Analyzer (Tools ➤ Timing Analyzer) reads and writes timing constrains in the industry-standard Synopsys Design Constraint (SDC) format. The Timing Analyzer provides a GUI interface that allows you to manually create and modify timing constraints.
Note: Ensure that the project is open before using the Timing Analyzer.

- To create a new SDC file from the Timing Analyzer, click File ➤ New SDC File.
- To apply templates for SDC constraints, click Edit ➤ Insert Template command.

The Timing Analyzer Text Editor uses syntax coloring for SDC reserved words and comments.

Figure 6. Timing Analyzer Text Editor

Related Information
- Timing Analyzer Cookbook
- The Intel Quartus Prime Timing Analyzer In Intel Quartus Prime Pro Edition Handbook Volume 3

3.3.5. Synthesis

The Intel Quartus Prime Pro Edition Synthesis provides full support for VHDL, Verilog HDL, SystemVerilog, and Block Design File (.bdf) schematic entry.

Table 15. Synthesis Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthesis</td>
<td>Synthesis</td>
<td>Analysis and Synthesis</td>
</tr>
<tr>
<td></td>
<td>Third-Party EDA Synthesis</td>
<td>Third-Party EDA Synthesis</td>
</tr>
</tbody>
</table>
Features

The Intel Quartus Prime Pro Edition Synthesis engine enforces strict industry-standard HDL structures. For the 17.1 release, The Intel Quartus Prime Pro Edition Synthesis supports the following enhancements:

- Support for modules with System Verilog Interfaces
- Improved support for VHDL2008
- New RAM inference engine that infers RAMs from GENERATE statements or array of integers
- Stricter syntax/semantics check for improved compatibility with other EDA tools
  - Does not support the third-party netlists as input, but can generate netlist that other EDA tools use

At the end of synthesis, the Compiler generates an atom netlist, which is a database of the atom elements that design synthesis requires to implement the design in silicon. The Analysis & Synthesis module of the Compiler creates one or more project databases for each design partition. You can specify various settings that affect synthesis processing.

Access

The Assignments ➤ Settings ➤ IP Settings dialog box allows you to control the IP regeneration stage for synthesis or simulation.

The Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Synthesis) dialog box allows you to set options that affect the analysis and synthesis stage of the compilation flow. These options include Optimization Technique, State Machine Processing, Restructure Multiplexers, and others.

Related Information

- Design Synthesis
  In Intel Quartus Prime Pro Edition Handbook Volume 1
- Selecting Third-Party EDA Tools
  In Intel Quartus Prime Pro Edition Handbook Volume 1

3.3.6. Design Implementation

The implementation flow in the Vivado software places and routes the netlist onto the FPGA device resources based on the constraints of the design. The Finalize flow in the Intel Quartus Prime Pro Edition software consists of the Plan, Place, Route, Retime\(^{(17)}\), and Finalize compilation stages. Start the Finalize flow by clicking Fitter in the Compilation Dashboard.

### Table 16. Design Implementation Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Implementation</td>
<td>Implementation</td>
<td>Fitter (Plan, Early Place, Place, Route, Retime(^{(17)}) and Finalize)</td>
</tr>
</tbody>
</table>

\(^{(17)}\) Retime and Fast-Forward Compilation available only for Intel Stratix 10 devices.
Features
The Intel Quartus Prime Pro Edition Compiler offers unique features, such as:

- **Incremental Fitter Optimizations**—run and optimize Fitter stages incrementally. Each Fitter stage generates detailed reports. You can view detailed report data and analyze the timing of each stage while downstream stages are still running.

- **Hyper-Aware Design Flow**—use Hyper-Retiming and Fast Forward compilation for the highest performance in Intel Stratix 10 devices.

You can start each phase in the compilation flow independently either from GUI or from the command line. The Compilation Dashboard allows you to use the tools and features of the software and monitor progress from a flow-based layout.

Access
The Assignments ➤ Settings ➤ Compiler Settings ➤ Advanced Settings (Fitter) dialog box allows you customize the place and route of the compilation flow.

Related Information
- **Compilation Dashboard**
  In *Intel Quartus Prime Help*
- **Fitter Settings Reference**
  In *Intel Quartus Prime Pro Edition Handbook Volume 1*

### 3.3.7. Finalize Pinout

In the Vivado software, you can use the I/O Planning View Layout to finalize the pinout. For I/O planning of Memory Interfaces, the Vivado software uses the Memory Bank/Byte Planner.

Intel Quartus Prime Pro Edition Software provides the Interface Planner and the Pin Planner to help you with the I/O Planning.

| Table 17. Finalize Pinout Comparison |
|-------------------------------|-----------------------------------|-----------------------------------|
| **GUI Feature** | **Xilinx Vivado Software** | **Intel Quartus Prime Pro Edition Software** |
| Finalize Pinout | Byte Planner for memory banks Device Window and Package Window in I/O Planning View Layout | Interface Planner Pin Planner |

### 3.3.7.1. Pin Planner

The Intel Quartus Prime Pro Edition Pin Planner provides a graphical package view, allowing you to validate I/O assignments by performing legality checks on a design's I/O pins and surrounding logic. With the Pin Planner, you can identify I/O banks, VREF groups, and differential pin pairings to help you with the I/O planning process. To access the Pin Planner, click **Assignments ➤ Pin Planner**.

*Note:* Modifications that you make in the Pin Planner affect the .qsf file.
In the Intel Quartus Prime Pro Edition software, the Interface Planner tool (Tools ➤ Interface Planner) simplifies the planning of accurate constraints for physical implementation after synthesis, with features such as:

- Prototype interface implementations
- Plan clocks
- Rapidly define a legal device floorplan
Figure 8. Interface Planner GUI

Drag or click Design Elements to place in legal locations

Flow Controls provide quick access to planning functions

Selected element’s properties

Tcl Console window

Undo/Redo Placement History

Related Information

Interface Planning
In Intel Quartus Prime Pro Edition Handbook Volume 2

3.3.8. Viewing and Editing Design Placement

The Vivado software provides the Device Window for floorplanning and design analysis. In the Intel Quartus Prime Pro Edition software, the Chip Planner simplifies floorplan analysis by providing visual display of chip resources.

Table 18. Design Placement Methods Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewing and Editing Design Placement</td>
<td>Device Window (in I/O Planner View Layout) Package Window (in I/O Planner View Layout)</td>
<td>Chip Planner</td>
</tr>
</tbody>
</table>

With the Chip Planner, you can view post-compilation placement, connections, and routing paths. You can also make assignment changes, such as creating and deleting resource assignments.
Figure 9. **Chip Planner**

![Chip Planner](image)

To open the Chip Planner, click **Tools ➤ Chip Planner**.

### 3.3.9. Static Timing Analysis

The Report Timing Summary in Vivado generates the Post-Place and Post-Route Static Timing Report. Similarly, the Intel FPGA Timing Analyzer analyzes and reports the performance of all logic in your design, allowing you to determine all the critical paths that limit your design’s performance.

**Table 19.** **Static Timing Analysis Methods Comparison**

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Timing Analysis</td>
<td>Report Timing</td>
<td>Timing Analyzer</td>
</tr>
</tbody>
</table>

The Intel FPGA Timing Analyzer is an easy-to-use, second-generation, ASIC-strength static timing analyzer that supports the industry-standard Synopsys Design Constraints (SDC) format.
The major difference between performing timing analysis with the Report Timing Summary in Vivado and the Intel FPGA Timing Analyzer is that in the Vivado software, a change in timing constraint triggers a recompile. In contrast, the Timing Analyzer GUI allows you to experiment with timing constraints and timing model without recompiling.

**Access**

Static timing analysis with the Timing Analyzer is part of the full compilation flow, but you can also run the module separately.

To run the Timing Analyzer over a post-fit netlist, click **Processing ➤ Start ➤ Start Timing Analyzer**.

To open the **Timing Analyzer GUI**, click **Tools ➤ Timing Analyzer**.

**Related Information**

- Timing Analyzer Cookbook
- The Intel Quartus Prime Timing Analyzer
  
  **In Intel Quartus Prime Pro Edition Handbook Volume 3**

### 3.3.10. Generation of Device Programming Files

Similar to the Hardware Manager in the Xilinx Vivado software, the Assembler in the Intel Quartus Prime Pro Edition software generates files that the Programmer can use to program or configure a device with Intel FPGA programming hardware.
Table 20. Methods to Generate Programming Files Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generation of Device Programming Files</td>
<td>Hardware Manager</td>
<td>Assembler</td>
</tr>
</tbody>
</table>

**Features**

The Assembler converts the Fitter’s device, logic cell, and pin assignments into a programming image for the device, in the form of one or more Programmer Object Files (.pof) or SRAM Object Files (.sof) for the target device. You use a .sof file to program Intel FPGA devices and a .pof file to configure Intel FPGA CPLD devices.

Assembler is a stage of the Intel Quartus Prime Pro Edition full compilation flow. You can also run Assembler separately, by clicking **Processing ➤ Start ➤ Start Assembler**.

**Related Information**

Programming Intel FPGA Devices

*In Intel Quartus Prime Pro Edition Handbook Volume 3*

3.3.11. Power Analysis

Similar to the Xilinx Power Estimator and Report Power tool, Intel provides an Early Power Estimator (EPE) and Power Analyzer tool that allows you to estimate power consumption from early design concept through design implementation.

Table 21. Power Analysis Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Analysis</td>
<td>Xilinx Power Estimator (XPE) Report Power</td>
<td>Early Power Estimation (EPE) Power Analyzer</td>
</tr>
</tbody>
</table>

The EPE is a spreadsheet tool that helps you estimate power consumption at early design concept. You download the EPE tool from the Early Power Estimators (EPE) and Power Analyzer page in the Altera website.

The Intel Quartus Prime Pro Edition Power Analyzer tool performs post-fitting power analysis and generates a report that details power consumption of the design by block type and entity. To open the Power Analyzer tool, click **Processing ➤ Power Analyzer Tool**.

**Related Information**

- Early Power Estimators (EPE) and Power Analyzer
- Power Optimization
  *In Intel Quartus Prime Pro Edition Handbook Volume 2*
- Power Analysis
  *In Intel Quartus Prime Pro Edition Handbook Volume 3*
3.3.12. Simulation

Both the Xilinx Vivado and the Intel Quartus Prime Pro Edition software support integration with third-party EDA simulation tools, such as Mentor Graphics* ModelSim, Cadence NC-Sim, and Synopsys VCS. In addition, the Intel Quartus Prime Pro Edition software supports the Aldec Active-HDL and Riviera-PRO simulation tools.

Table 22. Simulation Support Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>Vivado Simulator</td>
<td>ModelSim - Intel FPGA Starter Edition</td>
</tr>
<tr>
<td></td>
<td>Third-Party Simulation Tools</td>
<td>Third-Party Simulation Tools</td>
</tr>
</tbody>
</table>

Access

To specify third-party simulation tools in the Intel Quartus Prime Pro Edition software:

1. Click Assignments ➤ Settings.
2. In Category, click EDA Tool Settings.
3. Under Simulation select the simulation tool.

You can also specify third-party simulation tools in the New Project Wizard.

3.3.12.1. Simulation Models for Designs Containing LPMs or IP Cores

Functional Simulation

The Intel Quartus Prime Pro Edition software provides functional simulation models that allow you to perform functional/behavioral simulation on designs containing LPMs or Intel FPGA IP cores.

Table 23. Simulation Model Files

<table>
<thead>
<tr>
<th></th>
<th>Verilog HDL</th>
<th>VHDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPM</td>
<td>220model.v</td>
<td>220pack.vhd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>220model.vhd</td>
</tr>
<tr>
<td>Intel FPGA IP cores</td>
<td>altera_mf.v</td>
<td>altera_mf.vhd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>altera_mf_components.vhd</td>
</tr>
</tbody>
</table>

Gate-level Functional Simulation

To perform gate-level functional simulation on a design, the Intel Quartus Prime Pro Edition software generates output netlist files containing information about how the design was placed into device-specific architectural blocks.

Table 24. Generated Output Files

<table>
<thead>
<tr>
<th></th>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verilog HDL output file</td>
<td>.vo</td>
</tr>
<tr>
<td>VHDL output file</td>
<td>.vho</td>
</tr>
</tbody>
</table>
You can perform simulations with pre-compiled model libraries by using the ModelSim - Intel FPGA Edition simulator included in the Intel Quartus Prime Pro Edition software. You can also compile your own selection of model libraries with the Simulation Library Compiler tool in the Intel Quartus Prime Pro Edition software.

### 3.3.13. Hardware Verification

Xilinx offers solutions for hardware debugging that have equivalent tools in Intel FPGA software. In contrast, not all Intel FPGA debug solutions have an equivalent Xilinx tool.

#### Table 25. Hardware Verification Tools Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware verification</td>
<td>Hardware Manager</td>
<td>System Console</td>
</tr>
<tr>
<td></td>
<td>Integrated Logic Analyzer (ILA) and System ILA IP</td>
<td>Signal Tap Logic Analyzer</td>
</tr>
<tr>
<td></td>
<td>Xilinx Virtual Input Output (VIO)</td>
<td>In-System Sources and Probes</td>
</tr>
<tr>
<td></td>
<td>JTAG-to-AXI Master</td>
<td>System Console</td>
</tr>
<tr>
<td></td>
<td>IBERT IP and Serial I/O Analyzer Tool</td>
<td>Transceiver Toolkit</td>
</tr>
<tr>
<td></td>
<td>Memory Calibration Debug Tool</td>
<td>EMIF Debug Toolkit</td>
</tr>
<tr>
<td></td>
<td>Remote Debug using Xilinx Virtual Cable (XVC)</td>
<td>Remote Debug using existing TCP/IP connection</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>Signal Probe&lt;sup&gt;(18)&lt;/sup&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In-System Memory Content Editor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Logic Analyzer Interface (LAI)</td>
</tr>
</tbody>
</table>

#### 3.3.13.1. System Console

Xilinx Vivado software's Hardware Manager provides a TCL console to interact with the debug IP on the hardware. Similarly, in the Intel Quartus Prime Pro Edition Software, you can perform the same tasks using the System Console.

#### Table 26. System Console Features and Usage

<table>
<thead>
<tr>
<th>Features</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Provides real-time in-system debugging capabilities.</td>
<td>You need to perform system-level debugging.</td>
</tr>
<tr>
<td>• Allows you to read from and write to Memory Mapped components in a system without a processor or additional software</td>
<td>For example, if you have an Avalon-MM slave or Avalon-ST interfaces, you can debug the design at a transaction level.</td>
</tr>
<tr>
<td>• Communicates with hardware modules in a design through a Tcl interpreter.</td>
<td></td>
</tr>
<tr>
<td>• Allows you to take advantage of all the features of the Tcl scripting language.</td>
<td></td>
</tr>
<tr>
<td>• Supports JTAG and TCP/IP connectivity.</td>
<td></td>
</tr>
</tbody>
</table>

**Related Information**

Analyzing and Debugging Designs with System Console
In *Intel Quartus Prime Pro Edition Handbook Volume 3*

<sup>(18)</sup> Signal Probe available only for Intel Arria 10 devices.
3.3.13.2. Signal Tap Logic Analyzer

The Vivado software includes the Integrated Logic Analyzer (ILA) feature to debug post-implemented designs on a FPGA. Similarly, the Intel Quartus Prime provides the Signal Tap Logic Analyzer; a multiple-input, digital acquisition instrument that captures and stores signal activity from any internal device node or nodes. The Signal Tap Logic Analyzer helps debug an FPGA design by probing the state of the internal signals in the design without using external equipment.

Table 27. Signal Tap Logic Analyzer Features and Usage

<table>
<thead>
<tr>
<th>Features</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Uses FPGA resources.</td>
<td>You have spare on-chip memory and you want functional verification of a design running in hardware.</td>
</tr>
<tr>
<td>• Samples test nodes, and outputs the information to the Intel Quartus Prime software for display and analysis.</td>
<td></td>
</tr>
</tbody>
</table>

Related Information

Design Debugging with the Signal Tap Logic Analyzer
In Intel Quartus Prime Pro Edition Handbook Volume 3

3.3.13.3. In-System Sources and Probes

The Vivado software provides the Virtual Input/Output (VIO) debug feature to monitor and drive internal FPGA signals in real time. The equivalent in Intel FPGA software is the In-System Sources and Probes utility.

Table 28. In-System Sources and Probes Features and Usage

<table>
<thead>
<tr>
<th>Features</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides an easy way to drive and sample logic values to and from internal nodes using the JTAG interface.</td>
<td>You want to prototype the FPGA design using a front panel with virtual buttons.</td>
</tr>
</tbody>
</table>

Related Information

Design Debugging Using In-System Sources and Probes
In Intel Quartus Prime Pro Edition Handbook Volume 3

3.3.13.4. Transceiver Toolkit

The Vivado software uses IBERT IP along with the serial I/O analyzer tool to evaluate and monitor the transceivers in UltraScale devices. In the Intel Quartus Prime Pro Edition software, the Transceiver Toolkit allows you to check and improve signal integrity of high-speed serial links in Intel FPGAs.

Table 29. Transceiver Toolkit Features and Usage

<table>
<thead>
<tr>
<th>Features</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Allows you to test and tune transceiver link signal quality through a combination of metrics.</td>
<td>You need to debug or optimize signal integrity of a board layout even before finishing the design.</td>
</tr>
<tr>
<td>• Auto Sweeping of physical medium attachment (PMA) settings help you find optimal parameter values.</td>
<td></td>
</tr>
</tbody>
</table>

Related Information

Debugging Transceiver Links
In Intel Quartus Prime Pro Edition Handbook Volume 3
3.3.13.5. EMIF Debug Toolkit

In the Vivado software, the Memory Calibration Debug tool allows you to debug calibration or data errors in UltraScale memory interfaces. In the Intel Quartus Prime Pro Edition software, the External Memory Interface (EMIF) Debug Toolkit allows you to run custom traffic patterns, diagnose and debug calibration problems, and produce margining reports for the external memory interface.

Table 30. EMIF Debug Toolkit Features and Usage

<table>
<thead>
<tr>
<th>Features</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Tcl-based graphical user interface.</td>
<td>You want to debug hardware failures by accessing information gathered during calibration.</td>
</tr>
<tr>
<td>• Provides access to memory calibration data gathered by the Nios II sequencer, via a JTAG connection.</td>
<td></td>
</tr>
<tr>
<td>• Allows you to mask ranks for calibration and to request recalibration of the interface.</td>
<td></td>
</tr>
<tr>
<td>• The Driver Margining feature of the toolkit allows you to measure margins on the memory interface using a driver with arbitrary traffic patterns.</td>
<td></td>
</tr>
<tr>
<td>• The EMIF Toolkit can communicate with several different memory interfaces on the same device, but only one at a time.</td>
<td></td>
</tr>
</tbody>
</table>

Related Information

External Memory Interface Debug Toolkit

In External Memory Interface Handbook Volume 3: Reference Material

3.3.13.6. 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

3.3.13.7. Other Intel FPGA Debugging Tools

These Intel FPGA tools do not have a Xilinx tools to compare with.

Signal Probe

Table 31. Signal Probe Features and Usage

<table>
<thead>
<tr>
<th>Features</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incrementally routes internal signals to I/O pins while preserving results from the last place-and-routed design.</td>
<td>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.</td>
</tr>
</tbody>
</table>
Logic Analyzer Interface

Table 32. Logic Analyzer Interface Features and Usage

<table>
<thead>
<tr>
<th>Features</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Multiplexes a larger set of signals to a smaller number of spare I/O pins.</td>
<td>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.</td>
</tr>
<tr>
<td>• Allows you to select which signals switch onto the I/O pins over a JTAG connection.</td>
<td></td>
</tr>
</tbody>
</table>

In-System Memory Content Editor

Table 33. In-System Memory Content Editor Features and Usage

<table>
<thead>
<tr>
<th>Features</th>
<th>Typical Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displays and allows you to edit on-chip memory.</td>
<td>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.</td>
</tr>
</tbody>
</table>

Related Information
- In-System Debugging Using External Logic Analyzers
  In *Intel Quartus Prime Pro Edition Handbook Volume 3*
- In-System Modification of Memory and Constants
  In *Intel Quartus Prime Pro Edition Handbook Volume 3*

3.3.14. View Netlist

Similar to the Netlist Window and Schematic Window features available in the Vivado software to generate logical or physical hierarchy, the Intel Quartus Prime Pro Edition RTL Viewer and Technology Map Viewer provide powerful ways to view initial and fully mapped synthesis results during the debugging, optimization, and constraint entry processes.

Table 34. View Netlist Methods Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Netlist</td>
<td>Schematic Window (Elaborated)</td>
<td>RTL Viewer (Post Synthesis)</td>
</tr>
<tr>
<td></td>
<td>Schematic Window (Synthesized)</td>
<td>Technology Map Viewer (Post-Mapping)</td>
</tr>
<tr>
<td></td>
<td>Schematic Window (Implemented)</td>
<td>Technology Map Viewer (Post-Fitting)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fast Forward Viewer (Post-Fitting)</td>
</tr>
</tbody>
</table>

Related Information
- Introduction to the User Interface
  In *Intel Quartus Prime Pro Edition Handbook Volume 1*

3.3.14.1. RTL Viewer

To run the RTL Viewer for an Intel Quartus Prime Pro Edition project:
1. Click **Processing ➤ Start ➤ Start Analysis & Elaboration** to generate a RTL netlist
2. To open the RTL Viewer, click **Tools ➤ Netlist Viewers (RTL Viewer)**.
Alternatively, you can perform a full compilation on any Intel Quartus Prime Pro Edition flow that includes the initial Analysis and Elaboration stage.

**Related Information**
RTL Viewer Overview
   In *Intel Quartus Prime Pro Edition Handbook Volume 1*

### 3.3.14.2. Technology Map Viewer

The Technology Map Viewer is a detached window that provides a graphical representation of the schematic. To run the Technology Map Viewer for an Intel Quartus Prime Pro Edition project:

1. Click **Processing ➤ Start ➤ Start Analysis & Synthesis** to synthesize and map the design to the target technology.
2. Click **Tools ➤ Netlist Viewers ➤ Technology Map Viewer (Post-Mapping)** to view the post mapping netlist.
3. Click **Processing ➤ Start ➤ Start Fitter**.
   After completing the Fitter stage, the Technology Map Viewer displays how the Fitter modified the netlist as a result of optimizations. After completing the Timing Analysis stage, you can locate timing paths from the Timing Analyzer report in the Technology Map Viewer.
4. Click **Tools ➤ Netlist Viewers ➤ Technology Map Viewer (Post-Fitting)** to view the post fitting netlist.

To find cells by name, click **Edit ➤ Find**, type the cell name, and click **List**.

**Related Information**
Technology Map Viewer Overview
   In *Intel Quartus Prime Pro Edition Handbook Volume 1*

### 3.3.15. Design Optimization

The Intel Quartus Prime software offers advanced netlist optimization options that allow you to optimize a design beyond the standard Intel Quartus Prime Pro Edition compilation flow.

#### Table 35. Design Optimization Tools Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Optimization</td>
<td>-</td>
<td>Hyper-Aware Design Flow(^{(19)})</td>
</tr>
<tr>
<td>Physical Optimization</td>
<td>Physical Synthesis Optimization</td>
<td></td>
</tr>
</tbody>
</table>

\(^{(19)}\) Hyper-Aware Design Flow available only for Intel Stratix 10 devices.
3.3.15.1. Hyper-Aware Design Flow

Use the Hyper-Aware design flow to shorten design cycles and optimize performance for designs targeting Intel Stratix 10 devices. The Hyper-Aware design flow combines automated register retiming (Hyper-Retiming), with implementation of targeted timing closure recommendations (Fast Forward compilation), to maximize use of Hyper-Registers and drive the highest performance for Intel Stratix 10 designs.

Related Information
Running the Hyper-Aware Design Flow
In Intel Quartus Prime Pro Edition Handbook Volume 1

3.3.15.2. Physical Synthesis Optimization

The Vivado software applies timing-driven Physical Optimization in the post-place and reroute designs. The Intel Quartus Prime Pro Edition software performs Physical synthesis optimizations to improve performance regardless of the synthesis tool that you use, and you can apply it during synthesis and during fitting.

The main advantages of performing physical optimization are:

• Optimizations that occur during the synthesis stage change the netlist to improve either area or speed, depending on the optimization technique and effort level that you select.
• 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. You can set this option to Speed or Balanced.
• Optimizations that occur during the Fitter stage of the Intel Quartus Prime Pro Edition compilation flow make placement-specific changes to the netlist that improve speed performance results for a specific Intel FPGA device.

To view and modify synthesis netlist optimization options:

1. Click Assignments ➤ Settings ➤ Compiler Settings.
2. To enable physical synthesis, click Advanced Settings (Fitter), and then enable Advanced Physical Synthesis.
3. View physical synthesis results in the Netlist Optimizations report.

3.3.16. Techniques to Improve Productivity

Table 36. Techniques to Improve Productivity Comparison

<table>
<thead>
<tr>
<th>GUI Feature</th>
<th>Xilinx Vivado Software</th>
<th>Intel Quartus Prime Pro Edition Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Techniques to improve productivity</td>
<td>Incremental Compile</td>
<td>Rapid Recompile(20)</td>
</tr>
<tr>
<td></td>
<td>Hierarchical Design</td>
<td>Block-Based Design Flows</td>
</tr>
<tr>
<td></td>
<td>&quot;</td>
<td>Design Space Explorer II (DSE)</td>
</tr>
</tbody>
</table>

(20) Rapid Recompile available only for Intel Arria 10 devices.
3.3.16.1. Rapid Recompile

In Xilinx Vivado designs, the Incremental Compile design flow speeds up place and route runtime. For reduced compilation time, the Intel Quartus Prime Pro Edition provides Rapid Recompile.

During Rapid Recompile, the Intel Quartus Prime Pro Edition 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.

Note: Intel Stratix 10 devices support Rapid Recompile only for the Signal Tap Logic Analyzer.

To start Rapid Recompile following an initial compilation (or after running the Route stage of the Fitter), click **Processing ➤ Start ➤ Start Rapid Recompile**.

Related Information
Running Rapid Recompile
   In *Intel Quartus Prime Pro Edition Handbook Volume 1*

3.3.16.2. Block-Based Design Flow

In the Vivado software, the Hierarchical Design flow allows you to partition a design into smaller modules that you process independently. These flows are based on the ability to implement a partitioned module out-of-context (OOC) from the rest of the design. A similar feature in Intel Quartus Prime Pro Edition software is the Block-Based Design Flow, which supports preservation and reuse of design blocks in one or more projects.

The Block-Based Design Flow allows you to reuse synthesized, placed, or final design blocks within the same project, or export the block to other projects. Reusable design blocks can include device core or periphery resources.

You can define a logical design partition in a project, and then empty, preserve, or export the contents of that design partition after compilation. The Intel Quartus Prime Pro Edition software supports the following block-based design flows:

- **Incremental Block-Based Compilation**—preserve or empty a core design partition within a project. This flow works only with core resources, and requires no additional files or floorplanning. You can empty the partition, or preserve it at synthesis, placement, or final compilation stages.

- **Design Block Reuse**—export a core or periphery design partition and reuse it in another project. Core partition reuse preserves the placement and routing of timing-critical modules with specific optimized functionality or algorithms, such as modules for encryption, encoding, image processing, or other functions. Periphery partition reuse preserves the placement and routing of the periphery.
### Table 37. Block-Based Design Flows Comparison

<table>
<thead>
<tr>
<th>Xilinx Hierarchical Design Flows</th>
<th>Intel Quartus Prime Pro Edition Block-Based Design Flows</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom-Up Reuse</td>
<td>Design Block Reuse</td>
<td>Build a verified module (such as a piece of IP) which is placed, routed and you can reuse in other designs.</td>
</tr>
<tr>
<td>Top-Down Reuse</td>
<td>Periphery Reuse</td>
<td>Build a verified top-level design with details about the pinout, floorplan and timing requirements.</td>
</tr>
</tbody>
</table>

For more information about design planning and different design approaches, refer to *Block-Based Design Flows* in the *Intel Quartus Prime Pro Edition Handbook Volume 1*.

**Related Information**

*Block-Based Design Flows*

*In Intel Quartus Prime Pro Edition Handbook Volume 1*

### 3.3.16.3. Design Space Explorer II

Intel’s Design Space Explorer II (DSE II) tool 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 setting and constraints, and reports the best settings for the design. DSE II attempts multiple seeds to identify one meeting your requirements. DSE II can compile on multiple computers in parallel to streamline timing closure.

If the 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.

**Related Information**

*Design Space Explorer II*

*In Intel Quartus Prime Pro Edition Handbook Volume 2*

### 3.3.17. Cross-Probing in the Intel Quartus Prime Pro Edition Software

Cross-probing is the ability to select design elements from one tool and locate them in another tool. The integration between features and tools in the Intel Quartus Prime Pro Edition software results in a design environment that provides seamless cross-probing abilities.

For example, you can locate design elements from the RTL Viewer to the Assignment Editor. This eliminates the search time for node names and pin names when applying design constraints in the Assignment Editor. To locate design elements, use the right mouse click button.

### 3.4. Additional Intel Quartus Prime Pro Edition Features

In addition to providing the standard set of tools required in any FPGA design flow, the Intel Quartus Prime Pro Edition software includes additional features and tools to assist you with achieving your desired design requirements.
3.4.1. Scripting with Tcl in the Intel Quartus Prime Pro Edition Software

The Intel Quartus Prime Pro Edition GUI provides an easy way to access all features and commands that the software offers. However, as designs grow in resource utilization and complexity, the need to automate common tasks and streamline the entire FPGA design flow becomes a requirement.

The Intel Quartus Prime Pro Edition software provides support for Tcl to help facilitate project assignments, compilation, and constraints.

The Intel Quartus Prime Pro Edition software contains Tcl application program interface (API) functions that you can use to automate a variety of common tasks, such as making assignments, compiling designs, analyzing timing, and controlling simulation.

You can run your Tcl scripts in the following ways:

Running Scripts from the DOS or UNIX Prompt on page 44
Running Scripts in Batch Mode from a Shell on page 44
Running Tcl Commands Directly from the Command Line on page 44
Running Tcl Commands Interactively from the Shell on page 45
The Intel Quartus Prime Tcl Console Window on page 45

3.4.1.1. Running Scripts from the DOS or UNIX Prompt

The following command runs the Intel Quartus Prime Tcl shell and uses the Tcl file specified by the -t option as the input Tcl script:

```
quartus_sh -t <script_name>.tcl
```

The Intel Quartus Prime Tcl interpreter reads and executes the Tcl commands in the Tcl script file and then exits back to the command-line prompt.

3.4.1.2. Running Scripts in Batch Mode from a Shell

You can run Tcl scripts in a Tcl shell by typing:

```
source <script_name>.tcl
```

3.4.1.3. Running Tcl Commands Directly from the Command Line

You can use the --tcl_eval option to directly evaluate the rest of the command line arguments as one or more Tcl commands. If there are two or more Tcl commands, separate them with semicolons.

1. For example, typing:

```
quartus_sh --tcl_eval puts Hello\; puts World
```
Results in the following output:

Hello
World

The Tcl evaluate option allows external scripting programs (such as make, perl, and sh) to access information from the Intel Quartus Prime Pro Edition software. You can use these programs to obtain device family information for a targeted part.

The --tcl_eval option also provides Tcl help information directly from the command-line prompt.

3.4.1.4. Running Tcl Commands Interactively from the Shell

Using the -s or --shell switch option starts an interactive Tcl shell session, replacing the normal command line prompt with tcl, as shown in the following example:

1. In a console, type:

   quartus_sh -s

   When entering the console, you get the welcome message:

   Info: *******************************************************************
   Info: Running Quartus Prime Shell
   Info: Version 17.1.0 Internal Build 167 08/21/2017 SJ Pro Edition
   Info: Copyright (C) 2017 Intel Corporation. All rights reserved.
   Info: Your use of Intel Corporation's design tools, logic functions
   Info: and other software and tools, and its AMPP partner logic
   Info: functions, and any output files from any of the foregoing
   Info: (including device programming or simulation files), and any
   Info: associated documentation or information are expressly subject
   Info: to the terms and conditions of the Intel Program License
   Info: Subscription Agreement, the Intel Quartus Prime License Agreement,
   Info: the Intel FPGA IP License Agreement, or other applicable license
   Info: agreement, including, without limitation, that your use is for
   Info: the sole purpose of programming logic devices manufactured by
   Info: Intel and sold by Intel or its authorized distributors. Please
   Info: refer to the applicable agreement for further details.
   Info: Processing started: Thu Aug 31 12:50:32 2017
   Info: *******************************************************************
   tcl>

2. Enter any Tcl command.

   The Intel Quartus Prime Tcl interpreter directly evaluates everything that you type in the Tcl shell.

   The Tcl shell includes a history list of previously-entered commands.

3.4.1.5. The Intel Quartus Prime Tcl Console Window

You can execute Tcl commands directly in the Intel Quartus Prime Tcl Console window. To open the Tcl Console window, click View ➤ Tcl Console.
Example 1. Tcl Script

This example uses design files from the fir_filter tutorial in tutorial design in the
<quartus-installation-directory>/qdesigns directory. The script performs these tasks:

- Opens the fir_filter project, if it exists. If the project does not exist, the script creates the project.
- Sets the project to target an Intel Stratix 10 1SG280HU2F50E2VG device.
- Assigns the clk pin to the physical pin AW10.
- Compiles the project.

```tcl
# This Tcl file works with quartus_sh.exe
# This Tcl file will compile the Quartus Prime tutorial fir_filter design

# set the project_name to fir_filter
# set compiler setting to filtref
set project_name fir_filter
set csf_name filtref

# Create a new project and open it
# Project_name is project name
if {![project_exists $project_name]} {
    project_new -family Stratix10 -part 1SG280HU2F50E2VG -cmp $csf_name $project_name;
} else {
    project_open -cmp $csf_name $project_name;
}

# assign pin clk to pin location AW10
set_location_assignment -to clk PIN_AW10

# The project is compiled here
package require ::quartus::flow
execute_flow -compile

project_close
```

Related Information

Tcl Scripting

In *Intel Quartus Prime Pro Edition Handbook Volume 2*
4. Xilinx to Intel FPGA Design Conversion

To successfully convert a Xilinx-targeted design for use in an Intel FPGA device, you must consider the following aspects:

1. Replacing Xilinx primitives with Intel FPGA primitives, IP cores, or constraints.
2. Replacing Vivado IP Catalog modules with IP cores generated with the Intel FPGA IP Catalog.
3. Expressing timing, device, and placement constraints found in the Xilinx design with their counterpart in the Intel Quartus Prime software.
4. If applicable, setting up the simulation environment.

Related Information
Project Creation on page 19

4.1. Replacing Xilinx Primitives

When migrating a design, you must convert common Xilinx primitives to the Intel FPGA equivalents. Primitives are the basic building blocks of a Xilinx design. Primitives perform dedicated functions in the device, and implement standards for I/O pins in Xilinx devices. Primitives names are standard.

The following table lists common Xilinx primitives and describes the equivalent Intel FPGA design element.

Table 38. Common Xilinx Device-Specific Primitives and Intel FPGA Equivalents

<table>
<thead>
<tr>
<th>Xilinx Primitive</th>
<th>Description</th>
<th>Intel FPGA Equivalent</th>
<th>Conversion Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBUF</td>
<td>Single Input Buffer</td>
<td>wire/signal Assignment</td>
<td>HDL</td>
</tr>
<tr>
<td>OBUF</td>
<td>Single Output Buffer</td>
<td>wire/signal Assignment</td>
<td></td>
</tr>
<tr>
<td>BUFG</td>
<td>Global Clock Buffer</td>
<td>wire/signal and Global Signal Assignment</td>
<td>HDL and Assignment Editor</td>
</tr>
<tr>
<td>IBUF&lt;standard&gt;</td>
<td>Input Global Buffer with selectable interface</td>
<td>wire/signal, I/O Standard, and Global Signal Assignment</td>
<td></td>
</tr>
</tbody>
</table>

(21) The attributes of the <selectable I/O standard> are device-specific. For specific I/O standard information, refer to the Xilinx device's data sheet.

(22) For differential I/O buffer, you can assign differential I/O standard to the desired differential I/O signal. The Intel Quartus Prime software automatically creates a new signal, signal_name(n), that is opposite in phase with the desired signal.
<table>
<thead>
<tr>
<th>Xilinx Primitive</th>
<th>Description</th>
<th>Intel FPGA Equivalent</th>
<th>Conversion Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBUF_&lt;selectable I/O standard&gt; (21)</td>
<td>Input buffer with selectable interface</td>
<td>wire/signal and I/O Standard Assignment(22)</td>
<td></td>
</tr>
<tr>
<td>IOBUF_&lt;selectable I/O standard&gt; (21)</td>
<td>Bidirectional buffer with selectable interface</td>
<td>wire/signal and I/O Standard Assignment(22)</td>
<td></td>
</tr>
<tr>
<td>OBUFG_&lt;selectable I/O standard&gt; (21)</td>
<td>Output Global Buffer with selectable interface</td>
<td>wire/signal and I/O Standard Assignment(22)</td>
<td></td>
</tr>
<tr>
<td>OBUF_&lt;selectable I/O standard&gt; (21)</td>
<td>Output buffer with selectable interface</td>
<td>wire/signal and I/O Standard Assignment(22)</td>
<td></td>
</tr>
<tr>
<td>IBUFDS, OBUFDS</td>
<td>Differential I/O Buffer</td>
<td>wire/signal and I/O Standard Assignment(22)</td>
<td></td>
</tr>
<tr>
<td>SRL16</td>
<td>16-bit Shift Register</td>
<td>AUTO_SHIFT_REGISTER_RECOGNITION</td>
<td>Assignment Editor</td>
</tr>
</tbody>
</table>

### 4.1.1. Converting I/O Buffers

The Intel Quartus Prime Pro Edition Compiler inserts input, output, or bidirectional buffers automatically.

To convert a design's buffers to the Intel Quartus Prime Pro Edition software:

1. Remove all buffer primitives from the Xilinx design in the HDL code.
2. Replace the primitives with wire or signal assignments in the HDL code.
3. In the Assignment Editor, perform assignments depending on the type of buffer:

#### Type of Buffer

- **Buffers with selectable I/O standard**: I/O Standard
- **Global buffers**: Global Signal
- **Global buffer with I/O Standard**: I/O Standard

#### 4.1.1.1. Example of Converting I/O Buffer

In this example, the clk, a, and b inputs are global signals, and the a and b inputs use the IBUF DS I/O Standard.

**Example 2. Converting BUFG, IBUFDS, and OBUF in Verilog HDL.**

Original Verilog HDL Code in the Vivado Software

```verilog
module Top (a, b, c, clk);
    input a, b, clk;
    output c;
    reg c_buf;
    wire a_buf, b_buf, clk_buf;
    BUFG inst1 (.O (clk_buf), .I (clk));
    IBUFDS #("FALSE", "SSTL12") inst2 (.O (a_buf), .I (a));
    IBUFDS #("FALSE", "SSTL18_I") inst3 (.O (b_buf), .I (b));
    OBUF inst4 (.O (c), .I (c_buf));
    always @ (posedge clk_buf) c_buf <= a_buf & b_buf;
endmodule
```

---

4. Xilinx to Intel FPGA Design Conversion

AN-307 | 2020.08.24
Converted Verilog HDL Code in the Intel Quartus Prime Pro Edition Software

```verilog
module Top (a, b, c, clk);
    input a, b, clk;
    output c;
    reg c_buf;
    wire a_buf, b_buf, clk_buf;
    assign clk_buf = clk;
    assign a_buf = a;
    assign b_buf = b;
    assign c = c_buf;
    always @ (posedge clk_buf)
        c_buf <= a_buf & b_buf;
endmodule
```

Example 3. Converting `BUFG`, `IBUFG`, and `OBUF` in VHDL.

Original VHDL Code in the Vivado Software

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY buf_top IS PORT(
    a, b : IN STD_ULOGIC; clk : IN STD_ULOGIC; c : OUT STD_ULOGIC);
END buf_top;
ARCHITECTURE Behave OF buf_top IS
    SIGNAL a_buf, b_buf, c_buf, clk_buf : STD_ULOGIC;
    COMPONENT BUFG PORT (O : OUT STD_ULOGIC; I : IN STD_ULOGIC); END COMPONENT;
    COMPONENT IBUFG generic(IBUF_LOW_PWR : boolean := FALSE;
    IOSTANDARD : String := "DEFAULT"); END COMPONENT;
    COMPONENT OBUF PORT (O : OUT STD_ULOGIC; I : IN STD_ULOGIC); END COMPONENT;
BEGIN
    inst1 : BUFG PORT MAP (O => clk_buf, I => clk);
    inst2 : IBUFG generic map( IBUF_LOW_PWR => FALSE,
                             IOSTANDARD => "SSTL12")
                PORT MAP (O => a_buf, I => a);
    inst3 : IBUFG generic map( IBUF_LOW_PWR => FALSE,
                             IOSTANDARD => "SSTL18_I")
                PORT MAP (O => b_buf, I => b);
    inst4 : OBUF PORT MAP (O => c, I => c_buf);
    PROCESS(clk_buf) BEGIN
        IF (clk_buf'event and clk_buf = '1')
            THEN c_buf <= a_buf AND b_buf;
        END IF;
    END PROCESS;
END Behave;
```

Converted VHDL Code in the Intel Quartus Prime Pro Edition Software

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY Top IS
```
PORT(a, b: IN STD_ULOGIC;
    clk: IN STD_ULOGIC;
    c: OUT STD_ULOGIC);
END Top;
ARCHITECTURE Behave OF Top IS
    SIGNAL a_buf, b_buf, c_buf, clk_buf: STD_ULOGIC;
    BEGIN
        PROCESS (a, b, c_buf, clk)
        BEGIN
            clk_buf <= clk;
            a_buf <= a;
            b_buf <= b;
            c <= c_buf;
        END PROCESS;
        PROCESS(clk_buf)
        BEGIN
            IF (clk_buf'event and clk_buf = '1') THEN
                c_buf <= a_buf AND b_buf;
            END IF;
        END PROCESS;
    END Behave;

To set the ports with specific assignments in the Intel Quartus Prime Pro Edition software, use the Assignment Editor.

**Figure 11. Global Signal and I/O Standard Assignments Using the Assignment Editor**

<table>
<thead>
<tr>
<th>No.</th>
<th>From</th>
<th>To</th>
<th>Assignment Name</th>
<th>Value</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>i/O Standard</td>
<td>55TL-12</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>i/O Standard</td>
<td>55TL-18 Class I</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>i/O Standard</td>
<td>1.8 V</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>clk</td>
<td>i/O Standard</td>
<td>1.8 V</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td>Location</td>
<td>PIN_AU25</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>Location</td>
<td>PIN_F14</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>c</td>
<td>Location</td>
<td>PIN_AW20</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>clk</td>
<td>Location</td>
<td>PIN_AW25</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>a</td>
<td>Global Signal</td>
<td>Global Clock</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>b</td>
<td>Global Signal</td>
<td>Global Clock</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>clk</td>
<td>Global Signal</td>
<td>Global Clock</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

In the figure, inputs a, b, and clk are assigned as global signals, with clk as the global clock. Input ports a and b are assigned with specific I/O standards, while other ports are automatically assigned with the device-specific default I/O standard.

**4.1.2. Changing Default I/O Standard for Pins**

The default I/O standard for pins on the target device in the Intel Quartus Prime Pro Edition software is device-specific. To change the default I/O standard:

1. Click **Assignments ➤ Device**.
2. On the **Device** dialog box, click **Device and Pin Options**.
3. In the **Category** list, select **Voltage** and then select the desired I/O standard.
4.2. Converting IP Cores

This section describes how to convert IPs generated using the Xilinx IP Catalog to Intel FPGA IP cores generated with the Intel Quartus Prime Pro Edition IP Catalog.

4.2.1. Converting Memory Blocks

To convert Xilinx memory blocks to Intel FPGA memory blocks, you must consider the embedded memory blocks in the target device, address the differences between memories in Intel FPGA and Xilinx devices, and perform port mapping.

The Xilinx Block Memory Generator defines the following types of memory blocks:

- Single Port RAM
- Simple Dual Port RAM
- True Dual Port RAM
- Single Port ROM
- Dual Port ROM


4.2.1.1. Embedded Memory Blocks

The following table lists the memory blocks that Intel FPGA devices support:

<table>
<thead>
<tr>
<th>Device</th>
<th>Types of Memory Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Arria 10 and Intel Stratix 10</td>
<td>MLAB blocks M20K blocks</td>
</tr>
</tbody>
</table>

For information about memory features and memory specification, refer to the appropriate Embedded Memory Blocks chapter for the target device.

Related Information

- Intel Stratix 10 Embedded Memory User Guide

4.2.1.2. Differences Between Xilinx Memory and Intel FPGA Memory

Consider the differences between Xilinx memory and Intel FPGA memory features and behavior:

- Memory Mode on page 52
- Clocking Mode on page 52
- Write and Read Operation Triggering on page 53
- Read-During-Write Operation at the Same Address on page 53
- Error Correction Code (ECC) on page 54
4.2.1.2.1. Memory Mode

Xilinx memory and Intel FPGA memory support single-port RAM, simple dual-port RAM, true dual-port RAM, single-port ROM, and dual-port ROM.

In addition, Intel FPGA supports simple quad-port RAM, which allows user to perform two read and two write operations to different locations in a single clocking mode. Xilinx memory does not have a built-in simple quad-port RAM.

Related Information

- Memory Port Mapping on page 58
- Inferring Memory Functions from HDL Code
  In Intel Quartus Prime Pro Edition Handbook Volume 1

4.2.1.2.2. Clocking Mode

In Intel FPGAs, the clock mode depend on which embedded memory block you select:

Table 40. Clocking Mode

<table>
<thead>
<tr>
<th>Intel FPGA Clocking Mode</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>All ports share the common clock.</td>
<td></td>
</tr>
<tr>
<td>Input/output</td>
<td>A separate clock is available for input ports and output ports.</td>
<td></td>
</tr>
<tr>
<td>Read/write</td>
<td>A separate clock is available for read ports and write ports.</td>
<td></td>
</tr>
</tbody>
</table>

Xilinx memories do not differentiate these two clocking modes. However, the clocking mode behavior with simple dual-port RAM can be identical to Intel FPGA clocking mode in the following situations:

<table>
<thead>
<tr>
<th>Port A</th>
<th>Port B</th>
<th>Identical to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Output</td>
<td>Input/output clock mode</td>
</tr>
<tr>
<td>Write</td>
<td>Read</td>
<td>Read/write clock mode</td>
</tr>
</tbody>
</table>

For more information about supported clocking modes, refer to the Intel Stratix 10 Embedded Memory User Guide.

Related Information

- Clocking Modes and Clock Enable
- Intel Stratix 10 Embedded Memory Clocking Modes
  In Intel Stratix 10 Embedded Memory User Guide
4.2.1.2.3. Write and Read Operation Triggering

Ensure to resolve potential write contentions external to the RAM, because writing to the same address location at both ports results in unknown data storage at that location. Therefore, knowing when the write operation was triggered is crucial.

The write operation in Intel FPGA memory can occur at either falling clock edges or rising clock edges, depending on the type of embedded memory block. To avoid delta delay, do not trigger control signals together with clock signals.

Related Information
- Write and Read Operations Triggering
- Intel Stratix 10 Embedded Memory Design Considerations
  In Intel Stratix 10 Embedded Memory User Guide

4.2.1.2.4. Read-During-Write Operation at the Same Address

There are two types of read-during-write operations: same-port operations and mixed-port operations.

Figure 12. Read-During-Write Data Flow

The same-port read-during-write mode applies to either:
- a single-port RAM
- the same port of a true-dual port RAM
- the same port a of simple quad-port RAM

Mixed-port read-during-write mode applies to a RAM in:
- simple-dual port
- true-dual port
- simple-quad port mode

that has one port reading and the other port writing to the same address location with the same clock.

Intel FPGA RAM and Xilinx RAM support both read-during-write port modes. However, they have different output options. These options vary depending on the operation mode and type of embedded memory block or device that you select.
Intel FPGA RAMs support configurations with output options of NEW_DATA (flow-through), OLD_DATA, DONT_CARE, or NEW_A_OLD_B. Xilinx RAMs support configurations with output options READ_FIRST, WRITE_FIRST, or NO_CHANGE.

Table 41. Output Options in Xilinx RAM and Intel FPGA RAM for Read-During-Write Operation

<table>
<thead>
<tr>
<th>Description</th>
<th>Output after Read-During-Write operation</th>
<th>Types of RAM Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output reflects the new data at that address.</td>
<td>New data</td>
<td>WRITE_FIRST</td>
</tr>
<tr>
<td>Outputs reflect the old data at that address before the new data is written into memory.</td>
<td>Old data</td>
<td>READ_FIRST</td>
</tr>
<tr>
<td>Outputs reflect the previous read data and remains unaffected by the write operation.</td>
<td>Unaffected</td>
<td>NO_CHANGE</td>
</tr>
<tr>
<td>Read-during-write writes new data into memory, and the output displays unknown values.</td>
<td>Unknown</td>
<td>Not supported(23)</td>
</tr>
<tr>
<td>For simple quad port, the read-during-write operation behaves differently for each port:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• In port A, the operation writes new data into memory and displays new data at output</td>
<td>Port A: New data</td>
<td>Not supported</td>
</tr>
<tr>
<td>• In port B, the operation writes new data but displays old data.</td>
<td>Port B: Old data</td>
<td>NEW_A_OLD_B</td>
</tr>
</tbody>
</table>

In Intel FPGA RAMs, the output choices depend on the operation mode and the type of embedded memory block. For information about output choices for same-port and mixed-port read-during-write modes, refer to the Embedded Memory Blocks chapter in the corresponding device handbook.

4.2.1.2.5. Error Correction Code (ECC)

Xilinx and Intel FPGA RAMs use Error Correction Code (ECC) to detect errors in the memory array, and present the corrected single-bit error data on the output.

(23) To implement NO_CHANGE behavior, you must add additional logic. Use the write_enable signal and compare the write and read addresses to track the operation.

(24) You can choose DONT_CARE for a read-during-write operation if the output is not crucial to your design.
Table 42. Comparison of ECC Support and Status Output Signals for Intel FPGA and Xilinx RAMs

<table>
<thead>
<tr>
<th></th>
<th>Intel FPGA</th>
<th>Xilinx</th>
</tr>
</thead>
</table>
| ECC support            | • Built-in support for M20K type of Embedded Memory Block selected in simple dual-port mode.  
                          | • Intel FPGA also provides a dedicated soft IP ECC core that is flexibly implemented in your design, and is not restricted by the type of memory block used. For more information about the ECC IP core, refer to the Intel FPGA Integer Arithmetic IP Cores User Guide. | For UltraScale+ and 7-series devices in simple dual-port RAM. |
| Status Signal          | Indicates the status of the M20K block using a three-bit status flag eccstatus[1..0] | Indicates the status of the data read using two status outputs:  
                          | • SBITERR  
                          | • DBITERR |

Note: Intel FPGA does not support using ECC with the byte-enable feature. In addition, Intel FPGA does not support read-during-write “old data” mode if ECC is selected.

Related Information
ALTECC (Error Correction Code: Encoder/Decoder) IP Core
Intel FPGA Integer Arithmetic IP Cores User Guide

ECC Parity Flip (Intel Stratix 10 Devices only)

The ECC parity flip feature dynamically flips the parity value generated in the encoder of M20K blocks to observe the ECC behavior through simulation.

When the ECC Encoder Bypass (eccencbypass) port is high, the built-in ECC encoder values are XOR-ed with the 8 parity bits through the parity ports to generate a new set of encoder value. When the ECC Encoder Bypass port is low, the encoder generates the parity bits according to the data input during a write process.

The following table shows an example to construct an 8-bit data width for the parity port.

Table 43. Example of Setting the 8-Bit Parity Ports

<table>
<thead>
<tr>
<th>Parity Bit Sequence</th>
<th>ECC Feature</th>
<th>Is the ECC Decoder able to Recognize and Correct the Data Bit?</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>Single-error correction</td>
<td>Yes</td>
</tr>
<tr>
<td>00000001</td>
<td>Double-adjacent-error correction</td>
<td>Yes</td>
</tr>
<tr>
<td>00000111</td>
<td>Triple-adjacent-error correction</td>
<td>Yes</td>
</tr>
<tr>
<td>00000101</td>
<td>Triple-adjacent-error correction</td>
<td>Yes</td>
</tr>
<tr>
<td>00010011</td>
<td>Non-adjacent double/triple correction/detection</td>
<td>No guarantee</td>
</tr>
</tbody>
</table>

For more information about ECC, refer to the chapter about Embedded Memory Blocks in your target device handbook.
4.2.1.2.6. Byte Enable

To ensure that the operation writes only specific bytes of data, embedded memory blocks support the byte enable property, that masks the input data. The unwritten bytes or bits retain the previous value.

**Note:** Xilinx RAMs support byte enable in Virtex-4 and newer devices.

The following table compares byte enable implementation in Xilinx and Intel FPGA RAMs.

### Table 44. Byte Enables Differences in Xilinx RAM and Intel FPGA RAM

<table>
<thead>
<tr>
<th>Differences</th>
<th>Xilinx RAM</th>
<th>Intel FPGA RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlling signals</td>
<td>The WEA[n:0] signal controls the byte enable. Each bit in WEA[n:0] acts as a write enable for the corresponding input data byte.</td>
<td>Uses two signals, write_enable and byte_enable. To control which byte to write, assert the write_enable signal and the specific bit of the byte_enable signal. For example, in a RAM block in x16 mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>byte_enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input data width support</td>
<td>Support multiples of 8 or 9 bits.</td>
<td>Support multiples of 5, 8, 9, 10 bits. For configurations smaller than two bytes wide, the write_enable or clock_enable signals control the write operation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output value of masked byte when performing read-during-write to the same location.</td>
<td>Output depends on read-during-write configuration: WRITE_FIRST, READ_FIRST, NO_CHANGE</td>
<td>Output depends on the type of memory block:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory Block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SDP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True dual Port</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Simple Quad port</td>
</tr>
</tbody>
</table>

**Related Information**

- Read-During-Write Operation at the Same Address on page 53
- Byte Enable in Intel Stratix 10 Embedded Memory Blocks In Intel Stratix 10 Embedded Memory User Guide

---

(25) Only MLAB memory blocks support byte_enable for input data width that is multiple of 5.
4.2.1.2.7. Address Clock Enable

Intel FPGA memory supports the address clock enable feature. The address clock enable holds the previous address value for as long as `addressstall` is enabled. Xilinx RAM blocks support an equivalent feature, called Address Enable.

For more information about the address clock enable feature, refer to the Embedded Memory Blocks chapter in your target device handbook.

**Related Information**
- Memory Blocks Address Clock Enable Support
- Address Clock Enable Support
  In Intel Stratix 10 Embedded Memory User Guide

4.2.1.2.8. Parity Bit Support

Embedded memory blocks in Intel FPGAs have built-in parity-bit support for each byte. While Xilinx memories support separate input and output buses for parity bits, the embedded memory blocks in Intel Stratix 10 devices allow you to inject parity bits through the ECC encoder bypass feature.

The amount of memory in each RAM block includes the parity bits. No parity function is actually performed on the parity bits. You can use the parity bits for purposes other than ensuring data integrity; for example, to store user-specified control bits.

For more information about using the parity bit to detect memory errors, refer to the *Using Parity to Detect Errors* White Paper.

**Related Information**
- Parity Bit
  In Intel Stratix 10 Embedded Memory User Guide

4.2.1.2.9. Memory Initialization

In Intel FPGA devices, all embedded memory blocks support memory initialization, and initialize memory contents through memory initialization files (.mif) or Hexadecimal (Intel-Format) files (.hex). You can create these files with the Intel Quartus Prime Pro Edition software. You specify the initialization file name while configuring your memory IP core through the IP Catalog/Parameter Editor.

Xilinx devices use a memory coefficient (COE) file for initialization. Alternatively, you can use the default data option.

**Related Information**
- Hexadecimal (Intel-Format) File (.hex) Definition
  In Intel Quartus Prime Help
- Memory Initialization File (.mif) Definition
  In Intel Quartus Prime Help
- Converting Xilinx RAM initialization .coe/.mif Format to Intel PSG .mif/.hex Format
### 4.2.1.2.10. Output Synchronous Set/Reset

Xilinx memory supports optional synchronous set/reset pins that control the reset operation of the last register in the output stage. This ability initializes the memory’s output to a user-defined value.

Intel FPGA memory also supports asynchronous clear and synchronous clear on output latches and output registers. If the RAM does not use output registers, clear the RAM outputs using the output latch asynchronous clear (aclr). The aclr signal is generated at any time. The internal logic extends the clear pulse until the next rising edge of the output clock. When the aclr signal asserts, the outputs are cleared and stay clear until the next read cycle.

### 4.2.1.3. Determining Memory Block and Mapping Ports

1. If you are not sure which memory block to select, or are not particular about the memory block type, select **AUTO** in the IP Catalog/Parameter Editor. This option allows the Intel Quartus Prime software to determine the memory block type at compile time.
   - To find the type of memory block that the Intel Quartus Prime software assigned to your design, check the Intel Quartus Prime Fitter RAM Summary Report.

2. Otherwise, build the memory blocks in the IP Catalog/Parameter Editor using the proper plug-in. The available plug-ins are:

<table>
<thead>
<tr>
<th>Table 45. Memory Modes/Functions and Related Plug-In</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory Modes/Function</strong></td>
</tr>
<tr>
<td>Single-port RAM</td>
</tr>
<tr>
<td>Simple dual-port RAM</td>
</tr>
<tr>
<td>True dual-port RAM</td>
</tr>
<tr>
<td>Simple quad-port RAM (Intel Stratix 10 only)</td>
</tr>
<tr>
<td>Single-port ROM</td>
</tr>
<tr>
<td>Dual-port ROM</td>
</tr>
</tbody>
</table>

For information about memory options, and how to build the memory function through the IP Catalog/Parameter Editor, refer to the embedded memory user guide.

3. Identify the port-mapping from Xilinx memory ports to Intel FPGA memory ports.

**Related Information**

- Fitter Feature Specific Reports
  - In *Intel Quartus Prime Help*

### 4.2.1.4. Memory Port Mapping

The following table lists the memory ports that the Vivado’s IP Catalog generates, and their corresponding mapping to Intel FPGA memory ports for different memory modes.
### Table 46. Block Memory Generator’s Memory Port Mapping to Intel FPGA Memory Ports

<table>
<thead>
<tr>
<th>Port Description</th>
<th>Xilinx Ports</th>
<th>Port-Mapping to Intel FPGA Ports in Different Memory Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Single-Port RAM</td>
</tr>
<tr>
<td>Port A: address</td>
<td>addra</td>
<td>address</td>
</tr>
<tr>
<td>Port A: data input</td>
<td>dina</td>
<td>data</td>
</tr>
<tr>
<td>Port A: parity data input</td>
<td>dinpa</td>
<td>—</td>
</tr>
<tr>
<td>Port A: clock enable for the input register</td>
<td>ena</td>
<td>inclocken/ clken</td>
</tr>
<tr>
<td>Port A: clock enable for the last output register</td>
<td>regcea, ena</td>
<td>outclocken/ clken</td>
</tr>
<tr>
<td>Port A: write enable</td>
<td>NA</td>
<td>wren</td>
</tr>
<tr>
<td>Port A: byte enable(26)</td>
<td>wea</td>
<td>byteena</td>
</tr>
<tr>
<td>Port A: asynchronous clear</td>
<td>NA</td>
<td>outaclr/ aclr</td>
</tr>
<tr>
<td>Port A: synchronous set/reset</td>
<td>rsta/ rstrega</td>
<td>sclr</td>
</tr>
<tr>
<td>Port A: read enable</td>
<td>ena (in SDP(27) mode)</td>
<td>rden</td>
</tr>
<tr>
<td>Port A: in clock</td>
<td>clka</td>
<td>inclock/ clock</td>
</tr>
<tr>
<td>Port A: out clock</td>
<td>NA</td>
<td>outclock/ clock</td>
</tr>
<tr>
<td>Port A: data output</td>
<td>douta</td>
<td>q</td>
</tr>
<tr>
<td>Port A: parity data output</td>
<td>doutpa</td>
<td>—</td>
</tr>
</tbody>
</table>

(26) For configurations less that two bytes wide, Xilinx write enable signals (wea and web) are equivalent to Intel FPGA write enable signals (wren, wren_a, or wren_b) signals, depending on the memory mode used. For configurations of more than two bytes, Xilinx’s write enable buses (wea[] and web[]) are equivalent to Intel FPGA byte enable buses(byteena[], byteena_a[], or byteena_b[]), depending on the memory mode used. Also, the Intel FPGA write enable signal needs to be asserted for the write operation.

(27) Xilinx simple dual-port RAM generated through Block Memory Generator
<table>
<thead>
<tr>
<th>Port Description</th>
<th>Xilinx Ports</th>
<th>Port-Mapping to Intel FPGA Ports in Different Memory Modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port A: address enable</td>
<td>addrena(^{(28)})</td>
<td>Single-Port RAM</td>
</tr>
<tr>
<td></td>
<td>addressstall(_a)</td>
<td>wr_addressstall</td>
</tr>
<tr>
<td>Port B: address</td>
<td>addrb</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: data input</td>
<td>dinb</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: parity data input</td>
<td>dinpb</td>
<td>—</td>
</tr>
<tr>
<td>Port B: clock enable for the input register</td>
<td>enb</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: clock enable for the last output register</td>
<td>regceb, enb</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: write enable</td>
<td>enb (in SDP(^{(27)}) mode)</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: byte enable</td>
<td>web</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: asynchronous clear</td>
<td>—</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: synchronous set/reset</td>
<td>rstb/rstregb</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: read enable</td>
<td>—</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: clock</td>
<td>clkb</td>
<td>outclock/clock</td>
</tr>
<tr>
<td>Port B: data output</td>
<td>doutb</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: address enable</td>
<td>addrenb</td>
<td>NA</td>
</tr>
<tr>
<td>Port B: parity data output</td>
<td>doutpb</td>
<td>—</td>
</tr>
<tr>
<td>Single bit error</td>
<td>sbiterr</td>
<td>NA</td>
</tr>
<tr>
<td>Double bit error</td>
<td>dbiterr</td>
<td>NA</td>
</tr>
</tbody>
</table>

\(^{(28)}\) Port mappings denoted with NA are not applicable for that memory mode; port-mappings denoted with — are not supported in Intel FPGA memory.
### Port-Mapping to Intel FPGA Ports in Different Memory Modes

<table>
<thead>
<tr>
<th>Port Description</th>
<th>Xilinx Ports</th>
<th>Single-Port RAM</th>
<th>Simple Dual-Port RAM</th>
<th>True Dual-Port RAM</th>
<th>Single-Port ROM</th>
<th>Dual-Port ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC encoder bypass port</td>
<td>-</td>
<td>NA</td>
<td>eccencbypass</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>ECC parity flip port</td>
<td>-</td>
<td>NA</td>
<td>eccncparity[7:0]</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Inject single bit error</td>
<td>injectsberr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>Inject double bit error</td>
<td>injectdbiterr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NA</td>
</tr>
</tbody>
</table>

You can also infer RAM in HDL. For more information, refer to the *Recommended HDL Coding Styles* in the Intel Quartus Prime Pro Edition Handbook Volume 1.

#### Related Information
- Memory Mode on page 52
- Inferring Memory Functions from HDL Code
  In *Intel Quartus Prime Pro Edition Handbook Volume 1*

### 4.2.1.5. Example: Converting Simple Dual-Port RAM

This example includes Verilog HDL and VHDL code for the top level that instantiates the Xilinx simple dual-port RAM.

In this example, the top-level entity `test` instantiates `sdp_ram`, a Xilinx simple dual-port RAM generated through Block Memory Generator, with the following properties:

#### Table 47. Properties of Simple Dual-Port RAM

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input data width</td>
<td>16 bits</td>
</tr>
<tr>
<td>Memory depth</td>
<td>8 words</td>
</tr>
<tr>
<td>Clocking Mode</td>
<td>Different input and output clocks</td>
</tr>
<tr>
<td>ECC feature</td>
<td>Selected</td>
</tr>
<tr>
<td>Out data registered status</td>
<td>Output registered (one stage pipeline)</td>
</tr>
<tr>
<td>Read-during-write</td>
<td>WRITE_FIRST (New Data)</td>
</tr>
</tbody>
</table>

The original Verilog HDL Code in the Vivado Software is:

```verilog
module test(
    .clka,
    .ena,
    .wea,
    .addra,
    .dina,
    .clkb,
    .enb,
    .addrb,
    .doutb,
    .sbiterr,
    .dbiterr,
    .rdaddrecc);
```

```verilog
simple dual port ip 11(
    .clka(clka),
```
The original VHDL Code in the Vivado Software is:

```vhdl
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
LIBRARY work;
ENTITY test IS
  port (                  
    clka: IN STD_LOGIC;  
    ena: IN STD_LOGIC; 
    wea: IN STD_LOGIC_VECTOR(0 DOWNTO 0);  
    addra: IN STD_LOGIC_VECTOR(2 DOWNTO 0); 
    dina: IN STD_LOGIC_VECTOR(15 DOWNTO 0);  
    clkb: IN STD_LOGIC;  
    enb: IN STD_LOGIC;  
    addrb: IN STD_LOGIC_VECTOR(2 DOWNTO 0);  
    doutb: OUT STD_LOGIC_VECTOR(15 DOWNTO 0); 
    dbiterr: OUT STD_LOGIC;  
    sbiterr: OUT STD_LOGIC;  
    rdaddrecc: OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END test;
ARCHITECTURE arch OF test IS
  component simple_dual_port_ip
    PORT (                  
      clka: IN STD_LOGIC;  
      ena: IN STD_LOGIC; 
      wea: IN STD_LOGIC_VECTOR(0 DOWNTO 0);  
      addra: IN STD_LOGIC_VECTOR(2 DOWNTO 0); 
      dina: IN STD_LOGIC_VECTOR(15 DOWNTO 0);  
      clkb: IN STD_LOGIC;  
      enb: IN STD_LOGIC;  
      addrb: IN STD_LOGIC_VECTOR(2 DOWNTO 0);  
      doutb: OUT STD_LOGIC_VECTOR(15 DOWNTO 0); 
      dbiterr: OUT STD_LOGIC;  
      sbiterr: OUT STD_LOGIC;  
      rdaddrecc: OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
  end component;
BEGIN
  il: simple_dual_port_ip
    PORT MAP (                  
      clka => clka,             
      ena => ena,               
      wea => wea,               
      addra => addra,           
      dina => dina,             
      clkb => clkb,             
      enb => enb,               
      addrb => addrb,           
      doutb => doutb,           
      dbiterr => dbiterr,       
      sbiterr => sbiterr,       
      rdaddrecc => rdaddrecc);    
END;
```
To convert a Xilinx Simple Dual Port RAM to Intel FPGA:
1. Create an Intel FPGA simple dual-port RAM through the Intel Quartus Prime software IP Catalog/Parameter Editor.
2. Configure the RAM with the following options:

**Table 48. Parameters of Simple Dual-Port RAM**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>How will you be using the dual port RAM?</td>
<td>Specifies how you use the dual port RAM.</td>
</tr>
<tr>
<td>With one read port and one write port</td>
<td></td>
</tr>
<tr>
<td>Read/Write Ports</td>
<td>Specifies the width of the input and output ports.</td>
</tr>
<tr>
<td>How wide should the 'q_a' output bus be?</td>
<td>16 bits</td>
</tr>
<tr>
<td>How wide should the 'q_b' output bus be?</td>
<td>16 bits</td>
</tr>
<tr>
<td>What should the memory type be?</td>
<td>Specifies the memory block type. The available memory blocks depend on the target device.</td>
</tr>
<tr>
<td>RAM Block Type</td>
<td>M20K</td>
</tr>
<tr>
<td>What clocking method do you want to use?</td>
<td>Specifies the clocking method to use.</td>
</tr>
<tr>
<td>Dual clock: use separate 'read' and 'write' clock</td>
<td>A write clock controls the data-input, write-address, and write-enable registers while the read clock controls the data-output, read-address, and read-enable registers.</td>
</tr>
<tr>
<td>ECC Checking</td>
<td></td>
</tr>
<tr>
<td>Enable Error Correction Check (ECC)</td>
<td>On</td>
</tr>
<tr>
<td>Enable ECC Pipeline Registers</td>
<td>Specifies whether to enable the ECC pipeline registers before the output decoder to achieve that same performance as non-ECC mode at the expense of one cycle of latency</td>
</tr>
<tr>
<td>Clock Enables</td>
<td>Specifies whether to create clock enables for read and write registers.</td>
</tr>
<tr>
<td>Use different clock enables for registers</td>
<td>On</td>
</tr>
<tr>
<td>Use clock enable for write input registers</td>
<td>On</td>
</tr>
<tr>
<td>Use clock enable for output registers</td>
<td>On</td>
</tr>
</tbody>
</table>

**Note:** Intel FPGA RAMs do not support read-during-write ‘old data’ mode when the ECC feature is enabled. Therefore, when you convert a Xilinx RAM to an Intel FPGA RAM, you can expect to see a ‘don’t care’ value when read-during-write to the same address occurs. The workaround to get the read-during-write ‘old data’ mode behavior is to add additional logic to retain its previous read data when read-during-write to the same address occurs.

3. Instantiate the new Intel FPGA RAM to replace the Xilinx RAM.

The converted Verilog HDL code in the Intel Quartus Prime Software after instatiating the new RAM:

```verilog
defmodule test(
    input clka,
    input ena,
    input [0:0]wea,
    input [2:0]addra,
    input [15:0]dina,
```
The converted VHDL code in the Intel Quartus Prime Software:

```vhdl
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.all;
LIBRARY work;
ENTITY test IS
  port (
    clka: IN STD_LOGIC;
    ena: IN STD_LOGIC;
    wea: IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    dina: IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    clkb: IN STD_LOGIC;
    end: IN std_Iogic;
    addrb: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    doutb: OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
    dbiterr: OUT STD_LOGIC;
    sbiterr: OUT STD_LOGIC;
    rdaddrecc: OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END test,
ARCHITECTURE arch OF test IS
  component simple_dual_port_ip
    PORT(
      wrclock: IN STD_LOGIC;
      wrclocken: IN STD_LOGIC;
      wren: IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      wraddress: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      data: IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      rdclock: IN STD_LOGIC;
      rdaddress: IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      q: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
      eccstatus: OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
  end component;
  signal eccstatus_o: STD_LOGIC_VECTOR(1 DOWNTO 0);
  BEGIN
    dbiterr <= eccstatus_o(1);
    sbiterr <= eccstatus_o(0);
    i1: simple_dual_port_ip
      PORT MAP(
        wrclock => clka,
        wrclocken => ena,
        wren => wea,
        wraddress => addra,
        data => dina,
        rdclock => clkb,
        rdaddress => addrb,
        q => doutb,
        eccstatus => eccstatus_o);
  END;
```

The converted VHDL code in the Intel Quartus Prime Software:
4.2.2. Converting Mixed-Mode Clock Manager (MMCM) to Phase-Locked Loop (PLL)

Similar to Mixed-Mode Clock Managers (MMCM) in Xilinx devices, some Intel FPGA device families support PLLs. This ability increases device and board-level performance by allowing you to minimize clock skew and clock delay and provide support for clock synthesis.

You can convert MMCMs to PLLs in Intel FPGA devices with the IP Catalog/Parameter Editor by using the Intel FPGA IOPLL IP core, which allows you to create custom PLLs targeting to Intel FPGA devices.

Note: Intel now refers to the ALTPLL IP Core as Intel FPGA IOPLL IP core.

Xilinx MMCMs require specific input buffers to feed into the source clock port; for example, IBUF, IBUFG, or BUFGMUX. In contrast, PLLs in Intel FPGA devices do not require input buffers when using the IP Catalog/Parameter Editor.

4.2.2.1. Feature Comparison

The following table compares MMCM features in UltraScale+ with PLL features in Intel Stratix 10 devices.

<table>
<thead>
<tr>
<th>Features</th>
<th>Xilinx MMCM (UltraScale+)</th>
<th>Intel FPGA IOPLL (Intel Stratix 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency Synthesis</td>
<td>Clock Multiplication and Division</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Phase Shifting</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Clock Duty Cycle</td>
<td>Yes</td>
</tr>
<tr>
<td>MMCN Deskew Adjust</td>
<td>Internal Feedback</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Spread Spectrum</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>System Synchronous</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Normal Mode</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Source Synchronous</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Zero Delay Buffer</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No Compensation</td>
<td>Yes (CMT to CMT connection)</td>
</tr>
<tr>
<td></td>
<td>External Feedback</td>
<td>Yes</td>
</tr>
</tbody>
</table>

(29) LVDS compensation mode can also be considered as a special case of Source Synchronous Compensation Mode for pins of internal serializer/deserializer (SERDES) capture register.
### 4.2.2.2. Port Mapping Reference

The following table shows the mapping between MMCM UltraScale ports, created with the Xilinx IP Catalog, and PLL ports in Intel Stratix 10 device, created with the IP Catalog.

**Table 50. Port-Mapping MMCM UltraScale versus PLL Intel Stratix 10**

<table>
<thead>
<tr>
<th>Xilinx MMCM Core Port</th>
<th>Intel FPGA IOPLL IP Core Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk_in1</td>
<td>refclk</td>
<td>First clock input</td>
</tr>
<tr>
<td>clk_in2</td>
<td>refclk1</td>
<td>Second clock input</td>
</tr>
<tr>
<td>clkfb_in</td>
<td>fbclk</td>
<td>External clock feedback</td>
</tr>
<tr>
<td>clkfbout</td>
<td>fboutclk</td>
<td>Feeds into the feedback port</td>
</tr>
<tr>
<td>−</td>
<td>activeclk</td>
<td>Output signal that indicates which reference clock source the I/O PLL uses</td>
</tr>
<tr>
<td>clk_in_sel</td>
<td>extswitch</td>
<td>Switch between input clock ports</td>
</tr>
<tr>
<td>reset</td>
<td>rst</td>
<td>Asynchronous reset port</td>
</tr>
<tr>
<td>clk_out1, clk_out2, clk_outX</td>
<td>outclk_[]</td>
<td>Clock frequency output ports. Xilinx MMCM has fixed settings for most outputs, and you can configure the Intel FPGA IOPLL IP core to suit them.</td>
</tr>
<tr>
<td>clkinstopped</td>
<td>clkbad[1..0]</td>
<td>Indicates whether the clock input signal stopped switching</td>
</tr>
<tr>
<td>clkfb_stopped</td>
<td>−</td>
<td>Specifies whether the feedback clock stopped</td>
</tr>
<tr>
<td>locked</td>
<td>locked</td>
<td>Specifies whether the PLL is locked</td>
</tr>
<tr>
<td>−</td>
<td>adjpllin</td>
<td>Input signal that feeds from upstream I/O PLL</td>
</tr>
<tr>
<td>−</td>
<td>cascade_out</td>
<td>Output signal that feeds into downstream I/O PLL</td>
</tr>
<tr>
<td>−</td>
<td>zdbfbclk</td>
<td>Bidirectional port that connects to the mimic circuitry. You connect this port to a bidirectional pin that is placed on the positive feedback dedicated output pin of the I/O PLL. The zdbfbclk port is available only if the I/O PLL is in zero-delay buffer mode.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Xilinx MMCM Core Port</th>
<th>Dynamic Phase Shift Ports in Intel FPGA IOPLL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>psclk</td>
<td>scanclk</td>
<td>Specifies clock that drives the dynamic phase shift operation</td>
</tr>
<tr>
<td>psen</td>
<td>phase_en</td>
<td>Start dynamic phase-shift operation</td>
</tr>
<tr>
<td>psincdec</td>
<td>updn</td>
<td>Specifies direction of phase shift operation</td>
</tr>
</tbody>
</table>

continued...
### Description

<table>
<thead>
<tr>
<th>Xilinx MMCM Core Port</th>
<th>Dynamic Phase Shift Ports in Intel FPGA IOPLL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>cntsel</td>
<td>Specifies counter for dynamic phase shift operation</td>
</tr>
<tr>
<td>—</td>
<td>num_phase_shift</td>
<td>Specifies number of phase shifts per dynamic phase shift operation</td>
</tr>
<tr>
<td>psdone</td>
<td>phase_done</td>
<td>Specifies completion of dynamic phase shift operation</td>
</tr>
<tr>
<td>power_down</td>
<td></td>
<td>Enables power_down input port for user selection</td>
</tr>
</tbody>
</table>

For more information about using dynamic PLL reconfiguration, refer to the Phase-Locked Loops (Intel FPGA IOPLL) IP Core User Guide.

### Related Information
- Altera I/O Phase-Locked Loop (Altera IOPLL) IP Core User Guide
- Intel Stratix 10 Clocking and PLL User Guide

### 4.2.2.3. Example: Converting Xilinx MMCM into an Intel PLL

This example uses a `mymmcm` module generated with the Xilinx IP Catalog. The top module instantiates the `mymmcm` module with `i1`. The parameters are:

<table>
<thead>
<tr>
<th>Table 51. Example Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Input Clock Frequency</td>
</tr>
<tr>
<td>Clock frequency output port</td>
</tr>
<tr>
<td>Clock frequency output port</td>
</tr>
</tbody>
</table>

#### Original Verilog Code in the Vivado Software:

```verilog
module top;
    // Clock out ports
    output clk_out1,
    output clk_out2,
    // Status and control signals
    input reset,
    output locked,
    // Clock in ports
    input clk_in1
);
mymmcm i1 (.
    .reset(reset),
    .clk_in1(clk_in1),
    .locked(locked),
    .clk_out1(clk_out1),
    .clk_out2(clk_out2)
);
endmodule
```

To recreate the same behavior using Intel FPGA software:

1. In the IP Catalog/Parameter Editor, point to **Library ➤ Basic Functions ➤ Clocks, PLLs and Resets ➤ PLL**, and double-click **Intel FPGA IOPLL**.
2. Generate an IP variant named mypll.
3. In the Parameter Editor, set the following parameters:

Table 52. Parameters of mypll

<table>
<thead>
<tr>
<th>General</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference Clock Frequency</td>
<td>100 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Clocks</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Clocks</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Specifies the number of clocks that your design requires</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>outclk0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Name</td>
<td>clk_out1</td>
</tr>
<tr>
<td>Desired Frequency</td>
<td>50 MHz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>outclk1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Name</td>
<td>clk_out2</td>
</tr>
<tr>
<td>Desired Frequency</td>
<td>400 MHz</td>
</tr>
</tbody>
</table>

4. Click Finish.
5. Create a top module, and instantiate the mypll module with i1.

The converted Verilog HDL code in the Intel Quartus Prime Software is:

```verilog
module top(output clk_out1,
            output clk_out2,
            input reset,
            output locked,
            input clk_in1);
    mypll i1(.rst(reset),
              .refclk(clk_in1),
              .locked (locked),
              .outclk_0 (clk_out1),
              .outclk_1(clk_out2));
end module
```

4.2.3. Converting Multipliers

The following section discusses converting instances of the Xilinx Multiplier Core to Intel FPGA Multiplier IP cores. Intel provides two IP cores for implementing multiply, multiply-accumulate, and multiply-add functions using DSP blocks or logic resources:
• The LPM_MULT IP core, which performs multiply functions only.
• The Intel FPGA Multiply Adder IP core, which performs multiply, multiply-add, or multiply-accumulate functions.

To perform the conversion:
1. In the original code, identify whether the dataa and datab ports have the same sign.
2. If the ports have the same sign, replace the Xilinx Multiplier Core with the LPM_MULT IP core,
3. Otherwise, replace with the Intel FPGA Multiply Adder IP core.
4. In the IP Catalog, click the selected IP core.
5. Assign the parameters and generate HDL Intel FPGA IP core.

For more information, refer to Inferring Multipliers in the Recommended HDL Coding Styles chapter of the Intel Quartus Prime Pro Edition Handbook Volume 1.

Related Information
Inferring Multipliers and DSP Functions
In Intel Quartus Prime Pro Edition Handbook Volume 1

4.2.3.1. Feature Comparison

The Xilinx Multiplier Core and the Intel FPGA LPM_MULT IP core have similar features; however, you must consider one difference:
• In the LPM_MULT IP core, the dataa and datab ports must have the same sign. If your design does not meet this requirement, you can use the Intel FPGA Multiply Adder IP core to replace the Xilinx Multiplier Core.

The following table compares the Xilinx Multiplier Core and the Intel FPGA LPM_MULT IP core.

Table 53. Xilinx Multiplier Core versus Intel FPGA LPM_MULT IP Core

<table>
<thead>
<tr>
<th>Feature</th>
<th>Xilinx Multiplier Core Generator Module</th>
<th>Intel FPGA LPM_MULT IP Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant Coefficient</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Signed and Unsigned Data</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Configurable Pipeline Latency</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Area versus Speed Trade-off</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Asynchronous Clear</td>
<td>—</td>
<td>Yes</td>
</tr>
<tr>
<td>Synchronous Clear</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Port A and Port B support different sign</td>
<td>Yes</td>
<td>— Consider using the Intel FPGA LPM_MULT IP core to replace the Xilinx Multiplier Core.</td>
</tr>
</tbody>
</table>

Related Information
LPM_MULT (Multiplier) IP Core
In Intel FPGA Integer Arithmetic IP Cores User Guide
4.2.3.2. Port Mapping

The following table shows the port mapping between the Xilinx Multiplier Core and the Intel FPGA LPM_MULT IP core.

<table>
<thead>
<tr>
<th>Xilinx Multiplier Core Port</th>
<th>Intel FPGA LPM_MULT IP Core Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A []</td>
<td>dataa []</td>
<td>Data Input Port A</td>
</tr>
<tr>
<td>B []</td>
<td>datab []</td>
<td>Data Input Port B</td>
</tr>
<tr>
<td>CLK []</td>
<td>clock</td>
<td>Clock Port</td>
</tr>
<tr>
<td>CE</td>
<td>clken</td>
<td>Clock Enable Port</td>
</tr>
<tr>
<td>SCLR</td>
<td>sclr</td>
<td>Synchronous Clear Port</td>
</tr>
<tr>
<td>N/A</td>
<td>aclr</td>
<td>Asynchronous Clear Port</td>
</tr>
<tr>
<td>P []</td>
<td>result []</td>
<td>Multiplication Result Port</td>
</tr>
</tbody>
</table>

4.2.3.3. Example: Converting to the LPM_MULT IP Core

You can convert the Xilinx Multiplier Core that targets a Xilinx device into multipliers for an Intel FPGA device by using the IP Catalog.

In this example, the test module instantiates the mymult module, created using the Xilinx Core Generator. The parameters are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplier Type</td>
<td>Parallel multiplier where neither of the on the input buses is a constan value</td>
</tr>
<tr>
<td>Input data width</td>
<td>18 bits</td>
</tr>
<tr>
<td>Input data type</td>
<td>Signed</td>
</tr>
<tr>
<td>Output result width</td>
<td>Restricted to 36 bits</td>
</tr>
<tr>
<td>Number o pipeline stages</td>
<td>2</td>
</tr>
<tr>
<td>Implemented using Multipliers</td>
<td>and optimized for Speed</td>
</tr>
</tbody>
</table>

The Original Verilog HDL Code in the Vivado Software is:

```verilog
module top(
    input clk,
    input [17:0] a,
    input [17:0] b,
    input ce,
    input sclr,
    output [35:0] p
);
    mymult i1 (    
        .CLK(clk),
        .A(a), // Bus [17: 0]
        .B(b), // Bus [17: 0]
        .CE(ce),
        .SCLR(sclr),
        .P(p)); // Bus [35: 0]
endmodule
```
The original VHDL Code in the Vivado Software is:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all; LIBRARY work;

ENTITY test IS
port (
    clk: IN STD_LOGIC;
    a: IN STD_LOGIC_VECTOR(17 downto 0);
    b: IN STD_LOGIC_VECTOR(17 downto 0);
    sclr: IN STD_LOGIC;
    ce: IN STD_LOGIC;
    p: OUT STD_LOGIC_VECTOR(35 downto 0)
);
END test;

ARCHITECTURE arch OF test IS

component mymult
    PORT(
        CLK: IN STD_LOGIC;
        A: IN STD_LOGIC_VECTOR(17 downto 0);
        B: IN STD_LOGIC_VECTOR(17 downto 0);
        CE: IN STD_LOGIC;
        SCLR: IN STD_LOGIC;
        P: OUT STD_LOGIC_VECTOR(35 downto 0)
    );
end component;

BEGIN
    i1: mymult
        PORT MAP(CLK => clk,
            A => a, B => b, CE => ce,
            SCLR => sclr, P => p);
END;
```

1. In the IP Catalog, select the LPM_MULT IP core to create the equivalent `mymult` module.

 Converted Verilog HDL Code in the Intel Quartus Prime Pro Edition software:

```verilog
module test(
    input clk,
    input [17:0] a,
    input [17:0] b,
    input ce,
    input sclr,
    output [35:0] p
);

mymult i1 ( // Instantiate the LPM_MULT IP core
    .clock(clk),
    .dataa(a), // Bus [17: 0]
    .datab(b), // Bus [17: 0]
    .clken(ce),
    .sclr(sclr),
    .result(p)); // Bus [35: 0]
endmodule
```

Converted VHDL Code in the Intel Quartus Prime Pro Edition Software

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY test IS
    port ( 
        clk: IN STD_LOGIC;
        a: IN STD_LOGIC_VECTOR(17 downto 0);
```
b: IN STD_LOGIC_VECTOR(17 downto 0);
ce: IN STD_LOGIC;
sclr: IN STD_LOGIC;
p: OUT STD_LOGIC_VECTOR(35 downto 0)
);
END test;
ARCHITECTURE arch OF test IS
component mymult
  PORT(clock: IN STD_LOGIC;
       dataa: IN STD_LOGIC_VECTOR(17 downto 0);
       datab: IN STD_LOGIC_VECTOR(17 downto 0);
       clken: IN STD_LOGIC;
       sclr: IN STD_LOGIC;
       result: OUT STD_LOGIC_VECTOR(35 downto 0)
  );
end component;
BEGIN
  i1: mymult
  PORT MAP(clock => clk,
           dataa => a,
           datab => b,
           clken => ce,
           sclr => sclr,
           result => p);
END;

4.2.3.4. Example: Converting to the Intel FPGA Multiply Adder IP core

The following example shows VHDL multipliers compiled in the Intel Quartus Prime Pro Edition Software after the conversion. The IP Catalog implements the Multiply Adder IP core by creating the mymult_add module.

The converted VHDL Code in the Intel Quartus Prime Pro Edition Software is:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY work;
ENTITY test IS
  port
  (  
    clk:IN STD_LOGIC;
    a: IN STD_LOGIC_VECTOR(17 downto 0);
    b: IN STD_LOGIC_VECTOR(17 downto 0);
    ce: IN STD_LOGIC;
    sclr: IN STD_LOGIC;
    p: OUT STD_LOGIC_VECTOR(35 downto 0)
  );
END test;
ARCHITECTURE arch OF test IS
component mymult_add
  PORT(
    clock0: IN STD_LOGIC;
    dataa_0: IN STD_LOGIC_VECTOR(17 downto 0);
    datab_0: IN STD_LOGIC_VECTOR(17 downto 0);
    ena0: IN STD_LOGIC;
    sclr0: IN STD_LOGIC;
    result: OUT STD_LOGIC_VECTOR(35 downto 0)
  );
end component;
BEGIN
  i1: mymult_add
```
4.3. Setting Equivalent Xilinx Design Constraints

Xilinx designs store all the constraints and attributes in Xilinx Design Constraint (.xdc) files, including timing and device constraints. Intel FPGA designs use separate files for device (.qsf) and timing (.sdc) constraints. The Design Constraints section in FPGA Tools Comparison lists the appropriate GUIs to enter design constraints.

Note: Xilinx-based placement constraints do not carry over to Intel FPGA placement constraints. Avoid making placement constraints to a design until you finish the conversion to the Intel Quartus Prime software.

Related Information
- Design Constraints on page 24
- Timing Constraints on page 79

4.3.1. Device Constraints

The following table summarizes the most common Xilinx device constraints and Intel FPGA equivalent device constraints.

Table 56. Intel FPGA Equivalent Device Constraints

<table>
<thead>
<tr>
<th>Xilinx Constraint</th>
<th>Intel FPGA Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIVE</td>
<td>CURRENT_STRENGTH_NEW</td>
<td>Controls the output pin current value</td>
</tr>
<tr>
<td>SLEW</td>
<td>SLEW_RATE</td>
<td>Turns on Fast Slew Rate Control.</td>
</tr>
<tr>
<td>IOB</td>
<td>FAST_INPUT_REGISTER</td>
<td>Specifies whether the Compiler places a register in the device’s IOB.</td>
</tr>
<tr>
<td></td>
<td>FAST_OUTPUT_REGISTER</td>
<td></td>
</tr>
<tr>
<td>IOSTANDARD</td>
<td>IO_STANDARD</td>
<td>Specifies the I/O standard for an I/O pin</td>
</tr>
<tr>
<td>KEEP</td>
<td><em>attribute keep</em> (VHDL)</td>
<td>Prevents a net from either being absorbed by a block or synthesized out.</td>
</tr>
<tr>
<td></td>
<td><em>synthesis keep</em> (Verilog)</td>
<td></td>
</tr>
</tbody>
</table>

To set or modify a device constraint, use the Intel Quartus Prime Assignment Editor. Alternatively, you can edit the .qsf file.

4.3.1.1. DRIVE

Equivalent to the DRIVE constraint in the Xilinx Vivado software, the CURRENT_STRENGTH_NEW logic option sets the drive strength of a pin. You must assign this option to an output or bidirectional pin; otherwise, the Compiler ignores it.
The following example shows how to set the equivalent DRIVE constraint with 12 mA to the output “q1”.

Example of XDC command:

```plaintext
# Set drive strength 12 mA to q1
set_property DRIVE 12 [get_ports q1];
```

Equivalent QSF command:

```plaintext
# Set drive strength 12 mA to q1
set_instance_assignment -name CURRENT_STRENGTH_NEW 12MA -to q1
```

For more information about the current strength feature in the device, refer to the specific device handbook and the Intel Quartus Prime Help.

**Related Information**

Current Strength logic option

In *Intel Quartus Prime Help*

### 4.3.1.2. SLEW

Equivalent to the SLEW constraint in the Xilinx Vivado software, the SLEW_RATE logic option helps to reduce switching noise by controlling low-to-high or high-to-low transitions on output pins. When a large number of output pins switch simultaneously, pins that use the lower SLEW_RATE option help reduce switching noise. This option is only applicable to output or bidirectional pins.

The following example shows how to set the equivalent SLEW constraint to the output “q1”.

Example of XDC command:

```plaintext
# set fast slew rate to q1
set_property SLEW FAST [get_ports q1]
```

Equivalent QSF command:

```plaintext
# set programmable slew rate to q1
set_instance_assignment -name SLEW_RATE 1 -to q1
```

For more information about the slew rate feature in the device, refer to the specific device handbook.

### 4.3.1.3. IOB

Equivalent to the IOB constraint in Xilinx, the FAST_INPUT_REGISTER and FAST_OUTPUT_REGISTER logic options directs the Compiler to implement an input register and output register in an I/O cell that has a fast, direct connection from an I/O pin.

The following example shows how to set the equivalent IOB constraint to the input “d1” or the output “q1”.

```plaintext
```
Example of XDC command:

```cpp
# Set IOB to input d1
set_property IOB TRUE [get_ports d1];

# Set IOB to output q1
set_property IOB TRUE [get_ports q1];
```

Equivalent QSF command:

```cpp
# Set FAST_INPUT_REGSITER to input d1
set_instance_assignment -name FAST_INPUT_REGISTER ON -to d1

# Set FAST_OUTPUT_REGSITER to output q1
set_instance_assignment -name FAST_OUTPUT_REGISTER ON -to q1
```

For more information about the slew fast input and output register features in the device, refer to the specific device handbook and the Intel Quartus Prime Help.

**Related Information**

- Fast Input Register logic option
  In [Intel Quartus Prime Help](#)
- Fast Output Register logic option
  In [Intel Quartus Prime Help](#)

### 4.3.1.4. IOSTANDARD

Equivalent to the IOSTANDARD constraint in Xilinx, the IO_STANDARD logic option uniquely defines the input and output (VCCIO) voltage, reference VREF voltage (if applicable), and the types of input and output buffers used for I/O pins.

The following example shows how to set the equivalent IOSTANDARD constraint (Differential SSTL-2 Class I) to the “q2” output.

**Example XDC command:**

```cpp
# Set Differential SSTL18_I I/O Standard to q2
set_property IOSTANDARD SSTL18_I [get_ports q2];
```

**Equivalent QSF command:**

```cpp
# Set Differential SSTL-18 Class I I/O Standard to q2
set_instance_assignment -name IO_STANDARD "SSTL-18 CLASS I" -to q2
```

**Related Information**

- I/O Standard logic option
  In [Intel Quartus Prime Help](#)

### 4.3.1.5. KEEP

Equivalent to the KEEP constraints, the Attribute Keep (VHDL) or Synthesis Keep (Verilog) synthesis attributes direct the Compiler to keep a wire or combinational node through logic synthesis minimizations and netlist optimizations. Similarly, you can also set the Implement as Output of Logic Cell logic option in the Intel Quartus Prime Assignment Editor.
The following example shows how both VHDL and Verilog HDL set the equivalent `KEEP` constraint (Differential SSTL-2 Class I) to the `my_wire` signal.

Verilog HDL example in the Vivado software:

```verilog
(* KEEP = "TRUE" *) wire my_wire
```

Equivalent Verilog HDL example in the Intel Quartus Prime software:

```verilog
(    *preserve*) wire my_wire;
```

VHDL example in the Vivado software:

```vhdl
signal my_wire: bit;
attribute keep: string;
attribute keep of my_wire: signal is "TRUE";
```

Equivalent VHDL example in the Intel Quartus Prime software:

```vhdl
signal my_wire: bit;
attribute syn_keep: boolean;
attribute syn_keep of my_wire: signal is true;
```

**Related Information**

- **Preserve Registers During Synthesis**
  In *Intel Quartus Prime Pro Edition Handbook Volume 1*
- **keep VHDL Synthesis Attribute**
  In *Intel Quartus Prime Help*
- **keep Verilog HDL Synthesis Attribute**
  In *Intel Quartus Prime Help*

### 4.3.2. Placement Constraints

The following table compares the most common Xilinx placement constraints with the Intel FPGA equivalent placement constraints:

**Table 57. Placement Constraints**

<table>
<thead>
<tr>
<th>Xilinx Constraint</th>
<th>Intel FPGA Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assignment Name</strong></td>
<td><strong>QSF Variable</strong></td>
<td></td>
</tr>
<tr>
<td>PLOCK</td>
<td>Logic Lock Region</td>
<td>CORE_ONLYPLACE_REGION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FLOATING_REGION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PLACE_REGION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REGION_NAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESERVEPLACE_REGION</td>
</tr>
</tbody>
</table>

*continued...*
<table>
<thead>
<tr>
<th>Xilinx Constraint</th>
<th>Intel FPGA Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Assignment Name</td>
<td>QSF Variable</td>
</tr>
<tr>
<td>ROUTE_REGION</td>
<td></td>
<td>ROUTE_REGION</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACKAGE_PIN &lt;Pin Number&gt;</td>
<td>Location Assignment</td>
<td>PIN_&lt;Pin number&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC (for primitive cell such as SLICE, RAMB)</td>
<td>Location Assignment</td>
<td>&lt;Location&gt; &lt;Value&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEL (for registers, LUT, SRL, LUTRAM)</td>
<td>Location Assignment</td>
<td>&lt;Location&gt; &lt;Value&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROHIBIT</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

To set or modify placement constraints, use the Intel Quartus Prime Assignment Editor. Alternatively, you can edit the .qsf file.

**Related Information**

Viewing and Editing Design Placement on page 31

### 4.3.2.1. PBLOCK

Equivalent to the PBLOCK constraint in the Xilinx Vivado software, Logic Lock regions are floorplan location constraints in the Intel Quartus Prime Pro Edition software. Logic Lock region assignments have Placement and Routing Regions.

The following example shows how to set a module's attributes in a XDC file, using:

- **EXCLUDE_PLACEMENT**—directs the Fitter to place only pblock's logic in the device resources within the pblock region.
- **CONTAIN_ROUTING**—directs the Fitter to route signals in the pblock area using only resources available within the pblock area.

In this example, the module does not contain any I/O resources.

**XDC Command:**

```plaintext
create_pblock pblock_uut_inst
add_cells_to_pblock [get_pblocks pblock_uut_inst] [get_cells -quiet [list uut_inst]]
resize_pblock [get_pblocks pblock_uut_inst] -add (SLICE_X218Y284:SLICE_X220Y346)
resize_pblock [get_pblocks pblock_uut_inst] -add (RAMB18_X12Y114:RAMB18_X12Y137)
resize_pblock [get_pblocks pblock_uut_inst] -add (RAMB36_X12Y57:RAMB36_X12Y68)
set_property CONTAIN_ROUTING 1 [get_pblocks pblock_uut_inst]
set_property EXCLUDE_PLACEMENT 1 [get_pblocks pblock_uut_inst]
```

**Equivalent QSF Command:**

```plaintext
set_instance_assignment -name PLACE_REGION "X1 Y1 X20 Y20" -to uut_inst
set_instance_assignment -name RESERVE_PLACE_REGION ON -to uut_inst
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to uut_inst
set_instance_assignment -name REGION_NAME uut_inst -to uut_inst
set_instance_assignment -name ROUTE_REGION "X1 Y1 X20 Y20" -to uut_inst
```
REGION_NAME creates an Logic Lock region named “uut_inst”, and assigns module uut_inst to this region. This is similar to XDC’s CREATE_PBLOCK and ADD.Cells_TO_PBLOCK assignments.

```plaintext
set_instance_assignment -name REGION_NAME uut_inst -to uut_inst
```

PLACE_REGION defines the coordinates for the placement region, similar to XDC’s Resize_Pblock assignments.

```plaintext
set_instance_assignment -name PLACE_REGION "X1 Y1 X20 Y20" -to uut_inst
```

RESERVE_PLACE_REGION prevents the Fitter from placing other logic in the region, equivalent to XDC’s EXCLUDE_PLACEMENT assignment.

```plaintext
set_instance_assignment -name RESERVE_PLACE_REGION ON -to uut_inst
```

Defining ROUTE_REGION with the same coordinates as PLACE_REGION indicates the routing to remain within the placement region, equivalent to XDC’s CONTAIN_ROUTING assignment.

```plaintext
set_instance_assignment -name ROUTE_REGION "X1 Y1 X20 Y20" -to uut_inst
```

By default, Logic Lock assignments include periphery resources, and you prevent this behavior with the CORE_ONLY_PLACE_REGION logic option. However, in XDC, the default behavior is to exclude periphery resources, so you must apply pblock constraints to include periphery resources in the pblock region.

```plaintext
set_instance_assignment -name CORE_ONLY_PLACE_REGION ON -to uut_inst
```

**Related Information**

Logic Lock Assignment Examples

In *Intel Quartus Prime Pro Edition Handbook Volume 1*

### 4.3.2.1.1. Differences Between PBLOCK and Logic Lock Regions

The default behavior of PBLOCK Logic Lock regions presents one important difference:

- When you assign a module to a Logic Lock region, the resources within the Logic Lock region become available to the module automatically. In the Vivado software, you must manually assign the resources that the module can use to the PBLOCK region.

### 4.3.2.2. PACKAGE_PIN

Equivalent to the PACKAGE_PIN constraint in the Xilinx Vivado software, PIN_<<pin number>> is the pin location constraint assignment that the Intel Quartus Prime Pro Edition uses.

The following example shows how to set the location for a clock pin on the device.

Example of XDC Command:

```plaintext
# Assign location for the clock pin
set_property PACKAGE_PIN B26 [get_ports clock]
```
Equivalent QSF Command:

```plaintext
# Assign location for the clock pin
set_location_assignment PIN_AU33 -to clock
```

### 4.3.2.3. LOC & BEL

In the Xilinx Vivado software, the LOC constraint specifies the placement of a logic cell to a specific SLICE, and the BEL constraint specifies the placement of a leaf cell within the SLICE. The Xilinx Vivado software uses the LOC & BEL constraints to place a register or LUT or SRL or memory to a specific location. An equivalent constraint in the Intel Quartus Prime Pro Edition software is `<Location> -to <value>`.

The following example shows how to constraint the location for a current node on the device.

Example of XDC Command:

```plaintext
# Assign location for an internal register
set_property LOC SLICE_X0Y0 [get_cells uut_inst/dout_reg]
```

Equivalent QSF Command:

```plaintext
# Assign location for an internal register
set_location_assignment FF_X60_Y119_N55 -to uut_inst|dout_reg
```

### 4.3.2.4. PROHIBIT

PROHIBIT specifies the BEL or LOC on a Xilinx device where placement is prohibited. An equivalent constraint is not available in the Intel Quartus Prime Pro Edition Software.

### 4.3.3. Timing Constraints

You can convert constraints defined in XDC files to SDC commands that the Intel Quartus Prime Pro Edition Timing Analyzer can use.

The following table summarizes the most common Vivado XDC timing constraints and the equivalent SDC timing constraints. You can set the constraints by either modifying the .sdc file or by using the Timing Analyzer GUI.

<table>
<thead>
<tr>
<th>Vivado XDC Timing Constraint</th>
<th>Timing Analyzer SDC Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create_clock</td>
<td>create_generated_clock</td>
<td>Defines all the clocks and their relationship in a design.</td>
</tr>
<tr>
<td>set_max_delay</td>
<td>set_false_path</td>
<td></td>
</tr>
</tbody>
</table>

| NA                           | derive_pll_clocks           | Automatically creates a generated clock constraint on each output of the PLLs in a design. Note: Only Intel Arria 10 devices support this command |

| NA                           | derive_clock_uncertainty    | Calculates clock-to-clock uncertainties within the FPGA due to characteristics like PLL jitter, clock tree jitter, etc. The Timing Analyzer generates a warning if the command is not present in the SDC files |

Table 58. Vivado XDC versus SDC Timing Constraints

```
```
### Vivado XDC Timing Constraint | Timing Analyzer SDC Command | Description
--- | --- | ---
set_input_delay | set_input_delay | Input timing constraint that you use to define the Pad-to-Setup timing requirement in a design.
set_output_delay | set_output_delay | Defines the global Clock to Pad timing requirement in a design.
s set_max_delay | set_max_delay | Combinational path that constrains all combinational pin to pin paths.
set_false_path | set_false_path | Eliminates the paths from timing consideration during Place and Route and timing analysis.
set_clock_groups w/ include_generated_clocks | set_clock_groups | Cuts timing between clocks in different groups.
NA | derive_pll_clocks | Vivado tools auto-derive the generated clocks; therefore, XDC files do not require these constraints.
NA | derive_clock_uncertainty |


**Related Information**

**The Intel Quartus Prime Timing Analyzer**

In *Intel Quartus Prime Pro Edition Handbook Volume 3*

#### 4.3.3.1. set_clock_groups

In the Xilinx Vivado software, the set_clock_groups constraint supports an additional switch named include_generated_clocks, to include generated clocks of a specific member in a clock group. However, in Intel Quartus Prime Pro Edition, you must add the generated clocks by name.

**Example XDC command:**

```plaintext
# Assign adc_clk, clocks generated from adc_clk and sys_clk, clocks generated from sys_clk to different clock groups
set_clock_groups -asynchronous
- group [get_clocks -include_generated_clocks adc_clk] 
- group [get_clocks -include_generated_clocks sys_clk]
```

**Equivalent SDC command:**

```plaintext
# Assign adc_clk, clocks generated from adc_clk and sys_clk, clocks generated from sys_clk to different clock groups
set_clock_groups -asynchronous
- group [get_clocks {adc_clk
the_adc_pll|Intel FPGA IOPLL_component_autogenerated|pll|clk[0]
the_adc_pll|Intel FPGA IOPLL_component_autogenerated|pll|clk[1]
the_adc_pll|Intel FPGA IOPLL_component_autogenerated|pll|clk[2]
}]
- group [get_clock {sys_clk
the_system_pll|Intel FPGA IOPLL_component_autogenerated|pll|clk[0]
the_system_pll|Intel FPGA IOPLL_component_autogenerated|pll|clk[1]
}]
```
4.3.4. Retimer Constraints

In the Intel Quartus Prime Pro Edition Software, the Fitter’s Retime stage moves (retimes) existing registers into Hyper-Registers for fine-grained performance improvement (available only in Intel Stratix 10). Xilinx devices do not have Hyper-Registers in their architecture, hence the existing Vivado based designs do not have equivalent constraints.

**Related Information**

Hyper-Aware Design Flow on page 41

4.4. Setting Up the Simulation Environment

Intel Quartus Prime Pro Edition software supports RTL and gate-level design simulation in the EDA simulators listed in the table. Unless you use a simulator specific to Xilinx, the simulation environment in the Intel Quartus Prime Pro Edition is similar. The Xilinx environment also supports all the following EDA simulators:

**Table 59. Supported Simulators**

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Simulator</th>
<th>Version</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aldec*</td>
<td>Active-HDL*</td>
<td>10.3</td>
<td>Windows*</td>
</tr>
<tr>
<td>Aldec</td>
<td>Riviera-PRO*</td>
<td>2016.10</td>
<td>Windows, Linux</td>
</tr>
<tr>
<td>Cadence*</td>
<td>Incisive Enterprise*</td>
<td>15.20</td>
<td>Linux</td>
</tr>
<tr>
<td>Mentor Graphics</td>
<td>ModelSim - Intel FPGA Edition</td>
<td>10.5c</td>
<td>Windows, Linux</td>
</tr>
<tr>
<td>Mentor Graphics</td>
<td>ModelSim PE</td>
<td>10.5c</td>
<td>Windows</td>
</tr>
<tr>
<td>Mentor Graphics</td>
<td>ModelSim SE</td>
<td>10.5c</td>
<td>Windows, Linux</td>
</tr>
<tr>
<td>Mentor Graphics</td>
<td>QuestaSim*</td>
<td>10.5c</td>
<td>Windows, Linux</td>
</tr>
<tr>
<td>Synopsys</td>
<td>VCS*</td>
<td>2016,06-SP-1</td>
<td>Linux</td>
</tr>
<tr>
<td></td>
<td>VCS MX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For more information about ModelSim - Intel FPGA Edition Products, refer to the Altera website.

For more information about supported simulation levels, refer to the *Intel Quartus Prime Pro Edition Handbook Volume 3*.

**Related Information**

- ModelSim - Intel FPGA Edition
- Simulation Quick-Start for ModelSim - Intel FPGA Edition (Intel Quartus Prime Pro Edition)

4.4.1. Simulation Levels

The Intel Quartus Prime Pro Edition software supports RTL and gate-level simulation in the supported EDA Simulators.

If you use the ModelSim - Intel FPGA Edition Simulator in a design that includes deep levels of hierarchy, turn off the Maintain hierarchy EDA tools option. This action prevents the Compiler to generate a large number of module instances in post-fit or
post-map netlist, thus exceeding the ModelSim - Intel FPGA Edition instance limitation. To access this option, click Assignments ➤ Settings ➤ EDA Tool Settings ➤ More Settings.


For information about supported simulation levels, refer to the Intel Quartus Prime Pro Edition Handbook Volume 3.

**Related Information**
**Simulation Levels**
In *Intel Quartus Prime Pro Edition Handbook Volume 3*

### 4.4.2. HDL Support for EDA Simulators

The Intel Quartus Prime Pro Edition software provides the following HDL support for EDA simulators:

- Verilog-2001 (IEEE Standard 1364-2001)

### 4.4.3. Value Change Dump (VCD) Support

All the EDA simulators that the Intel Quartus Prime Pro Edition software support, besides other third-party simulators can generate .vcd files. The Intel Quartus Prime Pro Edition Power Analyzer tool can read .vcd files for power measurements.

### 4.4.4. 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.

**Related Information**
**Simulating Intel FPGA IP Cores**
In *Intel Quartus Prime Pro Edition Handbook Volume 3*
5. Conclusion

The Intel Quartus Prime software provides a complete design environment that you can easily adapt to your design for the development of Intel FPGAs and CPLD devices.

Programmable logic design and compilation flow is very similar between Intel Quartus Prime software and Xilinx Vivado software, and in most cases, you can easily import a Vivado design into the Intel Quartus Prime software design environment. You can improve your design conversion experience by following the design conversion guidelines and considerations discussed in this application note, including migrating a design targeted at a Xilinx device to one that is compatible with an Intel FPGA device.
# 6. Document Revision History for Intel FPGA Design Flow for Xilinx Users

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020.08.24</td>
<td>17.1.0</td>
<td>• Added a link to Converting Xilinx RAM initialization .coe/.mif Format to Intel PSG .mif/.hex Format in the Memory Initialization topic.</td>
</tr>
</tbody>
</table>
| 2018.03.20       | 17.1.0                     | • Revised content for Intel Quartus Prime Pro Edition software version 17.1 and Xilinx Vivado Design Suite version 2017.2.  
                             • Added chapter that compares latest devices.  
                             • Updated for latest Intel naming conventions. |

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| May 2015         | 2015.05.11 | • Updated content for DSE II  
                             • Updated GUI screenshots to v15.0  
                             • Removed obsolete devices not supported in v15.0  
                             • Removed MegaWizard Plug-in Manager content and replaced with IP Catalog/Parameter Editor content  
                             • Updated links |
| March 2013       | 7.0     | • Revised content for software versions ISE 14.2 and Quartus II 12.1  
                             • Removed outdated design examples.  
                             • Updated template |
| November 2009    | 6.2     | Corrected set_max_delay constraint equivalents for OFFSET IN BEFORE and OFFSET OUTPUT AFTER UCF commands in Timing Constraints section. |
| April 2009       | 6.2     | Added Appendix A: Design Example and Appendix B |
| July 2008        | 6.0     | Revised and restructured content for software versions ISE 10.1 and Quartus II 8.0 |
| June 2005        | 5.0     | • Revised content for software versions ISE 7.1 and Quartus II 5.0  
                             • Updated terminology  
                             • Added Pin Planner subsection  
                             • Added Quartus II Incremental Compilation |

*Other names and brands may be claimed as the property of others.*
<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 2004</td>
<td>4.0</td>
<td>• Revised content for software versions ISE 6.3i and Quartus II 4.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated Table 6 for Power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated cross-probing chart</td>
</tr>
<tr>
<td>January 2004</td>
<td>3.1</td>
<td>Updated terminology</td>
</tr>
<tr>
<td>October 2003</td>
<td>3.0</td>
<td>• Revised content for software versions ISE 6.2i and Quartus II 4.1 sp2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added information on cross-probing</td>
</tr>
<tr>
<td>July 2003</td>
<td>2.0</td>
<td>• Revised content for software versions ISE 5.1i and Quartus II 3.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added information on the Quartus II modular executables and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>command-line scripting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added information on DDR RAM conversions</td>
</tr>
<tr>
<td>November 2002</td>
<td>1.0</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>